Diplom

Entwicklung von Dokumentfamilien auf Basis von Featuremodellen und dem Open Document Format

bearbeitet von

Martin Heinzerling
geboren am 11. April 1985 in Merseburg

Technische Universität Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie

Betreuer: Dipl.-Inf. Sven Karol
Hochschullehrer: Prof. Dr. rer. nat. habil. Uwe Aßmann

Eingereicht am 15. Mai 2010

Technische Universität Dresden
Fakultät Informatik

AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT

Name des Studenten: **Martin Heinzerling**

Immatrikulations-Nr.: **3208101**

Thema: **Entwicklung von Dokument Familien auf Basis von Featuremodellen und dem Open Document Format**

Zielstellung:

Das Open Document Format (ODF) ist ein standardisiertes, XML basiertes Format für verschiedene Arten von Office Dokumenten. Dies umfasst Textdokumente, Tabellen, Grafiken und Präsentationen bzw. Vorträge. Zwar wird durch XML prinzipiell die Einbettung von XML Dokumenten durch XInclude unterstützt, jedoch kann diese nicht speziell auf ODF Dokumente beschränkt werden und eignet sich insbesondere gut für hierarchische Dekomposition. Es lassen sich also beispielsweise verschiedene Varianten eines Dokuments für verschiedene Adressaten mit unterschiedlichem Kenntnisstand nur schwer realisieren.
In der Softwaretechnik lassen sich dagegen verschiedene Varianten einer Anwendung insgesamt als eine Produktlinie auffassen. Eine konkrete Variante wird dann "Instanz" der Produktlinie genannt. Zur Modellierung von Produktlinien können Featuremodelle verwendet werden, welche es erlauben den Variantenraum mit Hilfe hierarchisch angeordneter Belange (*Concerns* bzw. *Features*) darzustellen, die mit Bedingungen (*Constraints*) zur Auswahl von Varianten belegt werden können. Das Werkzeug FeatureMapper erlaubt die Erstellung von Abbildungen (*Mappings*) zwischen Belangen in Featuremodellen und Modellen im Artefaktraum sowie die Instanziierung und Visualisierung von Varianten über dem Artefaktraum.
Prinzipiell lässt sich dieser Ansatz auch auf XML Dokumente und damit ODF und Office Suiten wie OpenOffice übertragen. Dies genauer zu untersuchen ist der Kern dieser Arbeit. Mögliche Belange eines Dokuments sind zum Beispiel Umfang (Lang- bzw. Kurzfassung), Darstellung, homogene Aspekte wie Kopf- und Fußzeile und natürlich verschiedene inhaltliche Aspekte sein.

Fortsetzung Rückseite

Betreuer: Dipl.-Inf. Sven Karol

Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie

Beginn am: 15.11.2009
Einzureichen am: 15.05.2010

Dresden, 03.11.2009

Prof.Dr.rer.nat.habil. U. Aßmann
verantwortl. Hochschullehrer

Fortsetzung Zielstellung:

Die Arbeit umfasst damit folgende Punkte (gute Kenntnisse von XML und des Eclipse Modeling Frameworks sollten gegeben sein):

- Einarbeitung in das Open Document Format und das Komponentenmodell von OpenOffice
- Einarbeitung in Softwareproduktlinien, Featuremodelle und das Feature-Mapper Werkzeug
- Evaluation der Machbarkeit einer Integration des FeatureMappers mit dem Open Document Format und OpenOffice
    - ggf. deren prototypische Realisierung
    - andernfalls die Übertragung des FeatureMapper Ansatzes
- Erstellung einer komplexen Fallstudie, z.B., eine Produktlinie von Vorlesungsfolien
- Erstellung einer Featureklassifikation für Dokumentproduktlinien
- Untersuchung und Bewertung, ob und inwieweit sich Featuremodelle eigenen um Familien von Dokumenten zu modellieren
- Testen der Implementierung mit einer geeigneten Testmethodik

Die Literaturrecherche ist selbstständig durchzuführen.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Martin Heinzerling
Dresden, den 15. Mai 2010

Zusammenfassung

In den vergangenen Jahrzehnten gewannen digitale Dokumente zunehmend an Bedeutung. Mit der wachsenden Anzahl an Dokumenten gibt es immer wieder Szenarien, in denen neue Dokumente aus der Abwandlung von bereits vorhandenen entstehen. Im ersten Teil dieser Arbeit wird diese Motivation weiter verfolgt und daraus resultierende Schwierigkeiten erörtert. Darauf aufbauend folgt eine Einführung in Techniken, die bisher bei der Erstellung von Softwareproduktlinien Anwendung finden, und es wird die Übertragbarkeit auf Dokumentfamilien aufgezeigt. Mit diesen Erkenntnissen konnte ein Werkzeug entwickelt werden, welches dann zunächst vorgestellt und anschließend einer Untersuchung der Praxistauglichkeit an realen Anwendungsfällen unterzogen wird. Eine Bewertung des Ansatzes und eine Zusammenfassung schließen die Arbeit ab.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das erste Kapitel gibt zunächst einen kurzen Überblick über die Motivation und die Zielsetzung dieser Arbeit. Im Weiteren soll auch ein Einblick in verwandte Thematiken und Arbeiten gegeben werden.

1.1 Motivation

Digitale Dokumente gewannen in den letzten Jahrzehnten zunehmend an Bedeutung. Die Möglichkeit der einfachen Bearbeitung und Erweiterung existierender Dokumente erlaubt es in vielen Szenarien, neue Dokumente auf der Basis bereits vorhandener zu erstellen. Im einfachsten Fall existieren Vorlagen, die kopiert und für den einmaligen Gebrauch vervollständigt werden. Dabei treten im Allgemeinen auch keine Probleme in der Verwaltung der Dokumente auf. Deutlich unangenehmer sind Szenarien, in denen Änderungen am Originaldokument auf alle daraus abgeleiteten Dokumente angewendet werden müssen. Dies beginnt bei einzelnen Rechtschreibfehlern, die vor dem Kopieren übersehen wurden, kann aber auch einen deutlich größeren Aufwand umfassen, wenn z. B. ein Corperate Design zuvor auf eine solche Weise vervielfacht wurde. Weitere konkrete Szenarien werden im Kapitel 4 vorgestellt, aber grundlegend können alle Dokumente, die durch „Kopieren und Anpassen“ entstehen, zu einem solchen Aktualisierungsaufwand führen.

Dieser Aktualisierungsaufwand, der zunächst nur ein zeitliches Problem ist, führt aber in vielen Fällen auch zu einem Motivationsproblem der Person, die die Änderungen von Hand übertragen muss. Ab einem gewissen Umfang an Änderungen oder einer größeren Menge von Dokumenten bleibt es nicht aus, dass Einzelne nicht mehr aktualisiert werden. Über mehrere Iterationen von Änderungen existieren somit nicht nur eine Vielzahl von Varianten des aktuellen Quelldokuments, hinzu kommen auch noch Varianten älterer Versionen, womit sich der Verwaltungsaufwand ständig erhöht und dadurch mehr „fehlerhafte“ Dokumente in den Umlauf kommen.

Zur Lösung des analogen Problems in der Softwareentwicklung und zur Steigerung der Effizienz wurde Mitte der neunziger Jahre der Begriff der Softwareproduktlinien (vgl. Abschnitt 2.1.1) eingeführt, dessen Übertragbarkeit auf Dokumentfamilien in dieser Arbeit für einen konkreten Ansatz untersucht werden soll. Der Erstellung von Dokumenten sollen ebenfalls, wie der Entwicklung von Software, die Prinzipien „Single Source“ und DRY[1](Hunt und Thomas, 1999) zu Grunde liegen.

1.2 Zielsetzung

Der allgemein gehaltene Gedanke des vorherigen Abschnitts soll für die folgenden Betrachtungen zunächst etwas eingeschränkt werden. Wenn im Weiteren der Begriff *Dokument* verwendet wird, bezieht sich dies stets auf ein Dokument im Open Document Format (Abschnitt 2.1.3).

Dokument

[1] Don’t Repeat Yourself

Weiterhin soll durch den eingeschränkten zeitlichen Rahmen einer Diplomarbeit nur eine mögliche Art der Erstellung von Produktlinien bzw. Dokumentfamilien betrachtet werden. Die Wahl ist hier auf Featuremodelle (Abschnitt 2.1.2) gefallen.

Dementsprechend gilt es zu untersuchen, ob ein Dokument im Open Document Format mit Hilfe eines Featuremodells als Ausgangspunkt für eine Dokumentfamilie dienen kann. Dazu ist vorab festzustellen, ob das bereits vorhandene Werkzeug, der FeatureMapper (Abschnitt 2.2.2) geeignet ist, bzw. alternativ ein geeignetes Werkzeug zu entwerfen. Anschließend soll das Erzeugen von Dokumentfamilien an konkreten Anwendungsbeispielen gezeigt und ein allgemeinerer Ansatz für Featureklassifikationen von Dokumentproduktlinien abgeleitet werden.

1.3 Verwandte Arbeiten

Neben dem in Abschnitt 2.2.2 vorgestellten FeatureMapper der Technischen Universität Dresden gibt es mittlerweile eine Vielzahl von Programmen, die sich auf die eine oder andere Weise mit Featuremodellen oder Softwareproduktlinien auseinander setzen. Im Rahmen dieser Arbeit sei nur die folgende exemplarische Auflistung[2] gegeben. Dennoch wäre es sinnvoll, in einer nachfolgenden Arbeit sämtliche Ansätze und Zielstellungen detailliert zu vergleichen und daraus Verwendungsvorschläge für das eine oder andere Werkzeug abzuleiten.

> Ahead Tool Suite[3], Eclipse Modeling Framework Feature Model Project[4], FaMa Tool Suite[5], Feature Model Plug-in[6], FeatureIDE[7], Hydra[8], MOSKitt Feature Modeler[9], Pure::Variants[10], Requiline[11], S2T2 Configurator[12], SPLOT (Software Product Line Online Tools)[13] und XFeature[14].

Mit einem verwandten Ansatz zur vorliegenden Arbeit beschäftigt sich das Eclipse-Subprojekt *doc2model*[15]. Schwerpunkt ist hier das Extrahieren von In- doc2model

[2] kein Anspruch auf Vollständigkeit, ohne weitere Wichtung alphabetisch sortiert
[3] http://www.cs.utexas.edu/users/schwartz/ATS.html
[4] http://www.eclipse.org/proposals/feature-model/
[5] http://www.isa.us.es/fama/
[6] http://gsd.uwaterloo.ca/projects/fmp-plugin/
[7] http://fosd.de/fide/
[8] http://caosd.lcc.uma.es/spl/hydra/index.htm
[9] http://oomethod.dsic.upv.es/labs/index.php
?option=com_content&task=view&id=51&Itemid=35
[10] http://www.pure-systems.com/Variant_Management.49.0.html
[11] http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline/
[12] http://download.lero.ie/spl/s2t2/
[13] http://www.splot-research.org/
[14] http://www.pnp-software.com/XFeature/

formationen aus vorhanden Dokumenten basierend auf deren Erscheinungsbild bzw. Kontext. Das erhaltene EMF-Modell steht dann für weitere Bearbeitungen zur Verfügung und könnte mit dem FeatureMapper der Technischen Universität Dresden in Varianten modifiziert werden. Die Rücktransformation in ein Dokument ist aber nur dann möglich, wenn sämtliche Informationen aus dem Quelldokument auch in das EMF-Modell extrahiert werden, welches der eigentlichen Intention des Projekts, Daten aus einem Dokument in ein kompaktes Modell zu überführen, entgegen steht. Somit ist dieser Ansatz für die Zielstellung dieser Arbeit zu spezialisiert.

Bereits in der vorangegangenen Arbeit (Kopcsek, 2007) wurde eine hybride Kombination aus subtraktiven und additiven Verfahren der Featuremodellierung angestrebt (vgl. Abschnitt 2.1.2). Im Rahmen des Forschungsprojekts *Hyperadapt* (Niederhausen u. a., 2009) werden derzeit die Möglichkeiten einer Aspektorientierung von XML-Dokumenten untersucht. Die Schwerpunkte liegen dabei vor allem auf dem typsicheren Verweben von Dokumenten und der Adaption von Webdokumenten an den Kontext des Nutzers mittels Adaptionsregeln, die als Aspekte gekapselt werden.

Hyperadapt

Ansätze zur Anwendung von Featuremodellen auf Dokumente konnten in der Literatur nicht gefunden werden. Bisherige Betrachtungen zur Erzeugung von Dokumentfamilien konzentrieren sich meist auf eine konkrete Anwendung. So wird zum Beispiel in (Harris, 1997) ein additiver Ansatz basierend auf feingranularen Teildokumenten in Verbindung mit dem Adobe FrameMaker vorgestellt.

[15] http://www.eclipse.org/proposals/doc2model/

Kapitel 2

Vorbetrachtung und Einordnung

Das folgende Kapitel bietet im ersten Teil eine Übersicht über im Weiteren verwendete Technologien und Werkzeuge (Abschnitt 2.1 und 2.2). Jeder dieser Abschnitte kann nach eigenem Ermessen übersprungen werden und soll nur einen kurzen Einstieg zur Einordnung der Technologie sowie Hinweise zu weiterführender Literatur bieten.

Der zweite Teil fasst Vor- und Nachteile (Abschnitt 2.3) der einzelnen Technologien zusammen und leitet daraus eine Entscheidung für die konkrete Implementierung zur Lösung der Zielstellung ab. (Abschnitt 2.4)

2.1 Technologien

2.1.1 Softwareproduktlinien

Softwareproduktlinie

Der Begriff der *Softwareproduktlinien* findet erstmalig in (Brownsword u. a., 1996) Beachtung. Bis heute wird dieses Konzept im Carnegie Mellon Software Engineering Institute weiter verfolgt. Ein Schwerpunkt liegt in den letzten Jahren auf einer zunehmenden Bekanntmachung[1] der Möglichkeiten von Softwareproduktlinien in der Industrie.

Softwareproduktlinien greifen dabei den Grundgedanken der klassischen Produktlinien der Industrie auf und versuchen, Produkte auf einer gemeinsamen Basis (*Plattform*) zusammenzufassen. Dies geht aber weit über das einfache Wiederverwenden von Komponenten hinaus und wird meist unter dem Begriff „strategic reuse“ zusammengefasst. Es entsteht also nicht zuerst ein Produkt, aus dem dann einzelne Komponenten oder Module zur möglichen Wiederverwendung extrahiert werden, sondern im Blickfeld des gesamten Entwicklungsprozesses steht die komplette Architektur aller Produkte einer Produktline.

Plattform

Nach dieser Definition sind die folgenden Anwendungen keine Softwareproduktlinien: (Northrop, 2009, S. 24)

- Wiederverwenden von Code für ein einzelnes System durch Kopieren
- Wiederverwendung durch Bibliotheken mit Algorithmen, Modulen, Komponenten oder Objekten
- Ausschließlich Komponenten- oder Serviceorientierte Entwicklung
- Ausschließlich Versionen eines einzigen Produktes
- Ausschließlich eine konfigurierbare Architektur
- Ausschließlich eine Menge von technischen Standards

All diesen Ansätzen fehlt die zwingend notwendige Fokussierung auf eine ganzheitliche Architektur. Eine mögliche Wiederverwendung und die daraus resultierenden Erfolge hängen wesentlich von den Softwareingenieuren ab, die diese Techniken anwenden. Darüber hinaus findet keine Analyse der gemeinsamen Plattform mehrerer Produkte statt.

Für die Erstellung einer Softwareproduktline existieren mehrere Ansätze, die hier nur kurz aufgelistet werden sollen. Beim *proaktiven* Vorgehen wird zunächst die Plattform entwickelt, und die ersten Produkte besitzen noch einen geringeren Funktionsumfang. Dies setzt ein fundiertes Wissen über sämtliche Produkte voraus, um die gemeinsame Plattform korrekt zu bestimmen. Dem gegenüber

[1] http://www.sei.cmu.edu/productlines/

steht der *reaktive* Ansatz, der zunächst mit wenigen vollständigen Produkten beginnt und daraus die Plattform ableitet. In der Praxis lassen sich diese beiden Ansätze zu einem *inkrementellen* Vorgehen zusammenfassen, bei dem die Plattform schrittweise mit den neuen Produkten wächst und adaptiert wird.

Einen umfassenden Einstieg und einige Beispiele sind in (Walter, 2002) und (Northrop, 2009) gegeben.

2.1.2 Featuremodelle

Featuremodelle

Featuremodelle, im Deutschen auch gelegentlich als Merkmalsmodelle bezeichnet, stammen ursprünglich aus der *Feature-orientierten Domainanalyse* (FODA). Seit der ersten Veröffentlichung in (Kang u. a., 1990) nehmen die Verbreitung und die Anwendungsgebiete stetig zu. Sowohl die „Feature-Oriented Reuse Method“ (Kang u. a., 2002) als auch das „Feature-Oriented Product Line Engineering“ (Kang u. a., 1998) greifen Featuremodelle auf.

FODA

Nach (Kang u. a., 1990, S. 1) sind *Features* herausstechende bzw. markante Merkmale eines (Software-)Systems. Sie sind entweder für den Benutzer sichtbar oder charakterisieren das Anwendungsgebiet. Features fassen also Gemeinsamkeiten zusammen oder bieten die Möglichkeit, ähnliche Systeme an diesen zu unterscheiden und abzugrenzen.

Feature

In dieser Arbeit findet eine Erweiterung des ursprünglichen Konzepts ihre Anwendung. Mit den *kardinalitätsbasierenden Featuremodellen* (Czarnecki u. a., 2005) und der Ergänzung um weitere Nebenbedingungen (*Constraints*) (Czarnecki und Kim, 2005) kann sowohl die Ausdrucksstärke erhöht als auch die Verwendung der Kernfunktionalität vereinfacht werden.

Kardinalitätsbasierende Featuremodelle
Constraints

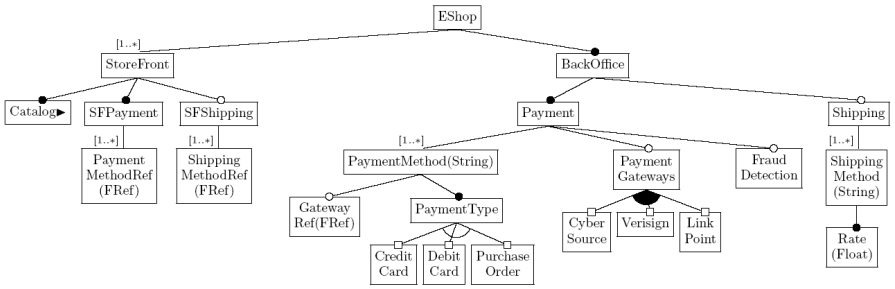


Abbildung 2.1
Beispiel Featuremodell (Czarnecki u. a., 2005, S. 2)

Abbildung 2.1 zeigt ein exemplarisches Featuremodell. Das Modell beschreibt einen Laden, der aus mindestens einer Abteilung mit Kundenkontakt und genau einer Abteilung ohne Kundenkontakt für die Verwaltung besteht. Dieser Abteilung ist eine weitere für die Bearbeitung der Rechnungen untergeordnet, die wiederum mögliche Schnittstellen und Arten der Bezahlung definiert. Die Verwaltung kann optional noch eine Versandabteilung enthalten. Das

Modell enthält Featuregruppen, Kardinalitäten und eine Referenz zu einem weiteren Featuremodell (Katalog).

Tabelle 2.1 fasst mögliche Diagrammelemente und deren Bedeutung zusammen.

Symbol	Erklärung
	Einzelnes Merkmal mit Kardinalität [1..1]; *erforderliches* Merkmal
	Einzelnes Merkmal mit Kardinalität [0..1]; *optionales* Merkmal
	Einzelnes Merkmal mit Kardinalität [0..m]; *optionales klonbares* Merkmal
	Einzelnes Merkmal mit Kardinalität [n..m]; *erforderliches klonbares* Merkmal
	Gruppiertes Merkmal mit Kardinalität [0..1]
	Gruppiertes Merkmal mit Kardinalität [1..1]
	Merkmal F mit Attribut vom Typ T und Wert *value*
	Referenz auf Featuremodell
	Merkmalsgruppe mit Kardinalität (1-1); *xor*-Gruppe
	Merkmalsgruppe mit Kardinalität (1-k), mit der Gruppengröße k; *or*-Gruppe
	Merkmalsgruppe mit Kardinalität (i-j)

Tabelle 2.1
Diagrammelemente Featuremodell (Czarnecki u. a., 2005, S. 3)

Ohne auf weitere Details einzugehen, sei nur noch auf die Angabe von Nebenbedingungen hingewiesen. Mit Nebenbedingungen können Featureimplikationen ausgedrückt werden, so dass z. B. im Kontext x bei der Auswahl des Features y auch das Merkmal z ausgewählt werden muss. Abschnitt 3.2.6 greift die konkrete Realisierung im Rahmen dieser Arbeit nochmals auf.

```
context X:
Y implies Z
```

Im Weiteren werden nicht sämtliche Möglichkeiten dieser Modelle voll ausgeschöpft, dennoch bieten die kardinalitätsbasierenden Featuremodelle mit Nebenbedingungen eine gute Basis für zukünftige Erweiterungen. So können im

Rahmen des entwickelten Prototyps zunächst keine Nebenbedingungen über die Implikation oder Exklusion hinaus angegeben werden. Auch auf die Möglichkeit,Teile des Featuremodells zu klonen wurde zunächst verzichtet. Dem gegenüber steht die Verwendung der Merkmalsattribute und der Kardinalitäten, von denen die späteren Anwendungsbeispiele, insbesondere durch übersichtlichere Modelle, stark profitieren.

Dokumente und Dokumentfamilien

Im Rahmen dieser Arbeit bezieht sich der Begriff *Dokument*, soweit nichts anderes angegeben, auf Dateien im Open Document Format, wie im nächsten Abschnitt ausgeführt wird. Diese Einschränkung ist aber nicht generell zu treffen. Der im Weiteren vorgestellte Ansatz lässt sich auch auf Dateien in anderen Formaten, so z. B. Latex, Office Open XML, anderen XML"=Dialekten oder CSV[2], übertragen.

Dokument

Eine *Dokumentfamilie* oder Dokumentproduktlinie unterliegt denselben Anforderungen wie sie auch für Softwareproduktlinien in Abschnitt 2.1.1 vorgestellt wurden. Das bloße Kopieren oder Wiederverwenden von Elementen aus Bibliotheken reicht nicht aus, um eine Dokumentfamilie zu begründen. Aus einer Reihe vorhandener oder zu erzeugender Dokumente ist eine gemeinsame Plattform zu ermitteln, die als Basis für eine „strategische Wiederverwendung" dienen kann.

Dokumentfamilie

Generierung von Produkten bzw. Zieldokumenten

Für die Realisierung konkreter Produkte einer Softwareproduktlinie gibt es einige Möglichkeiten, die Variabilität der Featuremodelle mit dem Quellcode zu verknüpfen. In (Gacek und Anastasopoules, 2001) werden verschiedene Ansätze auf ihre Leistungsfähigkeit und Schwachstellen untersucht. Dabei können zwei grundlegende Arten von Verfahren unterschieden werden.

Additive Verfahren Typische Vertreter der *additiven Verfahren* sind die aspektorientierten Ansätze (Kiczales u. a., 1997). Dabei werden Komponenten zunächst unabhängig voneinander erstellt und Merkmalen des Featuremodells zugeordnet. Nach Auswahl der Merkmale zu einer konkreten Variante werden die Komponenten zu einem Gesamtsystem verwoben.

Additive Verfahren

Subtraktive Verfahren Dem gegenüber stehen *subtraktive Verfahren*, wie der templatebasierte Ansatz nach Czarnecki (Czarnecki, 2005). Hierbei existiert ein Modell, das sämtliche Elemente für alle zu erzeugenden Varianten enthält und als Vorlage dient. Dieses Modell enthält darüber hinaus Annotationen mit

Subtraktive Verfahren

[2]Comma-Separated Values

Existenzbedingungen für die einzelnen Elemente. Beim Erzeugen einer konkreten Instanz werden, wie in Abbildung 2.2 dargestellt, zunächst die Existenzbedingungen ausgewertet und daraus resultierend die nicht benötigten Elemente aus der Vorlage entfernt.

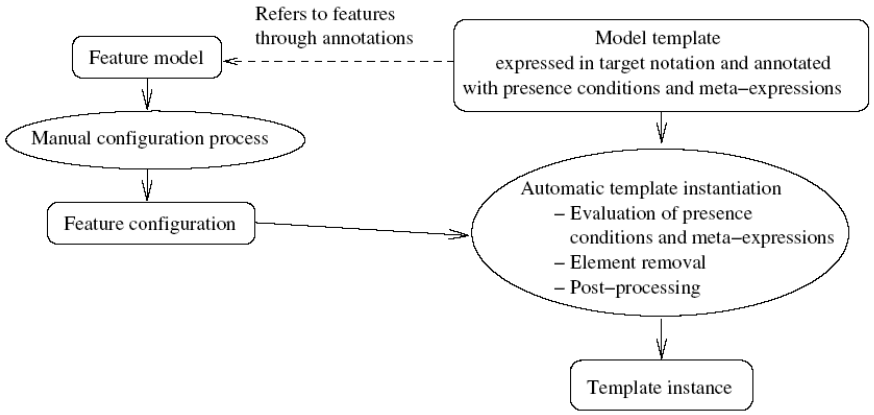


Abbildung 2.2
Schema templatebasierter Ansatz (Czarnecki, 2005)

Da ein Dokument ebenfalls als Modell aufgefasst werden kann und sich der templatebasierte Ansatz für Modelle nicht zuletzt durch (Kopcsek, 2007) in der Praxis bewährt hat, soll die Annahme der Übertragbarkeit dieses Ansatzes in dieser Arbeit untermauert werden.

2.1.3 Open Document Format

ODF

Das *Open Document Format*[3] ist ein offener Standard für diverse Bürodokumente. 2006 wurde der von OASIS und Sun entwickelte Standard als ISO/IEC 26300[4] zertifiziert.

Derzeit liegt das Open Document Format in der Version 1.1 vor.[5] Eine Version 1.2, die einige Mängel beheben und die Ausdrucksstärke und Kompatibilität erhöhen soll, befindet sich in Arbeit. Um die im weiteren Verlauf dieser Arbeit getroffenen Entscheidungen einordnen zu können, sei an dieser Stelle nochmals darauf hingewiesen, dass ausschließlich die Version 1.0 als ISO/IEC 26300 zertifiziert ist.

Das Open Document Format ist eine Erweiterung der Extensible Markup Language (XML) und greift zur Beschreibung der Dokumente bewährte Elemente der Auszeichnungssprache HTML[6] auf. Darüber hinaus können Formeln in MathML[7] verwendet werden. Ein vollständiges Dokument besteht aus mehreren

[3] vollständig: OASIS Open Document Format for Office Applications
[4] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485
[5] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office#odf11
[6] Hypertext Markup Language; z. B. Absatz `text:p`, Textbereich `text:span`
[7] Mathematical Markup Language

Dateien, u. a. einer `content.xml` für den eigentlichen Inhalt und einer `meta.xml` für Metadaten, die das Dokument genauer beschreiben, welche anschließend als Archiv im ZIP-Format[8] zu einer Datei mit der Endung `*.od(s|t|p|g)` verpackt werden.

Listing 2.1 zeigt ein minimales Textdokument im Open Document Format und Abbildung 2.3 dasselbe Dokument innerhalb von OpenOffice.org Writer. Mit dem Hinweis, dass `text:p` einen Absatz und `text:span` einen Textbereich kennzeichnet, sollte das beschriebene Dokument sofort erschlossen sein.

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-content ...>
  <office:scripts/>
  <office:font-face-decls>
    <style:font-face style:name="Arial"
                     svg:font-family="Arial"
                     style:font-family-generic="swiss"
                     style:font-pitch="variable"/>
  </office:font-face-decls>
  <office:automatic-styles/>
  <office:body>
    <office:text>
      ...
      <text:p text:style-name="Standard">
        <text:span>Ein Text</text:span>
      </text:p>
      <text:p text:style-name="Standard">
        <text:span>Noch ein Text in
            neuem Absatz</text:span>
      </text:p>
    </office:text>
  </office:body>
</office:document-content>
```

Listing 2.1
Minimales Textdokument in ODF

2.2 Werkzeuge

2.2.1 Eclipse

Die quelloffene integrierte Entwicklungsumgebung *Eclipse* ist zunächst für die Programmiersprache Java entwickelt worden. Durch die Möglichkeit, das Framework um weitere *Plugins* zu erweitern, gilt diese Einschränkung schon lange nicht mehr, und Eclipse findet heute nicht nur Anwendung als IDE für weite-

Eclipse

Plugin

[8] http://www.pkware.com/documents/casestudies/APPNOTE.TXT

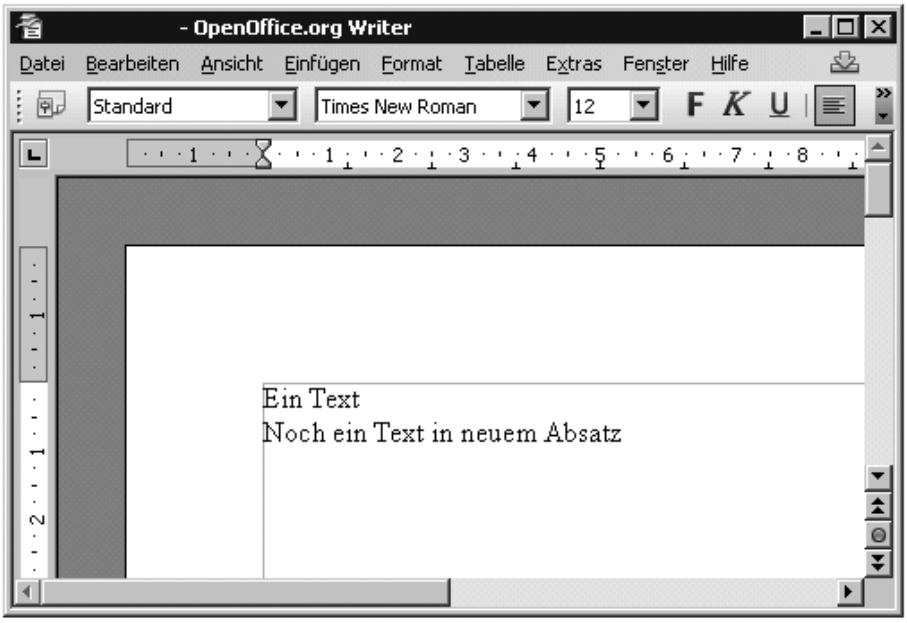


Abbildung 2.3
Minimales Textdokument in OpenOffice.org Writer

re Programmiersprachen, sondern wird auch in vielen Bereichen eingesetzt, die erhöhte Anforderungen an eine Benutzeroberfläche stellen.

Die aktuelle Version 3.5.1 Galileo kann unter [9] in verschiedene Konfigurationen bezogen werden.

Für einen Einstieg in die Pluginentwicklung sei auf [10] verwiesen. Dort befinden sich sowohl allgemeine Schritt-für-Schritt-Anleitungen für den Start als auch detailliertere Tutorials zum Beispiel über *JFace*, dem erweiterten GUI-Toolkit.

JFace

EMF

EMF

Das *Eclipse Modeling Framework* erweitert Eclipse um Elemente der modellgetriebenen Entwicklung. So besteht die Möglichkeit aus einem Modell Javacode zu generieren. Darauf aufbauend können weiterhin automatisch JFace-Steuerelemente erzeugt werden, die es zum Beispiel erlauben eine Modellinstanz in einem Editor zu bearbeiten. Auch die (De-)Serialisierung wird automatisch generiert.

Als Ausgangsbasis dient dabei ein Ecore-Modell, welches direkt erstellt oder u. a. aus vorhandenem annotierten Javacode, einem UML-Klassendiagramm oder einem XML-Schema (vgl. Abschnitt 2.3) erzeugt werden kann. Weiterhin besteht die Möglichkeit, Modelle auch erst zur Laufzeit zu generieren.

[9] http://www.eclipse.org/
[10] http://www.eclipse.org/articles/

2.2.2 FeatureMapper

FeatureMapper

Der *FeatureMapper* ist ein Eclipse-Plugin, um Features im Sinne der Featuremodelle (Abschnitt 2.1.2) auf Ecore-basierte Modelle abzubilden. Neben einer umfassenden grafischen Aufbereitung liegt der Schwerpunkt der Entwicklung auf einer hohen Erweiterbarkeit sowohl bei den Modellen als auch bei den Möglichkeiten der Transformation der Modelle. Die Wahl von Ecore als Basis bietet den großen Vorteil, ohne weiteren Aufwand eine Vielzahl von domainspezifischen Sprachen zu unterstützen, und auch via Eclipse mit geringem Aufwand neue geeignete Sprachen zu erstellen. Dabei liegt nach (Heidenreich, 2009) ein besonderes Augenmerk auf der Wohlgeformtheit und Validität der Quell- und Zielmodelle.

Abbildung 2.4 zeigt dabei das generelle Vorgehen zum Erzeugen eines Produkts einer SPL mit Hilfe des FeatureMappers. In einem ersten Schritt (1) werden den Elementen der Quellmodelle Merkmale des Featuremodells zugeordnet. Ein Quellmodell repräsentiert dabei die Gesamtheit aller gewünschten Zielmodelle, so dass bei der Erzeugung einer konkreten Instanz nur Elemente entsprechend der Zuordnung entfernt werden müssen. Darüber hinaus besteht die Möglichkeit, Elemente bzgl. ihrer Attribute zu verändern. Diesem Ableitungsschritt (3) geht noch die Spezifikation der Variante (2) voraus, in der, basierend auf dem Featuremodell, eine Menge an Merkmalen ausgewählt wird.

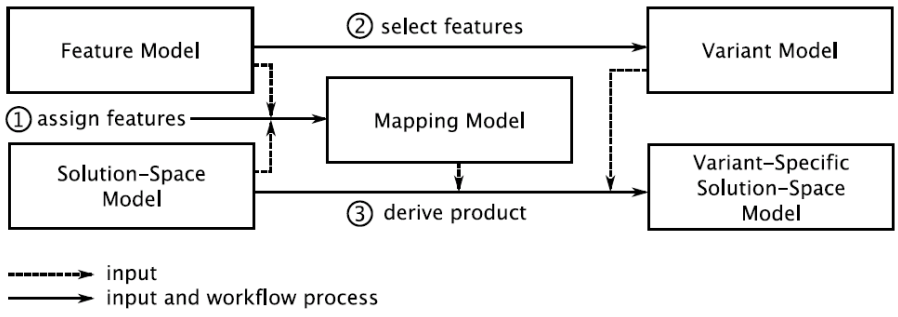


Abbildung 2.4
Arbeitsablauf zur Definition einer SPL und Ableitung eines konkreten Produkts im FeatureMapper. (Heidenreich, 2009, Abbildung 1)

Für das Erstellen der Merkmalszuordnung stellt der FeatureMapper eine Reihe von Werkzeugen zur Verfügung. Dem Benutzer steht es frei, die Abbildung manuell auf einem vorhanden Modell vorzunehmen oder automatisch beim Erstellen des Modells einzelne Merkmale aufzeichnen zu lassen. Dabei können sowohl EMF- als auch GMF-Editoren zur Auswahl der einzelnen Elemente verwendet werden.

Für umfangreichere Abbildungen können mehrere Merkmale mit Hilfe logischer Operationen zu komplexen Ausdrücken verknüpft werden. Um dennoch ohne Schwierigkeiten den Überblick auch in größeren Modellen zu behalten, existieren Filter, die innerhalb der Editoren angewendet werden können, um

entweder eine Variante oder ein Merkmal zu visualisieren.

Die aktuelle Version sowie detailliertere Dokumente sind unter [11] zu finden.

2.2.3 OpenOffice.org

Da das Open Document Format nur ein Standard ist, gilt es zunächst, eine Anwendung zu benennen, die diesen Standard unterstützt. Die Entscheidung fällt hier aus zwei wesentlichen Gründen auf OpenOffice.org in der aktuellen Version. Erstens ist OpenOffice.org neben den Produkten von Microsoft eine der am häufigsten verwendeten Büroanwendungssammlungen und genießt über alle Betriebssysteme hinweg eine gute Verbreitung (OpenOffice.org, 2009a). Weiterhin wurde der ODF-Standard in seiner ursprünglichen Form, ebenso wie OpenOffice.org, von Sun entwickelt, somit ist OpenOffice.org parallel zum Standard entstanden.

OpenOffice.org

OpenOffice.org ist eine freie quelloffene Sammlung von Büroanwendungen (Tabellenkalkulation, Textverarbeitung, Präsentation usw.). Seit der Version 2.x wird das Open Document Format unterstützt. OpenOffice.org kann in der aktuellen Version 3.2.x unter [12] bezogen werden und steht in unzähligen Sprachen und für diverse Betriebssysteme zur Verfügung. Auf Details zum Funktionsumfang soll an dieser Stelle verzichtet werden.

SDK, API und UNO

Software Development Kit

OpenOffice.org stellt ein *Software Development Kit*[13] zur Verfügung, mit welchem aus diversen Programmiersprachen auf bestehende Komponenten zugegriffen oder neue Komponenten erstellt werden können.

UNO

Den Kern bilden dabei *Universal Network Object*s, welche sprach-, technologie- und plattformunabhängig sind und verteilt über beliebige Netzwerke verarbeitet werden können. Die UNO-Komponenten können so OpenOffice.org erweitern oder als eigenständige Anwendung mit Zugriff auf OpenOffice.org agieren.

Kontext
ServiceManager
Service

Jede Komponente besitzt dabei einen *Kontext* mit einem *ServiceManager*, der als Factory für *Service*s zu Verfügung steht, die wiederum diverse Operationen ausführen können. So gibt es für den Zugriff auf das aktuelle und alle weiteren geöffneten Dokumente z. B. den Service COM.SUN.STAR.FRAME.DESKTOP, der auch eine Funktion zum Öffnen von neuen Dokumenten anbietet.

Interface

Für die Integration der Objekte in Java ist weiterhin ein Verständnis des *Interface*-Konzepts innerhalb von OpenOffice.org nötig. Abbildung 2.5 zeigt die

[11] http://www.featuremapper.org/
[12] http://www.openoffice.org
[13] http://download.openoffice.org/3.0.0/sdk.html

Interfaces eines Dokuments und des abgeleiteten Textdokuments. Zunächst ist zu erkennen, dass Wert darauf gelegt wird, die Interfaces möglichst schlank und präzise zu halten, darüber hinaus führt dies aber innerhalb des SDKs in Java zu vielen Methoden mit dem Rückgabetyp `Object` bzw. in OpenOffice.org `Any`.

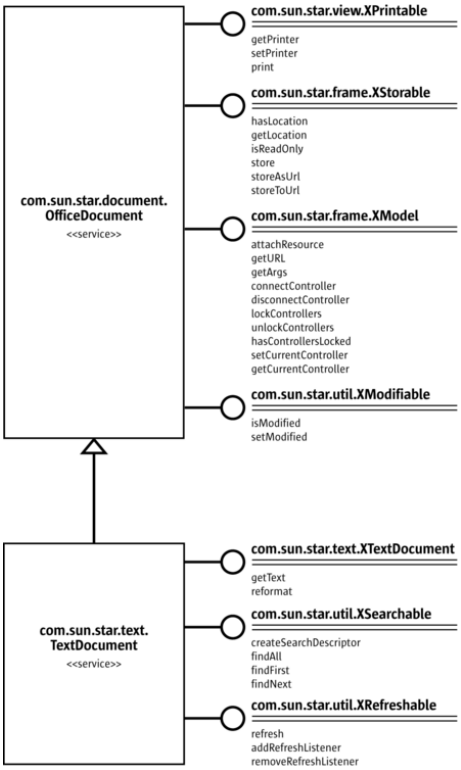


Abbildung 2.5
Interfaces von Dokumenten in OpenOffice.org. (OpenOffice.org, 2009b)

Um diesem Problem zu begegnen und in Java die Typen korrekt umzuwandeln, bietet die UNO-Laufzeitumgebung die Möglichkeit, Objekte nach konkreten Schnittstellen zu fragen. Listing 2.2 zeigt dies exemplarisch mit dem vom Desktop-Service implementierten Interface XComponentLoader. Implementiert ein Objekt das Interface nicht, wird `null` zurückgegeben.

```
Object desktop = xRemoteServiceManager.
                   createInstanceWithContext(
                     "com.sun.star.frame.Desktop",
                     xRemoteContext
                   );

XComponentLoader xComponentLoader = (XComponentLoader)
                 UnoRuntime.queryInterface(
                   XComponentLoader.class, desktop);
```

Listing 2.2
Anfrage eines Interfaces

Ein umfangreicheres Beispiel für Operationen auf einem Tabellendokument kann unter [14] betrachtet werden. Für das weitere Verständnis dieser Arbeit ist aber ein tieferer Einblick unnötig, da die UNO-Komponenten über die Analyse hinaus nur in der OpenOffice.org-Ansicht (vgl. Abschnitt 3.3.3) Anwendung gefunden haben. Für einen umfassenden Einstieg sei auf [15] verwiesen.

2.2.4 Nice Office Access for Eclipse

Auch der recht kurze Einstieg in die Möglichkeiten des SDKs sollte dessen Komplexität aufgezeigt haben. Um die Arbeit mit OpenOffice.org näher an die gewöhnliche Arbeitsweise in der Entwicklung von Software mit Java zu bringen,
NOA4e stellt die Firma ubios.ORS[16] ein Eclipse-Plugin mit dem Namen *Nice Office Access for Eclipse*[17] zur Verfügung, das den Zugriff auf die OpenOffice.org-API kapselt und eine einfachere Integration der Komponenten in Eclipse erlaubt.

Das Framework ist leider noch nicht vollständig, und wird dies vermutlich auch nie werden, da es derzeit nur als nicht kommerzielles Projekt von einem Entwickler betreut wird. Es sind viele häufig benötigte Operationen, insbesondere die Einbettung von OpenOffice.org in Eclipse enthalten. Dennoch benötigt man fundierte Kenntnisse über das SDK, um mögliche Lücken zur eigenen Anwendung zu schließen. Auch dies soll hier nicht weiter vertieft werden. Eine Realisierung kann in der Klasse OpenOfficeView[18] nachverfolgt werden.

[14] http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/FirstSteps/Example:_Working_with_a_Spreadsheet_Document
[15] http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/OpenOffice.org_Developers_Guide
[16] http://ubion.ion.ag
[17] http://ubion.ion.ag/loesungen
[18] Javadoc: de.mheinzerling.dfm.ooview.views.OpenOfficeView

2.3 Lösungsansätze

Basierend auf den im vorhergehenden Abschnitt vorgestellten Technologien stehen nun verschiedene Lösungsansätze zur Verfügung, die im Folgenden näher betrachtet werden sollen. Dabei ist zunächst eine Entscheidung zu treffen, ob ein in OpenOffice.org integrierter Ansatz oder ein eigenständiges Werkzeug der bessere Ausgangspunkt ist. Darüber hinaus müssen die Vor- und Nachteile einer Anwendung des Mappings entweder auf XML in Form des Open Document Formats oder direkt auf UNOs innerhalb der Open-Office-Komponenten untersucht werden. Abschließend ist die Eignung des vorhandenen FeatureMappers der Neuentwicklung eines eigenständigen Werkzeugs gegenüber zu stellen.

Bereits im Vorfeld sei angemerkt, dass theoretisch jede Kombination dieser drei Aspekte für ein Erreichen der Zielstellung denkbar wäre. Lediglich weitere Kriterien führen zu einer unterschiedlichen Bewertung der Lösungen.

Erweiterbarkeit Möglichkeiten und Aufwand für zukünftige Erweiterungen auf andere (Dokument-)Formate bzw. Erhöhung des Funktionsumfangs

Wartbarkeit Einarbeitungszeit für zukünftige Entwickler

Praxistauglichkeit Aufwand für das Erstellen einer Merkmalszuordnung

Benutzerfreundlichkeit Eignung für den normalen Benutzer ohne ein vertieftes Verständnis des Open Document Formats

Risiko Wahrscheinlichkeit und Kosten möglicher Komplikationen

2.3.1 Werkzeugintegration

Für die Verbindung der Merkmalszuordnung mit OpenOffice.org existieren zwei mögliche Lösungen. Zum einen kann die zusätzlich benötigte Funktionalität direkt in OpenOffice.org integriert werden. Dem gegenüber steht die Entwicklung eines eigenständigen Werkzeugs, welches wiederum OpenOffice.org einbettet. Tabelle 2.2 stellt die Alternativen in einer Übersicht gegenüber. Im Weiteren werden die einzelnen Punkte näher erläutert.

Eine prototypische Implementierung zur direkten Integration eines Werkzeugs in OpenOffice.org hat aufgezeigt, dass nicht alle Elemente des Standards eine direkte visuelle Repräsentation haben (z. B. ein häufiges Ziel von Modifikationen, die Textbereiche innerhalb eines Absatzes). Somit wäre hier eine Erweiterung der Oberfläche nötig geworden, die sich sehr wahrscheinlich negativ auf den Überblick und den Arbeitsablauf ausgewirkt hätte. Hinzu kommt weiterhin, dass viele Eigenschaften von Elementen in Dokumenten nur über mehrere

	Integration in OOo	Einbettung von OOo
Erweiterbarkeit	− beschränkt auf ODF	+ flexibel bei entsprechender Architektur
Wartbarkeit	− hoher Einarbeitsaufwand in OOo“=SDK und -API	
Praxistauglichkeit	− Nicht alle Elemente sind sichtbar − Überladen des GUIs bei Visualisierung des Mappings	+ Optimierung auf Szenario möglich − Synchronisierung zwischen Einbettung und Abbildungseditor nötig
Benutzerfreundlichkeit	+ gewohnte Umgebung − „viele Dialoge“ − Überladen der Oberfläche	− Mentale Karte zw. Dokument und baumartiger Inhaltsübersicht nötig
Risiko	mittel/hoch	mittel/gering

Tabelle 2.2 Gegenüberstellung Integration in vs. Einbettung von OpenOffice.org

Dialoge zu erreichen sind und Modifikationen daran einen erhöhten Aufwand für den Benutzer verursachen würden.

Auch eine visuelle Zuordnung von Merkmalen zu Elementen ist innerhalb eines WYSIWYG-Editors[19] dieses Umfangs nur bedingt ansprechend darstellbar. Ein Einfärben, Unterlegen oder Umranden innerhalb des Dokuments wirkt in den meisten Fällen störend, da Konflikte mit der Formatierung des eigentlichen Inhalts auftreten können. Eine zusätzliche Ebene im Sinne eines Overlays wird von OpenOffice.org leider nicht unterstützt, so dass solche Annotationen z. B. direkt in den Objekteigenschaften realisiert werden müssten, mit der daraus resultierenden Notwendigkeit, die eigentlichen Eigenschaften in irgendeiner Art zu sichern.

Ergänzend wäre auch die Frage zu klären, wie ein Mapping mit mehreren Dokumenten zu behandeln ist, da OpenOffice.org einzelne Dokumente jeweils in einer eigenen Instanz öffnet. Die Möglichkeit, eine Komponente zu entwickeln, die als Singleton über den Instanzen existiert, gibt es nicht, so dass jedes Dokument erneut die Merkmalszuordnung öffnen müsste.

Ein weiterer nicht zu unterschätzender Faktor ist die Komplexität des SDKs und der APIs von OpenOffice.org. Eine Entscheidung für eine vollständig integrierte Lösung stellt eine unnötige Hürde für zukünftige Entwickler dar. Trotz umfangreicher Dokumentation und einer in Allgemeinen Fragen stets kompetenten Entwicklergemeinde stößt man schnell an die Grenzen, wenn detailliertere Fragen zu beantworten sind.

Der Integration gegenüber steht die Einbettung einer OpenOffice.org-Ansicht in ein Werkzeug, welches die Merkmalszuordnung übernimmt und das Mapping

[19] What You See Is What You Get; Editor der bereits das Ergebnis anzeigt.

unabhängig von OpenOffice.org visualisiert. Hier würde sich eine baumartige Inhaltsübersicht anbieten, die sämtliche Objekte und Eigenschaften des Dokuments hierarchisch darstellt. Dies wiederum muss dem Benutzer vermittelt werden, so dass eine mentale Verknüpfung von Dokumentelementen mit Knoten im Baum möglich ist.

Bei einer Trennung der eigentlichen Darstellung und der Merkmalszuordnung ist darüber hinaus im Besonderen auf die Synchronisierung der Repräsentationen zu achten.

2.3.2 Ziel der Merkmalsabbildung

Mit der Verwendung von OpenOffice.org als präferierter Anwendung für die Untersuchung wird sofort auch die Frage aufgeworfen, ob eine Umsetzung der Generierung von Dokumentfamilien mit Featuremodellen mit dem UNO-Komponentenmodell und der OpenOffice-API möglich ist. Derzeit gibt es keinen Grund zu der Annahme, dass dies nicht möglich sei.

Dennoch würde eine Konzentration auf die OpenOffice.org-API zunächst zu einer sehr spezifischen Lösungen führen, die nicht direkt auf andere Formate übertragbar ist. Eine entsprechende Architektur könnte aber die Flexibilität für zukünftige Erweiterungen schaffen.

Auch hier muss der äußerst umfangreiche Einarbeitsaufwand für zukünftige Entwickler in das SDK der Realisierung der Merkmalszuordnung zu UNO negativ angerechnet werden.

Dem gegenüber steht die Umsetzung direkt auf dem XML-basierten Open Document Format, welches ebenfalls sehr komplex, aber als Spezifikation deutlich überschaubarer ist. Darüber hinaus kann diese Lösung einen höheren Umfang von Dokumenten in diversen XML-Dialekten abdecken, ohne dass explizit weitere Anpassungen nötig sind.

```
'Präsentationdokument
ThisComponent.Drawpages

'Textdokument
ThisComponent.Drawpage

'Tabellendokument
ThisComponent.Sheets.Drawpages
```

Listing 2.3
Zugriff auf Dokumentinhalte mit OOo-Basic

Ebenfalls nicht zu vernachlässigen ist der fehlende direkte Zusammenhang zwischen der internen Repräsentation von Dokumenten in OpenOffice.org und dem ODF. Dies erhöht u. a. den Aufwand, durch ein Dokument zu navigieren,

im Vergleich zur Iteration über einen XML-Baum oder der Verwendung von XPath. Weiterhin ist bei der Arbeit auf den UNOs jeder Dokumenttyp individuell zu behandeln. Listing 2.3 zeigt exemplarisch den Zugriff auf den Inhalt von Dokumenten in Abhängigkeit vom Dokumenttyp. Eine „Drawpage“ ist dabei in einer Präsentation eine einzelne Folie, im Textdokument hingegen das gesamte Dokument. Eine solche Unterscheidung gibt es im ODF nicht. Dies bedeutet aber im Umkehrschluss, dass eine Synchronisierung einer XML-basierten Repräsentation mit einer OpenOffice.org-Ansicht nicht trivial ist.

	UNO/SDK	ODF
Erweiterbarkeit	− zunächst auf OOo beschränkt + bei entsprechender Architektur auch auf andere Dokumente erweiterbar	+ alle XML-Dialekte und Archive + bei entsprechender Architektur auch auf andere Dokumente erweiterbar
Wartbarkeit	− hoher Einarbeitsaufwand in OOo-SDK und -API − Diskrepanz zw. UNO und ODF	+ „nur“ Kenntnisse des ODF nötig
Praxistauglichkeit	− Schwierige Identifikation von Elementen	− Schwierige Synchronisation mit OOo
Benutzerfreundlichkeit	im besten Fall merkt der Benutzer keinen Unterschied	
Risiko	hoch/hoch	mittel/mittel

Tabelle 2.3 Gegenüberstellung UNO/SDK vs. ODF

2.3.3 Erweiterung des FeatureMappers

Bei der Vorstellung des FeatureMappers (Abschnitt 2.2.2) wurde bereits auf die Ecore-Basis, aber auch die Schwerpunktsetzung auf Validität der Quell- und Zielmodelle hingewiesen. Für die direkte Verwendung des FeatureMappers ist also das ODF in ein Ecore-Modell zu überführen.[20]

Bevor aber ein Ecore-Modell erstellt werden kann, muss zunächst festgestellt werden, dass nur die wenigsten OpenOffice.org-Dokumente auch vollständig dem ODF-Standard entsprechen (Brown, 2008). Auch eine nach Weir (2008) angepasste Validierung gegen das Relax-NG-Schema[21] führte bei Dokumenten, die z. B. aus Microsoft-Formaten innerhalb von OpenOffice.org konvertiert wurden oder in die Inhalte aus Webseiten eingefügt wurden, zu Verletzungen des Schemas. Es steht zu befürchten, dass dieser Konflikt zwischen Standard und Realität durch die ausstehende Erweiterung auf die Version 1.2 des ODF-Schemas nicht gelöst werden kann oder gar durch neue Interpretationsversuche diverser

[20] Ein Ansatz mit UNOs ist derzeit im FeatureMapper nicht möglich.
[21] http://relaxng.org; alternative Beschreibung für valide XML-Dokumente

Hersteller verstärkt wird.

Um die Vorteile eines bereits vorhandenen Werkzeugs wie dem FeatureMapper nicht voreilig aus der Betrachtung zu entfernen, wurde dieser Pfad zunächst unter der Annahme, dass auch gültige Dokumente existieren, weiter verfolgt. Das EMF unterstützt aber das Erstellen von Ecore-Modellen derzeit nur aus XSD[22], somit muss das Relax-NG-Schema zunächst in eine XSD konvertiert werden. Trotz desselben Ursprungs dieser beiden Definitionssprachen ist dies nur eingeschränkt möglich. Im konkreten Fall bietet RNG die Möglichkeit, zu spezifizieren, dass ein XML-Element sowohl als Attribut als auch als Kindelement auftreten kann. Dies unterstützt XSD nicht und muss ebenso wie optionale Attributgruppen approximiert werden. In dem letzten noch ausstehenden Schritt der Generierung eines Ecore-Modells treten dann weitere Fehler bei der Abbildung von XSD-Attributen der Form Union Simple Type, also Attributen, die mehrere durch Leerzeichen getrennte Werte enthalten, auf.

Es kann also kein statisches Ecore-Modell erstellt werden, welches zusätzlich auch noch die Invalidität in real existierenden Dokumenten abdeckt. Aufgrund des Umfangs der ODF-Spezifikation ist der Zeitaufwand für eine manuelle Anpassung und der anschließende Nachweis der Korrektheit nicht abzuschätzen. EMF bietet darüber hinaus auch noch die Möglichkeit, Modelle erst zur Laufzeit zu erzeugen. Dies geschieht z. B. mit der GenericXMLResourceFactory (Steinberg u. a., 2009, s. 490). Ohne weitere Anpassung führt dies jedoch zu Problemen bei der Performance. Für eine konkrete Messung des Speicher- und Zeitbedarfs wurde ein Szenario gewählt, welches relativ häufig in der Benutzeroberfläche auftreten kann: Ein Dokument soll zunächst geladen werden, anschließend wird ohne weitere Operationen einmal über das Dokument iteriert, um es dann wieder zu speichern. Dieser Ablauf tritt z. B. immer dann auf, wenn die OpenOffice.org-Ansicht mit der Baumansicht synchronisiert wird, da keine Möglichkeit besteht, Änderungen in den Repräsentationen nur partiell zu propagieren. Schon bei einer relativ kleinen XML-Datei von 1,5 MB führt die Verwendung von Ecore zu einem mehr als doppelt so hohen Speicherverbrauch (JDOM: 6,88 Mb; Ecore: 14,46 Mb $\rightarrow$ 210,01 %) und mehr als dem vierfachen der Zeit (JDOM: 1443 ms; Ecore: 5827 ms $\rightarrow$ 403,81 %) im Vergleich zur direkten Repräsentation des XML-DOM[23]. Insbesondere die lange Laufzeit ist einem Endbenutzer nicht zu vermitteln und daher nicht akzeptabel.[24]

Nachdem ein Ecore-basierter Ansatz nicht praktikabel zu realisieren ist, gilt es, die Erweiterung des FeatureMappers für nicht Ecore-basierte Modelle ins

[22] http://www.w3.org/XML/Schema; XML Schema Definition
[23] unter Verwendung von JDOM; http://www.jdom.org
[24] In einer unveröffentlichten prototypischen Implementierung (liegt bei) im Vorfeld dieser Arbeit durch Sven Karol konnten ebenfalls Performanceprobleme in der späteren Darstellung der Merkmalszuordnung festgestellt werden, die jedoch hier nicht weiter verfolgt werden sollen.

Zentrum der Betrachtung zu rücken. Hierzu ist zunächst ein Zitat aus der dem FeatureMapper zugrunde liegenden Arbeit zu betrachten. (Kopcsek, 2007, S. 65, letzter Absatz)

> **Erweiterung auf andere Modellarten** Der vorgestellt Ansatz beschränkt sich in jeder Hinsicht auf Ecore-basierte Modelle. Kernmodelle sowie Fragmente, wie Aspekte, müssen Ecore-basierte Modelle sein, damit sie genutzt werden können. Durch die Bereitstellung bestimmter Schnittstellen und der Verallgemeinerung von grundlegenden Konzepten, wie der Adressierung von Elementen, könnte diese Beschränkung aufgelöst werden.

Zusammengefasst ist also zunächst keine Erweiterung vorgesehen und eine umfassendere Umstrukturierung nötig. Eine weitere Untersuchung der Abhängigkeiten der Plugins und der Paketstruktur (Abbildung 2.6) verdeutlicht nochmals detaillierter den nötigen Umfang der Umstrukturierung und die Verzahnung der einzelnen Plugins.[25] Die grau unterlegten Plugins weisen wiederum auf Zyklen in der internen Paketstruktur hin.

Dieser Vielzahl an Änderungen steht eine Testabdeckung, über alle Plugins hinweg, von lediglich 15 %[26] gegenüber. Für die grafische Benutzeroberfläche existieren sogar überhaupt keine Testfälle.

Selbst bei einer optimistischen Schätzung ist der Aufbau einer Testbasis eines „unbekannten“ Systems, das Beheben dabei auftretender Fehler und die für den weiteren Verlauf der Arbeit nötige Abstraktion der Schnittstellen kaum in der sechsmonatigen Bearbeitungszeit einer Diplomarbeit zu realisieren. Darüber hinaus enthält diese Abschätzung noch keine Anpassung hinsichtlich der Benutzerfreundlichkeit und Praxistauglichkeit für das konkret zu untersuchende Szenario, die Entwicklung eines Varianteneditors, einer Schnittstelle für Reparaturmaßnahmen an der Merkmalszuordnung oder die Einführung einer strikteren Trennung von Umsetzung der Abbildung und darauf aufbauenden Nachbearbeitungsschritten.

Eine saubere Realisierung unter Verwendung des FeatureMappers ist in dem gegeben Zeitrahmen somit nicht möglich. Darüber hinaus macht die komplexe Abhängigkeit zwischen den einzelnen Paketen eine sinnvolle Wiederverwendung einzelner Komponenten unmöglich. Dennoch können die theoretischen und konzeptionellen Überlegungen, die Grundlage der Entwicklung des FeatureMappers waren, auch als Basis für die Entwicklung eines Werkzeugs auf einer abstrakteren Ebene, unabhängig von Ecore und EMF, dienen und so auch zu einer späteren Integration des FeatureMappers führen.

[25] Die Plugins models.* wurden für die Übersicht zusammengefasst. In der Realität enden nicht immer alle Kanten bei allen Subpaketen.

[26] 7186 von 48984 Anweisungen

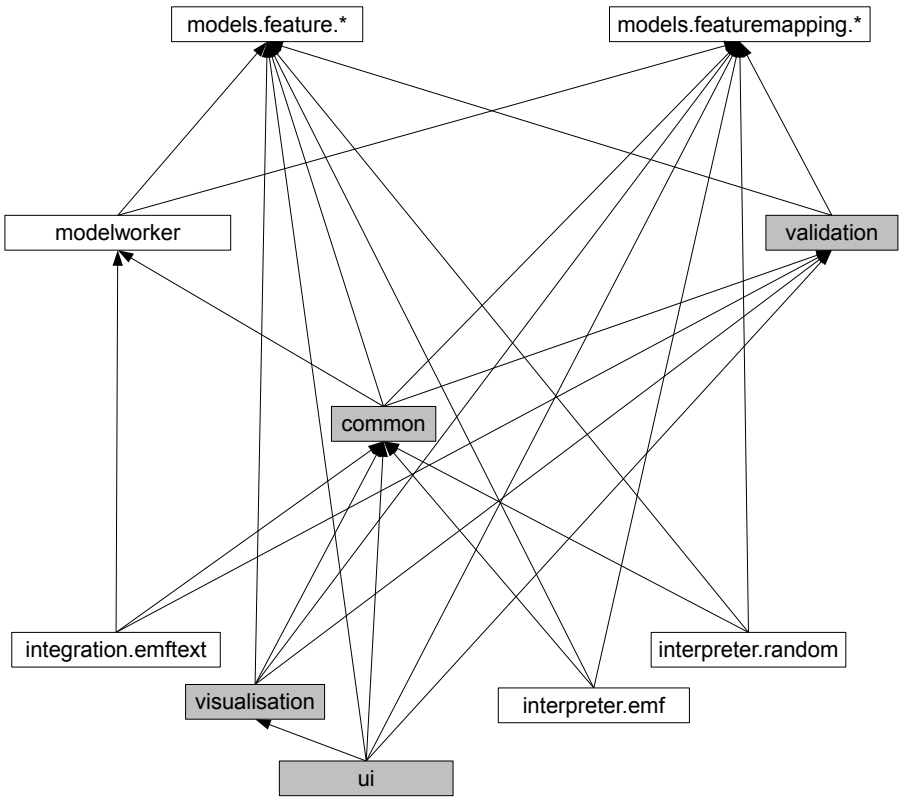


Abbildung 2.6
Abhängigkeiten der Kernplugins des FeatureMappers

Es sei nochmals ausdrücklich darauf hingewiesen, dass diese Analyse lediglich die Erweiterbarkeit des FeatureMappers im Rahmen dieser Arbeit betrachtet und sonst keine Aussagen über die Qualität des FeatureMappers getroffen werden. Es ist sogar so, dass die Entscheidung für ein eigenständiges Werkzeug zunächst dazu führt, auf die umfassenden Vorteile des FeatureMappers bzgl. der Validierung verzichten zu müssen. (Tabelle 2.4)

2.4 Schlussfolgerung

Aus den vorangegangenen Überlegungen resultiert zunächst die Erkenntnis, dass bereits unzählige Dokumente existieren, die ungültig gegen den ODFStandard sind. Die Entwicklung eines Programms darf also folglich nicht auf der Annahme basieren, gültige Dokumente zu erhalten.

Ebenso ist von einer intensiven Verwendung des OpenOffice.org-SDKs abzusehen, da ein umfassender Einstieg für zukünftige Entwickler unnötig erschwert wird, aber die Verwendung keine signifikanten Vorteile bringt. Dies gilt sowohl

	FeatureMapper	Eigenständiges Werkzeug
Erweiterbarkeit	− z. Z. keine Erweiterung außerhalb des EMF vorgesehen	+ flexibel bei entsprechender Architektur
Wartbarkeit	− unzureichende Testbasis − Architektur	− komplett neues Werkzeug + Testbasis
Praxistauglichkeit	+ Schwerpunkt auf Validität von Quell-/Zielmodellen	+ auch auf XML ohne Schema anwendbar + auf Szenario anpassbar (Varianteneditor, Reparatur, Nachbearbeitung, optimierte Merkmalszuordnung, Kommandozeilenversion usw.)
Benutzerfreundlichkeit	− Performance + Validierung + Realisierung + Rekorder	+ Optimierung für Endbenutzer möglich (z. B. Internationalisierung)
Risiko	hoch/hoch	mittel/gering

Tabelle 2.4
FeatureMapper vs. eigenständiges Werkzeug

beim Mapping als auch bei der Integration.

Der letzte Abschnitt hat ebenfalls aufgezeigt, dass eine Umsetzung der Zielstellung mit dem FeatureMapper in der gegebenen Zeit nicht möglich ist. Dennoch können das Metamodell der Featuremodelle und die theoretischen Grundlagen des FeatureMappers als Ausgangspunkt für die Entwicklung eines *DocumentFeatureMappers*[27] dienen. Weitere Anforderungen, der Integrationsansatz zu OpenOffice.org und die Herleitung einer für den Benutzer überschaubaren Darstellung der Featurezuordnung werden in Kapitel 3 vertieft. Dort sind ebenfalls sämtlich Hinweise für Erweiterungen und auch für die spätere Integration des FeatureMappers zu finden.

[27] Dieser Name ist mit der möglichen Integration des FeatureMappers nochmals zu überdenken, da die Funktion nicht auf Dokumente beschränkt ist.

Kapitel 3

DocumentFeatureMapper

Die Schlussfolgerungen aus Abschnitt 2.4 implizieren die Notwendigkeit eines Werkzeuges für die Abbildung von Featuremodellen auf XML-basierte Dokumente. Die Entwicklung, der Aufbau und die Funktionsweise dieses Werkzeugs sollen im folgenden Kapitel näher betrachtet werden. Hierzu werden die einzelnen Komponenten mit den ihnen zu Grunde liegenden Anforderungen, Designentscheidungen und Zielsetzungen vorgestellt.

Weiterführende Details zur Implementierung sind Anhang B zu entnehmen.

3.1 Architektur

Ressource und URI

Ressource
URI

Im Verlauf dieser Arbeit werden häufig die Begriffe Ressource und URI verwendet, daher sollen diese zunächst genauer definiert werden. Eine *Ressource*[1] kann auf dieser Ebene alles sein, was durch ein *URI*[2] identifiziert werden kann. Ein URI besteht dabei nach (Berners-Lee, 2005) aus den fünf Teilen: Schema, Autorität, Pfad, Query und Fragment. Ein mögliches Beispiel, wie es im Folgenden häufig Anwendung finden wird, ist in Listing 3.1 zu sehen.[3]

Listing 3.1
URI-Beispiel

```
zip:file://.../projektplan.ods!/content.xml#
/office:document-content/office:body/office:spreadsheet
```

Alles rechts der Raute ist das Fragment, in diesem Fall ein XPath-Ausdruck. Da es sich bei dieser Datei um ein lokales ZIP-Archiv handelt, ist das Schema mit `zip:file` angegeben. Dies hat auch zur Folge, dass der Pfad zur Datei hier als Autorität betrachtet wird, und nur der Pfad innerhalb des Archivs zwischen Ausrufezeichen und Raute als Pfad im Sinne eines URIs gilt.

Architektur

Abbildung 3.1 zeigt das Zusammenspiel der im Rahmen dieser Arbeit entstandenen Werkzeuge. Die Pfeile verdeutlichen den Datenfluss zwischen den einzelnen Werkzeugen und Dateien. In den folgenden Abschnitten werden die einzelnen Elemente im Detail vorgestellt.

Abschnitt 3.2 greift die eigentliche Kernaufgabe, das Zuordnen diverser Ressourcen zu den durch das Featuremodell gegebenen Merkmalen, auf. Die horizontale Unterteilung in der Abbildung soll verdeutlichen, dass die eigentliche Abbildung der Features sowohl unabhängig vom Open Document Format als auch von XML ist. Jede Ressource bzw. jedes Dokumentelement, das über ein URI eindeutig identifiziert werden kann, kann Grundlage der weiteren Bearbeitung sein.

Um dem Benutzer den Umgang mit den URIs zu erleichtern bzw. vollständig zu verbergen, wird ein Editor zur Verfügung gestellt. Dieser bildet die Schnittstelle, um Elemente aus diversen Dokumenten über einen eindeutige Identifikator anzusprechen. (Abschnitt 3.3)

[1] Nicht zu verwechseln mit EMF-Ressourcen. EMF-Ressourcen sind viel spezifischer und werden über einen URI mit anderem Fragmentinhalt adressiert.
[2] Uniform Resource Identifier
[3] Der URI kann jede beliebige Form annehmen, es muss lediglich ein Interpreter (vgl. Abschnitt 3.5) existieren, der diese verarbeiten kann.

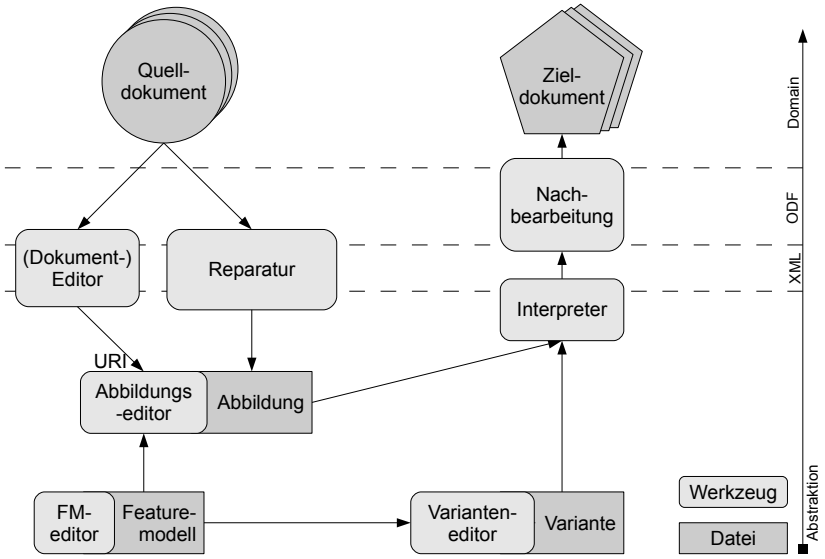


Abbildung 3.1
Architektur des
DocumentFeatureMappers

Einige URIs sind relativ instabil[4] und können beim nachträglichen Bearbeiten der Quelldokumente verändert oder gelöscht werden. Damit der Aufwand für den Benutzer reduziert wird, wurde eine Schnittstelle für Werkzeuge zur Reparatur der Abbildungsvorschriften entwickelt. (Abschnitt 3.4)

Nachdem aus dem zugrunde liegende Featuremodell eine Variante abgeleitet wurde, kann aus dieser in Verbindung mit der Abbildung eine Vorstufe der Zieldokumente erzeugt werden. Der Interpreter (Abschnitt 3.5) wertet dazu die Variante aus und entfernt sämtliche nicht sichtbaren URIs aus den zuvor erstellten Kopien der Quelldokumente.

Je nach Anwendungsszenario und den Fähigkeiten des Editors kann es nun in einem Nachbearbeitungsschritt (Abschnitt 3.6) nötig sein, die erzeugenden Dokumente wieder valide gegen ein bestimmtes Schema zu machen, unnötige Referenzen zu entfernen[5] oder sogar die Zieldateien zu einer einzigen zu verschmelzen.

3.2 Abbildungsansicht

Die Abbildungsansicht ist der zentrale Bestandteil des *DocumentFeatureMappers*. In diesem Abschnitt sollen die einzelnen Bereiche, einige zugrunde liegende Designentscheidungen und Hinweise zur Bedienung erörtert werden. Abbildung

[4] z. B. XPath \\page[3] beim Einfügen einer Seite vor dieser Seite
[5] z. B. nicht verwendete Bilder in ODF-Dokumenten

Abbildung 3.2
Quelldokumente

3.2 zeigt zunächst die vollständige, aber leere Ansicht. Über die Steuerelemente am oberen Rand kann ein neues Mapping unter Angabe eines Merkmalsmodells erstellt oder ein vorhandenes geladen werden.

3.2.1 Featuremodell

Die Featuremodelle des *DocumentFeatureMappers* basieren auf demselben Metamodell wie die des FeatureMappers, daher kann der vorhandene Editor weiter verwendet werden. Über *Datei→Neu→Andere...→Feature Model* und dem Folgen des Assistenten kann ein neues Featuremodell erstellt werden. Anschließend ist ein Bearbeiten durch Hinzufügen von Merkmalen, Gruppen und Nebenbedingungen möglich. Ergänzend zum FeatureMapper können auch Attribute definiert werden, die in einer Variante und im Mapping berücksichtigt werden.

3.2.2 Quelldokumente

Sämtliche Dokumente, die die Plattform für die Dokumentfamilie bilden, müssen innerhalb der Abbildungsansicht der Merkmalszuordnung bekannt gemacht werden. Dies hat den Vorteil gegenüber der Extraktion der Dokumente aus der Menge der verwendeten Identifikatoren, dass auch Dokumente in einer Variante generiert werden können, an denen selbst keine Veränderungen vorgenommen wurden. (Abbildung 3.3)

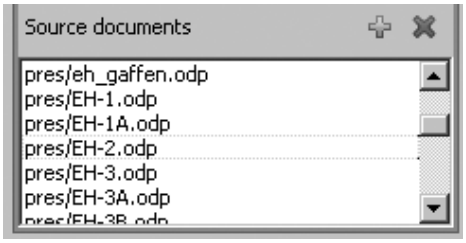


Abbildung 3.3
Quelldokumente

3.2.3 Zuordnung von Merkmalen zu Dokumentelementen

Die Kernaufgabe der Abbildungsansicht ist die Abbildung von Merkmalen auf Elemente der Quelldokumente. Dabei finden nicht bloß einzelne Merkmale Verwendung, sondern es ist auch eine Kombination von Merkmalen zu komplexen Termen möglich. Ein *Term* ist dabei eine logische Verknüpfung mehrerer Merkmale mit den Operatoren $\wedge$, $\vee$ und $\neg$.

Term

Abbildung 3.4 zeigt ein Szenario, in dem im Editor bereits mehrere Elemente ausgewählt wurden.[6] Dies erkennt man an den Auswahlboxen rechts neben den einzelnen Termen. Dem einfachen Term mit dem Merkmal „Netzverband“ sind alle Elemente der Auswahl zugeordnet und dem ebenfalls einfachen Term mit dem Merkmal „Randfolien“ ist wenigstens ein Element der Auswahl zugeordnet. Weiterhin ist zu erkennen, dass für jedes Merkmal (durch # gekennzeichnet) zunächst ein einfacher Term mit dem gleichen Namen automatisch erstellt wird. Es werden also ausschließlich Terme zugeordnet und nicht direkt einzelne Merkmale. Zu jedem Term gehört neben dem Bezeichner auch eine Farbe, die sowohl zur Gruppierung, als auch zur Zuordnung im Editor verwendet werden kann.

Die abgebildeten Steuerelemente dienen dem Hinzufügen und Entfernen von komplexeren Termen. Das Bearbeiten erfolgt direkt in jeder Zelle der Tabelle durch Doppelklick. Darüber hinaus können die Terme neben dem Bezeichner auch nach der Farbe oder dem Status der Zuordnung sortiert werden, um die Übersicht während der Arbeit zu verbessern.

Für Merkmale mit Attributen, hier „Name“, wird zunächst ein Term mit

[6] Die Bedeutung der einzelnen Merkmale soll zunächst keine Rolle spielen.

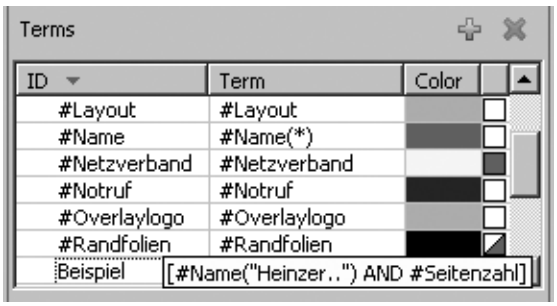


Abbildung 3.4
Termzuordnung

einem Platzhalter für beliebige Inhalte angelegt. Dieser Term findet meist in Modifikationen, wie im nächsten Abschnitt vorgestellt, Anwendung oder wenn der Wert des Attributs für die Existenz eines Dokumentelements irrelevant ist. Es können aber für Attribute auch konkrete Werte verwendet werden, wie der Term „Beispiel“ verdeutlichen soll. Dieser ist nur aktiv, wenn das Merkmal „Name“ aktiv ist, das Attribut den Wert „Heinzerling“ hat und wenn auch das Merkmal „Seitenzahl“ in der Variante ausgewählt wurde. Dies kann analog auf komplexere Terme und Kombinationen aus mehreren Attributen übertragen werden.

3.2.4 Modifikationen von Inhalten

Neben der bloßen Zuordnung von Termen zu Dokumentelementen ist es in der Abbildungsansicht auch möglich, Werte vorhandener Elemente oder Attribute zu verändern. Für die Situation in Abbildung 3.5 wurde ein Absatz im Editor ausgewählt. Anschließend ist es möglich, eine Veränderung einem Term zuzuordnen. Hier wird bei Aktivität des Terms, der nur das Merkmal „Titel“ enthält, nicht der Text durch eine fixe Zeichenkette ersetzt, sondern auf die Variantenbeschreibung zurückgegriffen und der dort angegebene Wert für das titel-Attribut für das Titel-Merkmal eingesetzt. Anders formuliert: Welcher Titel auf dem Dokument erscheint, kann über die Variantenspezifikation definiert werden, ohne sämtliche Möglichkeiten im Mapping abzubilden.

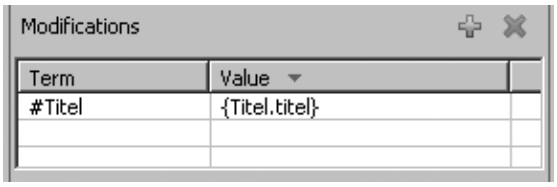


Abbildung 3.5
Modifikationen

Aus Mangel an umfassenderer praktischer Erfahrung wird bei Konflikten zwischen mehreren Modifikationen derzeit immer die erste angewendet und die Weiteren ignoriert. Der Interpreter für XML gibt in diesem Fall eine Warnung aus. Vorstellbar ist aber auch eine Ordnung nach Termkomplexität, unter der

Annahme, dass komplexere, größere Terme meist spezifischer sind und damit einfachere Terme überschreiben, oder basierend auf einer Subsumtion über den Termen. Zur Auflösung dieser Konflikte oder Warnungen können die Terme um zusätzliche Merkmale konjunktiv erweitert werden, so dass immer nur ein Term davon in einer Variante aktiv sein kann. Zum Beispiel aus dem möglichen Konflikt #Titel und #Titel $\wedge$ #Autor wird so #Titel $\wedge\neg$ #Autor und #Titel $\wedge$ #Autor .

3.2.5 Expertenansicht

Bei der Umsetzung der Anwendungsfälle, die später in Kapitel 4 genauer betrachtet werden sollen, hat sich herausgestellt, dass es bei einem Benutzer mit fortgeschrittenen Kenntnissen des Open Document Formats und XPath die Qualität der Abbildung hinsichtlich Aufwand des Einrichtens, Wartbarkeit und Stabilität gegen Änderungen deutlich verbessert, wenn Identifikatoren direkt eingegeben werden können. Abbildung 3.6 zeigt die Expertenansicht, die diesem Sachverhalt Rechnung trägt.

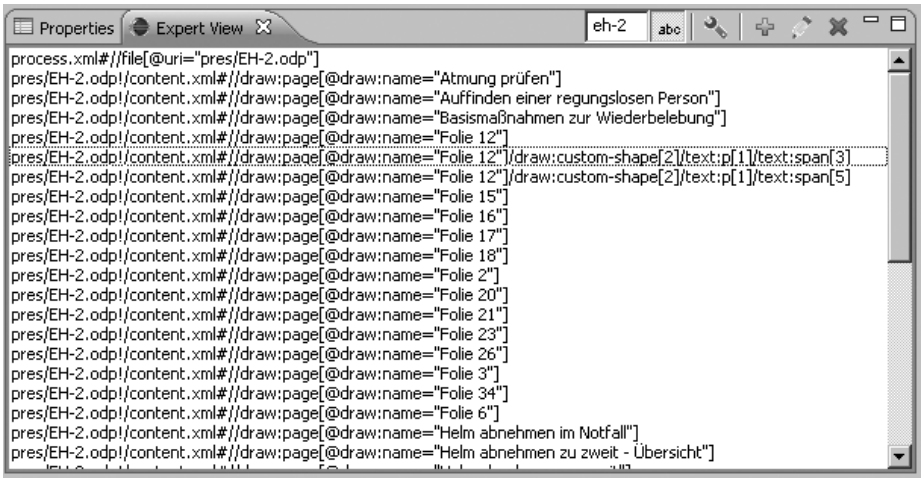


Abbildung 3.6
Expertenansicht

Im unterem Teil sind sämtliche Identifikatoren aufgelistet, die in der Merkmalszuordnung Anwendung finden. Diese können, wie in der Abbildung dargestellt, durch einen regulären Ausdruck gefiltert werden, um die Übersicht zu erhöhen. Die Schaltfläche neben der Filtereingabe aktiviert die Unterscheidung zwischen Groß- und Kleinschreibung.

Neben der bloßen Auflistung können über die Expertenansicht auch diverse Reparaturwerkzeuge (vgl. Abschnitt 3.4) auf das Mapping angewendet werden.

Die drei weiteren Schaltflächen erlauben das Hinzufügen und Löschen von Identifikatoren, sowie das Bearbeiten von Identifikatoren unter Beibehaltung der Merkmalszuordnung.

Durch die Auswahl eines Identifikators aus der Liste können analog zur Auswahl im Editor Modifikationen und Terme zugewiesen werden.

3.2.6 Varianteneditor

Nachdem die Abbildung der Merkmale auf Dokumentelemente und das Spezifizieren von Modifikationen abgeschlossen ist, kann über den Varianteneditor eine Variantendefinition unter Angabe eines Featuremodells erstellt oder eine vorhandene geöffnet werden.

In Abbildung 3.7 ist eine Beschreibung einer Variante im Editor dargestellt. Es ist zu erkennen, dass die Merkmale „Projekt“, „Ungültige Tage“, „Auswärts“ und „Monat“ in dieser Variante aktiv sind und auch die Bedingung der Gruppe ≥ 0 erfüllen.[7]

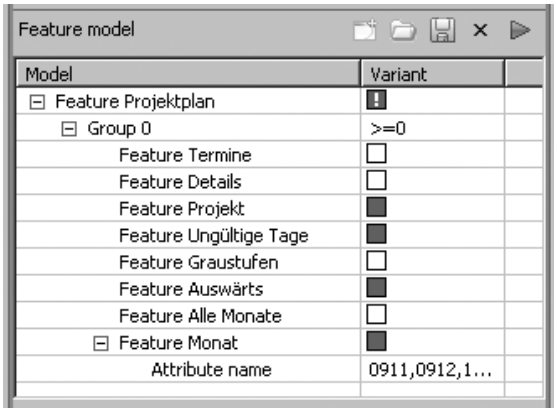


Abbildung 3.7
Varianteneditor mit Variantendefinition

Jedoch wurden noch nicht alle nötigen Merkmale ausgewählt. Das Merkmal „Projektplan“ ist ebenfalls noch durch den Benutzer auszuwählen, da untergeordnete Merkmale aktiviert wurden. Ein im Original rotes Ausrufezeichen trägt dem Rechnung. Analog wird ein rotes Fragezeichen angezeigt, wenn nach Auflösung der Gruppengröße, Eltern-Kind-Relation und einfacher Abhängigkeiten[8] in den Nebenbedingungen keine korrekte Auswahl des Merkmals mehr möglich ist. Dabei erfolgt aber bewusst zu keinem Zeitpunkt eine automatische Auswahl durch den Editor. Es steht dem Benutzer immer frei, auch ungültige Varianten zu generieren.

[7] Im Featuremodell ist als obere Grenze -1 im Sinne von „beliebig“ für die Gruppe zu spezifizieren. Nicht erfüllte Bedingungen werden rot unterlegt.

[8] Im Featuremodell können Nebenbedingungen erstellt werden, deren Attribut „Sprache“ den Wert „SD“ für „Simple Dependencies“, ein minimales, *DocumentFeatureMapper-spezifisches* Format für Abhängigkeiten zwischen Merkmalen, enthält. Als Ausdrücke können dann Zeichenketten mit dem Muster `((\+|-) ([^\+-]+))+` verwendet werden, die Merkmalsnamen mit vorangestelltem `+` für notwendig und `-` für verboten enthalten, z. B. `+Titel des Dokuments -Name`

Die Angabe von Attributwerten kann, wie dargestellt, direkt im Editor neben dem entsprechend Attribut erfolgen.

Wie bereits in der Vorbetrachtung erwähnt, soll auf das Klonen von Merkmalen in dieser Version des *DocumentFeatureMappers* noch verzichtet werden, daher findet es im Variantenеditor zunächst keine Beachtung.

3.2.7 Assistent zur Generierung von Instanzen der Dokumentfamilie

Über den in Abbildung 3.7 sichtbaren grünen Pfeil öffnet sich der in Abbildung 3.8 dargestellte Assistent und es kann ein konkretes Dokument bzw. eine Gruppe von Dokumenten aus den Quelldokumenten generiert werden.

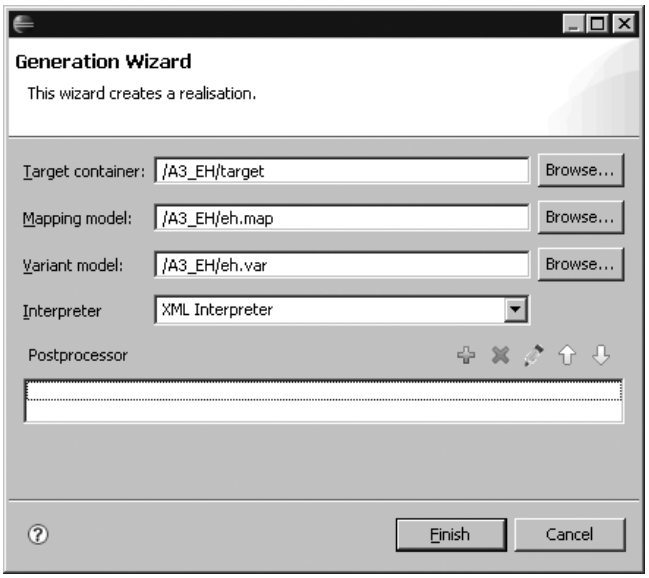


Abbildung 3.8
Generierungsassistent

Sind ein Mapping oder eine Variante geöffnet, werden diese automatisch übernommen, anderenfalls können sie im Dialog ausgewählt werden. Neben dem Zielverzeichnis ist auch noch ein Interpreter (vgl. Abschnitt 3.5) zu wählen und es besteht die Möglichkeit, weitere Nachbearbeitungsschritte (vgl. Abschnitt 3.6) zu konfigurieren.

3.3 Editor

Den Ausgangspunkt für die im vorangegangen Abschnitt vorgestellte Abbildungsansicht bildet ein *Editor*, der die nötigen Identifikatoren für die Merkmalszuordnung zur Verfügung stellt. Im Allgemeinen geschieht dies durch einen Benutzer mit der Auswahl von Inhalten im Editor.

Editor

Bevor die Schwierigkeiten bei der Identifikation von Elementen näher betrachtet werden und wie diesen zu begegnen ist, werden im Folgenden die zwei Editoren des *DocumentFeatureMappers* kurz vorgestellt. Die Abschnitte B.2.2 ff. enthalten weitere Details zur konkreten Realisierung weiterer Editoren bzw. zur Ergänzung der vorhandenen.

3.3.1 XML- und ODF-Baumeditor

Beiden Editoren liegt die selbe Basis zu Grunde, so dass deren Funktionsumfang in einem gemeinsamen Abschnitt behandelt werden kann. Den einzigen Unterschied bilden die dargestellten Daten. Der XML-Editor ist in der Lage, jede beliebige XML-Datei[9] als Baum sämtlicher Elemente darzustellen. Der ODF-Editor erwartet hingegen ein ZIP-Archiv mit den ODF-Dateien, die den Inhalt und das Layout[10] beschreiben. In beiden Varianten erhält jede so eingelesene Datei einen eigenen Reiter im Editor.

Filterung und Beschriftung der dargestellten Elemente

XML-Dateien sind nicht immer dazu ausgelegt, direkt dem Benutzer als Baum angezeigt zu werden. Eine vollständige Inhaltsbeschreibung in ODF ab mittlerer Dokumentgröße ist selbst bei tieferem Verständnis des Formats nicht mehr ohne weiteres schnell zu erfassen.

Um einem Endbenutzer nicht die vollständige XML-Struktur anbieten zu müssen, stellen die Editoren diverse Filter zur Verfügung, um die Anzeige auf ein überschaubares Maß einzuschränken.[11] Für ODF-Textdokumente kann es schon ausreichen, nur die drei Elemente Überschrift, Absatz und Textbereich darzustellen, und ein Dokument kann vollständig erfasst werden. Möchte der Benutzer hingegen eine Präsentation bearbeiten, weiß aber, dass die Folien selbst unverändert bleiben, reicht es völlig aus, nur die Folien aufzulisten. Analog kann eine Entscheidung getroffen werden, ob Animationen, Grafiken, Bilder, Listen oder Tabellen in der gefilterten Ansicht auftauchen sollen. Einem Endbenutzer hingegen, der sich fundierteres Wissen angeeignet hat, steht es frei, sich z. B. zusätzlich noch die Definition der Formatvorlagen mit anzeigen zu lassen. (Abbildung 3.9)

Neben der reinen Menge an Informationen kommen für die meisten Benutzer noch die kryptischen und gewöhnlich englischen Bezeichner von XML-Elementen und Namensräumen erschwerend hinzu. In einem Werkzeug, dessen Schwerpunkt auf Praxistauglichkeit liegt, ist dieser Punkt selbstverständlich nicht zu

[9] Gegebenenfalls sind von *.xml abweichende Endung dem Editor noch bekannt zu machen.
[10] content.xml und style.xml; auf die weiteren möglichen Dokumente wurde zunächst verzichtet.
[11] Es werden nur anwendbare Filter angezeigt und mit einer in der Praxis bewährten Vorauswahl aktiviert.

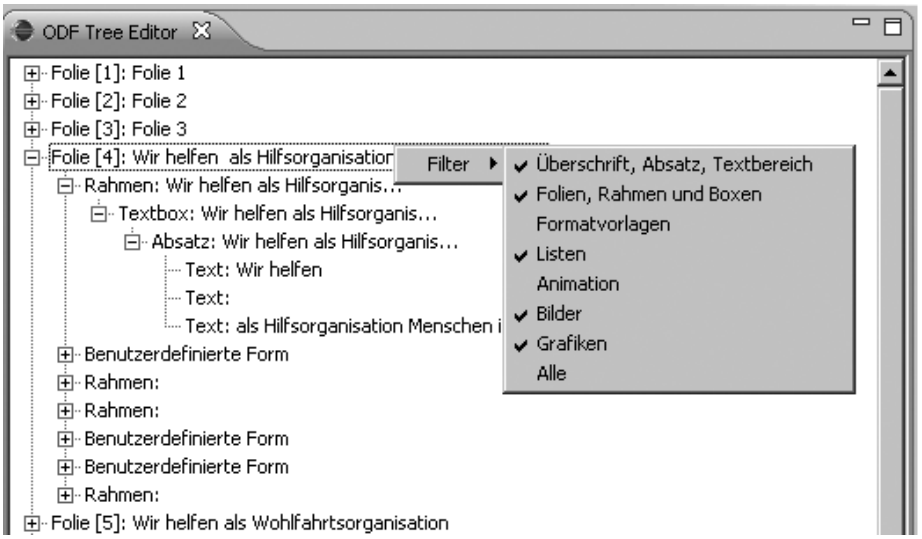


Abbildung 3.9
Filtern der Baumanicht

unterschätzen. Es besteht somit die Möglichkeit, für jedes XML-Element eine geeignete Bezeichnung zu definieren. Für die gängigen Elemente im Open Document Format liegen diese bereits dem *DocumentFeatureMapper* bei. Zum Beispiel wird aus einem `draw:page`-Element für den Benutzer ein „Folie X: Folientitel“ unter Hinzunahme seiner Position unter dem Elternelement und der Auswertung des `draw:name`-Attributes. Leider kann dies nicht vollständig automatisiert werden, so dass ein Generieren aus dem Schema nicht möglich ist, und dies für jedes Element manuell optimiert werden muss. Für Elemente bei denen diese Spezifikation nicht existiert, wird auf den qualifizierenden Namen des Elements zurückgefallen.

Operationen auf Elementen

Die Editoren sind nicht dazu konzipiert z. B. vollständige ODF-Dokumente aus dem Nichts zu erstellen, dennoch können Elemente mit zusätzlichen Operationen versehen werden, die Änderungen an der XML-Struktur vornehmen. Vorstellbar wäre z. B. eine Funktion, die einen Absatz in mehrere Textbereiche zerlegt, weil nur ein Teil davon ausgeblendet werden soll. Im weiteren Verlauf dieses Abschnittes soll dazu noch ein bereits implementiertes Beispiel aufgezeigt werden (Abbildung 3.10). Wie weitere Operationen zu implementieren sind, kann dem Anhang B.2.2 entnommen werden.

Erweiterung der Eigenschaftsansicht

Neben den XML-Elementen sollen ebenfalls die XML-Attribute durch eine Merkmalszuordnung veränderbar sein. Um dieses zu gewährleisten, wurde zu-

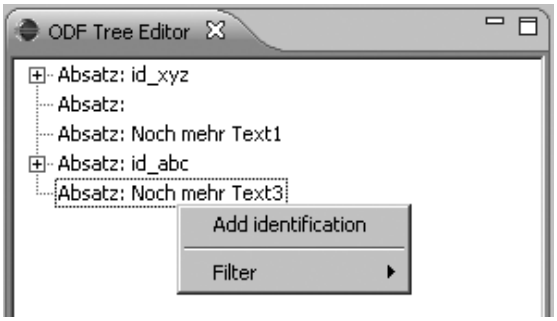


Abbildung 3.10
Hinzufügen einer eindeutigen Identifikation

nächst die innerhalb von Eclipse standardmäßig enthaltene PropertyView für die Darstellung der Attribute verwendet. Analog zu den Elementen können die Attribute durch eine geeignete Gruppierung und verbesserte Bezeichnungen dem Benutzer einfacher zugänglich gemacht werden. Darüber hinaus ist aber der dazugehörige Editor für das Bereitstellen von Identifikatoren verantwortlich, wenn eine Auswahl innerhalb der Eigenschaftsansicht stattfindet, um diese dann wie gewohnt innerhalb der Abbildungsansicht zuordnen zu können.[12] (Abbildung 3.11)

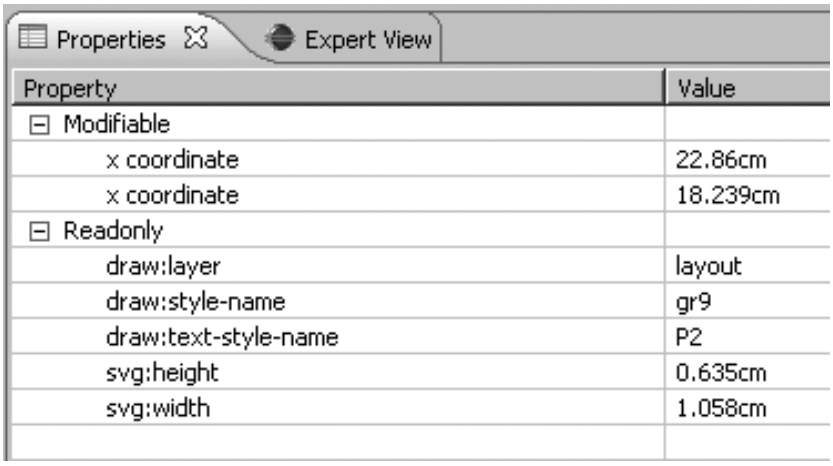


Abbildung 3.11
Eigenschaftsansicht

Einfärbung der Merkmalszuordnung

Ist parallel zum Editor ein Mapping in der Abbildungsansicht geöffnet, in deren Quelldokumenten sich die geöffnete Datei befindet, wird der Status der Abbildung an den Editor übertragen. Durch die Wahl von Farben in der Abbildungsansicht kann die Zuordnung direkt im Editor sichtbar gemacht werden. (Abbildung 3.12)

[12] Die PropertyView ist nicht dazu ausgelegt, selbst als SelectionProvider zu arbeiten, daher muss auf Änderungen des darunter liegenden Widgets gelauscht werden.

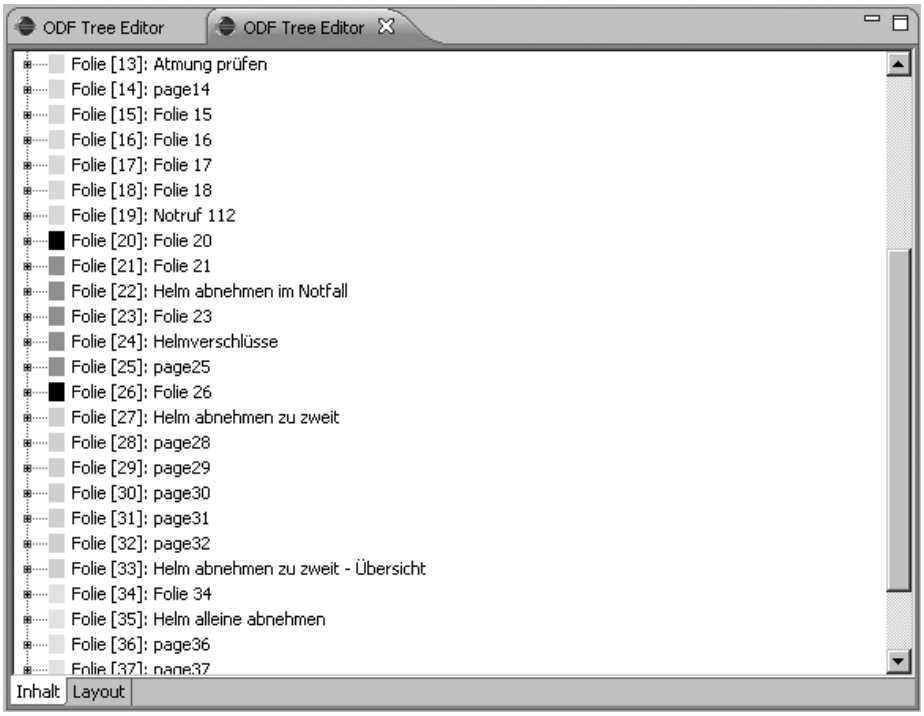


Abbildung 3.12
Visualisierung der Merkmalszuordnung

3.3.2 Eindeutige stabile Identifikation von Ressourcen

Für die Akzeptanz eines Benutzers und für eine Verwendung über akademische Beispiele hinaus ist es entscheidend, dass das Bearbeiten eines Dokuments nicht dazu führt, dass die Abbildung der Features neu und im schlimmsten Fall vollständig manuell erfolgen muss. Das vollständige Ersetzen der alten URIs durch die angepassten Formen lässt sich nur mit erhöhtem Aufwand realisieren. Daher gilt es, das Ändern von URIs bereits im Vorfeld wo immer möglich zu vermeiden.

Instabiler URI

Listing 3.2 zeigt einen *instabilen Identifikator*, den wir zunächst im Detail betrachten wollen. Der Beginn enthält den Hinweis, dass es sich um ein Archiv handelt, gefolgt von einer Pfadangabe[13] zu einer konkreten Datei. Der mittlere Teil vom Ausrufezeichen bis zum Beginn des Fragments (#) beschreibt den Pfad innerhalb des Archivs. Diese beiden Abschnitte geben also an, dass auf einer Präsentation im Open Document Format gearbeitet wird und deren content.xml verändert werden soll. Soweit ist der Identifikator zunächst *semistabil* (vgl. Listing 3.3 I). Ausschließliche Änderungen am Dateinamen oder am Pfad der Datei können zu einer Änderung des URIs führen. Beides liegt außerhalb der Reichweite des Editors und ist zunächst nicht zu verhindern oder zu überwachen. Der Identifikator erhält dennoch das Prädikat semistabil, da eine Anpassung der Abbildung, mit geringem Aufwand für den Benutzer, semiautomatisch

Semistabiler URI

[13] „...“ kann sowohl ein URI mit dem Protokoll file: als auch platform: im Eclipsekontext sein

ausgeführt werden kann. (vgl. Abschnitt 3.4.1)

```
zip:...sample.odp!/content.xml#/office:document-
        content/office:body/office:presentation/
        draw:page[7]/text:p[3]/text:span[2]"
```

Listing 3.2
Instabiler Identifikator

Wenn also nicht die Datei selbst das Problem ist, muss der XPath-Ausdruck im Fragment des URIs der Auslöser der Instabilität sein. Jedes Einfügen eines Elements in das zugrunde liegende XML-Dokument, das eine Verschiebung der Indizes bewirkt, führt also dazu, dass der URI auf ein anderes Ziel verweist und dennoch gültig ist.[14] Übersetzt heißt dies, man kann keine Seite vor der siebenten Seite in die Präsentation einfügen. Innerhalb der siebten Seite kann man keinen neuen Absatz vor dem dritten einfügen. Und dasselbe gilt analog auch für neue Bereiche vor dem zweiten Bereich des dritten Absatzes der siebten Seite der Präsentation.[15]

Es gilt also zunächst, eine präzisere Adressierung für Elemente innerhalb eines Dokuments zu finden. Erschwerend kommt an dieser Stelle für diesen konkreten Anwendungsfall und die gewählte Konfiguration der Untersuchung hinzu, dass OpenOffice.org sämtliche zusätzlich eingefügten Elemente oder Attribute innerhalb eines Dokuments, die nicht bekannt sind, verwirft, wenn dieses gespeichert wird. Dies wiederum bedeutet, dass ausschließlich Elemente und Attribute des Open Document Formats zur Identifikation verwendet werden können.[16] Darüber hinaus gibt es auch Attribute, deren Verwendung OpenOffice.org nur eingeschränkt zulässt, so wird z. B. das `text:id`-Attribute innerhalb von Absätzen und Überschriften entfernt, wenn keine Referenz zu diesem Absatz besteht.

Zusammenfassend lässt sich die Erkenntnis festhalten, dass für jedes Element eine individuelle Lösung gefunden werden muss, um die Instabilität aufzulösen. Exemplarisch sei hier zunächst eine Präsentationsfolie (`draw:page`) genannt, diese besitzt ein Attribut `draw:name`, welches den Namen der Folie kennzeichnet. Da sich der Name wieder außerhalb der Kontrolle des Editors bzw. des *DocumentFeatureMappers* ändern kann, aber dies recht einfach semiautomatisch korrigiert werden kann (vgl. Abschnitt 3.4.2), fällt dieser Identifikator in die Klasse semistabil (Listing 3.3 II und III)).

Einige Elemente wie z. B. Rahmen (`draw:frame`), Absätze (`text:p`) oder Überschriften (`text:h`) können ein ID-Attribut (`draw:id` bzw. `text:id`) besitzen. Dieses ist auf Grund der OpenOffice.org zugrunde liegenden Optimierung

[14] Ausnahme beim Löschen
[15] Beim Löschen analog
[16] Nach http://xml.openoffice.org/faq.html#5 sind weitere Elemente nur in style:properties erlaubt. Dies hilft aber hier nicht.

```
  (I) zip:...sample.odp!/content.xml#/
              (vgl. nächstes Listing)
 (II) zip:...sample.odp!/content.xml#
              //draw:page[@draw:name="page7"]
(III) zip:...sample.odp!/content.xml#
              //draw:page[@draw:name="Seitentitel"]
```

Listing 3.3
Semistabiler Identifikator

nicht mehr benötigter Referenzids als instabil einzuordnen. Der Aufwand, eine gelöschte ID zu reparieren, setzt beim Benutzer Verständnis für die XML-Struktur der Dokumente voraus, kann aber über die Expertenansicht angegangen werden.

Da bereits die Angabe einer Quelldatei die Stabilität eines Identifikators negativ beeinflusst, sei im Weiteren ein *stabiler Identifikator* ausschließlich über die Stabilität des XPath-Fragmentes definiert.

Stabiler URI

Um trotz der vielen Einschränkungen möglichst viele Identifikatoren zu stabilisieren, insbesondere jene, die häufig Verwendung finden oder deren Instabilität für die Akzeptanz kritisch ist, muss ggf. auf etwas „unangenehmere“ Lösungen zurückgegriffen werden.

Exemplarisch sei hier das Kernelement eines Dokuments, der Absatz (`text:p`) genannt. Diesem kann eine Textmarke (`text:bookmark`) zugeordnet werden. Innerhalb von OpenOffice.org wird diese Marke verwendet, um schnell innerhalb großer Dokumente zu springen. Eine Textmarke besitzt darüber hinaus ein Attribut `text:name`, über den sie eindeutig identifiziert werden kann. Dieser Sachverhalt wird in Listing 3.4 in einen Identifikator umgesetzt. Es wird zunächst die Textmarke stabil identifiziert und anschließend zum Elternelement (/..) navigiert. Dieses Vorgehen nötigt nun aber den Benutzer dazu Absätze innerhalb des Editors ggf. vor der Verwendung manuell durch eine Auswahl im Kontextmenü zu stabilisieren und eine Textmarke einfügen zu lassen. Ein bereits vorhandenes Mapping auf diese Elemente oder auf untergeordnete Elemente wird dabei aber vollautomatisch angepasst.

```
zip:...sample.odp!/content.xml#//text:bookmark
        [@text:name="Beliebiger␣Name"/..]
```

Listing 3.4
Stabiler Identifikator

Bedauerlicherweise kommt es an dieser Stelle wieder zu einer Diskrepanz zwischen der Theorie, dem Open Document Format, und der Praxis, den OpenOffice.org-Anwendungen. Auch wenn die Textmarken formal in jedem Absatz unabhängig vom Dokumenttyp zulässig sind, startet OpenOffice.org Im-

press nicht mit einer Präsentation, die eine Textmarke enthält, d. h. dies funktioniert nur in Textdokumenten.[17] Dies führt wieder zurück zu der Erkenntnis, dass die Grundlagen dieses Ansatzes nicht ausschließlich die Spezifikation oder das Schema von ODF sein können, sondern dass für jedes Element und jedes Attribut eine individuelle Lösung zu finden ist, und diese auch praktisch erprobt werden muss. Auf die konkrete Umsetzung weiterer Elemente, die nicht in den Fallbeispielen Anwendung finden, wurde daher verzichtet. Diese können dezentral während einer umfassenderen Erprobung des *DocumentFeatureMappers* nach Bedarf erstellt werden.

3.3.3 OpenOffice.org-Ansicht

Die wenigsten Endnutzer sind sich der Baumstruktur ihrer Dokumente bewusst, somit wäre die ausschließliche Darstellung in der XML-Baumansicht für ODF-Dokumente meist nicht sehr intuitiv. Um die Errichtung einer mentale Brücke zwischen Dokument und Baum auch für diese Benutzer zu ermöglichen, wurde einen OpenOffice.org-Ansicht als Ergänzung zur Baumansicht im *DocumentFeatureMapper* hinzugefügt. Diese basiert auf einer externen Erweiterung für Eclipse mit dem Titel NOA4e. Eine detaillierte Beschreibung der Ansicht erübrigt sich, da diese ausschließlich OpenOffice.org innerhalb von Eclipse öffnet, sonst aber keine Unterschiede zur „normalen“ Version aufweist.[18] (Abbildung 3.13)

Der *DocumentFeatureMapper* übernimmt die Kommunikation zwischen der Baumansicht und der OpenOffice.org-Ansicht und sorgt für die Synchronisierung der Inhalte sowie dem Freigeben der Sperren durch OpenOffice.org auf die Dokumente.

In der Praxis hat sich herausgestellt, dass gerade bei größeren Dokumenten eher davon abzusehen ist, in beiden Repräsentationen des Dokuments abwechselnd Änderungen vorzunehmen, da Änderungen nicht partiell propagiert werden können, und OpenOffice.org das Dokument jedes mal wieder komplett einlesen muss.

3.3.4 Integration vorhandener Editoren

Für die zukünftige Ergänzung um neue Editoren bietet die Abbildungsansicht noch einen Erweiterungspunkt, um bereits existierende Editoren über einen Adapter anzubinden. Die Adapter sind bidirektional angelegt, so dass die Auswahl

[17] In Präsentationen muss man versuchen, „versteckte“ Referenzen auf die Absätze entweder über die Änderungsverfolgung des ODFs oder über Animationen zu erzeugen. Wichtig ist dabei, daran zu denken, dass OpenOffice.org „jederzeit“ diese IDs verändern könnte.

[18] Eine OpenOffice.org-Installation wird vorausgesetzt.

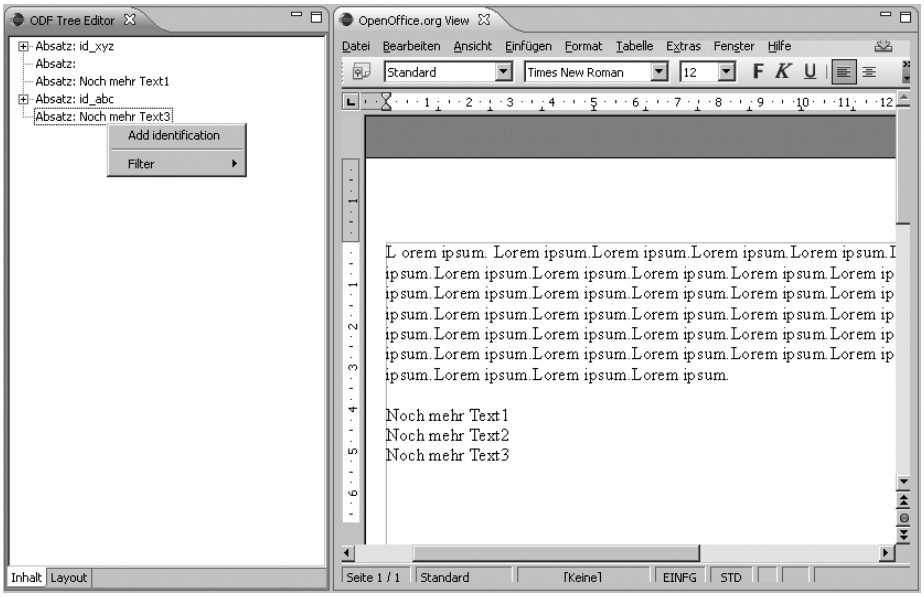


Abbildung 3.13
OpenOffice.org-Ansicht mit Textdokument

des Editors in eine Menge von Identifikatoren für die Abbildungsansicht umgewandelt, aber auch der Status der Zuordnung zurück an den Editor propagiert werden kann, um z. B. Elemente einzufärben. Exemplarisch wurde ein prototypischer Adapter für EMF-Editoren beigefügt. Damit ist zwar das Erstellen eines Mappings möglich, jedoch ist bisher kein EMF-Interpreter im *DocumentFeatureMapper* enthalten, um dieses auch in Varianten umzusetzen.

Abschnitt B.2.4 führt dies näher aus.

3.4 Reparatur

Im vorangegangenen Abschnitt wurde bereits aufgezeigt, dass eine stabile Identifizierung von Elementen im Quelldokument nur bedingt möglich ist. Um die daraus für den Benutzers resultierenden Unannehmlichkeiten der Neuzuordnung von Merkmalen nach Änderungen am Quelldokument zu reduzieren, bietet der *DocumentFeatureMapper* eine Schnittstelle für *Reparaturwerkzeuge* an. Diese können dann semiautomatisch mit Unterstützung des Benutzer oder vollautomatisch die Identifikatoren reparieren, d. h. entweder Teile oder den ganzen Identifikator durch die aktualisierte Adressierung ersetzen.

Reparatur

Die Reparaturwerkzeuge können in dem GUI in der Expertenansicht gezielt durch den Benutzer ausgewählt und angewendet werden (Abbildung 3.14). Dabei erhält er durch die Auswahl eines Identifikators bereits Vorschläge für die einzelnen Parameter des Werkzeuges. Darüber hinaus besteht aber auch die Möglichkeit, dass ein Editor eine Reparatur auslöst, wenn sich während der

Bearbeitung ein Identifikator verändert.

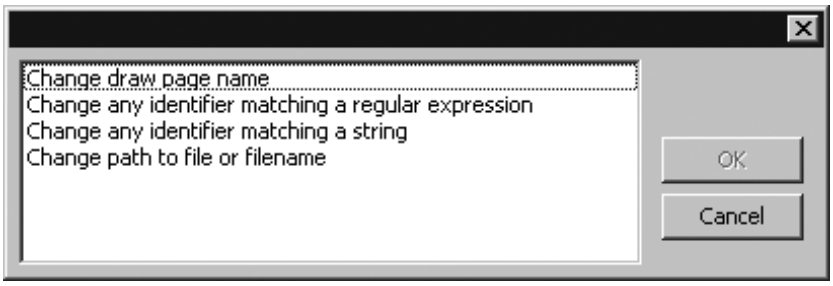


Abbildung 3.14
Liste der Reparaturwerkzeuge

Im Folgenden werden vier exemplarische Reparaturwerkzeuge näher betrachtet. Abschnitt B.2.5 erläutert das Vorgehen zur Implementierung eines eigenen Werkzeugs.

3.4.1 Umbenennen von Dateien

Eines der vermutlich häufigsten Szenarien, aber auch eines, das die meisten ungültigen Identifikatoren nach sich ziehen kann, ist das Umbenennen von Dateien, die Teil des Mappings sind. Da alle Identifikatoren innerhalb eines Dokuments mit dem Dateinamen beginnen, wäre alternativ eine manuelle Anpassung über die Abbildungsansicht mit einer vollständigen Neuzuordnung nötig.[19]

Eingabe:

target Zeichenkette mit Pfad- oder Dateiangabe oder ein Ausschnitt davon, um mehrere Dateien zu „verschieben“ oder „umzubenennen“

replacement Ersetzung

Anwendung: Alle Dateien

3.4.2 Ändern des Präsentationsfolientitels

Wie in Abschnitt 3.3 aufgezeigt, eignet sich für die Identifikation von Präsentationsfolien in einer Datei im Open Document Format am besten der Titel der Folie. Da sich dieser aber ändern, oder überhaupt erst vergeben werden kann, nachdem die Zuordnung erfolgt ist, ist ein entsprechendes Werkzeug notwendig.[20]

Eingabe:

file Zeichenkette mit Pfad- oder Dateiangabe oder ein Ausschnitt davon; dient als Einschränkung bei gleichen Folientiteln über mehrere Dokumente hinweg (z. B. Standardwerte „page 4“)

target Zeichenkette mit aktuellem (partiellen) Folientitel

replacement neuer Titel oder Titelausschnitt

[19] Javadoc: de.mheinzerling.dfm.repair.FileNameRepair
[20] Javadoc: de.mheinzerling.dfm.repair.PageNameRepair

Anwendung: Präsentationen im Open Document Format

3.4.3 Generische Zeichenkettenersetzung

Für Benutzer mit einem umfassenderen Verständnis der Identifikatoren[21] können mit Hilfe dieses Reparaturwerkzeugs Teile von Identifikatoren durch andere Zeichenketten ersetzt werden. Damit sind Korrekturen möglich, für die es bisher noch kein spezifisches Werkzeug gibt.[22]

Eingabe:

target Zeichenkette; Teil eines oder mehrerer Identifikatoren

replacement Ersetzung

Anwendung: Alle Dateien

3.4.4 Generische Ersetzung mit regulären Ausdrücken

Als Erweiterung für die reine Zeichenkettenersetzung können hier auch reguläre Ausdrücke verwendet werden.[23]

Eingabe:

target Java-Regulärer-Ausdruck; Teil eines oder mehrerer Identifikatoren

replacement Ersetzung (kann Gruppenreferenzen enthalten)

Anwendung: Alle Dateien

3.5 Interpreter

Die Realisierung einer Variante eines Merkmalsmodells erfolgt durch einen *Interpreter*. Dieser wertet dazu die den Dokumentelementen zugeordneten Ausdrücke aus und bestimmt die Menge der zu entfernenden oder modifizierenden Dokumentinhalte. Anschließend werden diese Änderungen an einer Kopie des Quelldokuments ausgeführt. Interpreter

Dem *DocumentFeatureMapper* liegt ein Interpreter für XML-Dateien bei, dessen Arbeitsweise im Folgenden detaillierter betrachtet werden soll. In Abschnitt B.2.1 sind weitere Hinweise zu finden, wie eigene Interpreter entwickelt werden können.

3.5.1 Interpreter für XML

Der Interpreter für XML kann ohne weitere Anpassungen auf sämtliche XML-Dialekte angewendet werden. Darüber hinaus können sich die XML-Dateien

[21] Sowohl die aktuell enthaltenen als auch zukünftige

[22] Javadoc: de.mheinzerling.dfm.repair.StringRepair

[23] Javadoc: de.mheinzerling.dfm.repair.RegexpRepair

auch in einem ZIP-Archiv befinden, wie es beim ODF vorgesehen ist.[24]

Identifikatoren

Zur Identifikation von Dokumentelementen wird bei diesem Interpreter ein URI mit XPath-Fragment wie in Listing 3.5 dargestellt verwendet.[25] Mit dem zusätzlichen Schema „regexp:“ können die Pfad- und Dateiangaben reguläre Ausdrücke enthalten, die gegen die Quelldokumente aufgelöst werden. So ist es möglich, Inhalte, die in mehreren Dateien auftreten, mit nur einem Identifikator zu beschreiben.[26]

```
[SCHEMA](regexp)?:/[PFAD][XMLDATEI]#[XPATH]

zip:[SCHEMA](regexp)?:/[PFAD_ZUM_ARCHIV][ARCHIV]!/
                       [PFAD_IM_ARCHIV][XMLDATEI]#
                       [XPATH]
```

Listing 3.5
Identifikatoren für XML-Dokumente

Arbeitsweise

Für das Generieren der Zieldokumente wird zunächst das Merkmalsmodell, basierend auf der zu erzeugenden Variante ausgewertet. In einem ersten Schritt werden alle im Modell verwendeten Ausdrücke berechnet, so dass jedem Ausdruck und auch jedem Merkmal ein boolescher Wert als Existenzbedingung zugeordnet ist. Darauf aufbauend können alle Identifikatoren aus dem Mapping ermittelt werden, bei denen alle zugeordneten Ausdrücke zu „falsch“ ausgewertet werden[27], um diese für das spätere Entfernen zwischen zu speichern. An dieser Stellen werden Identifikatoren mit regulären Ausdrücken gegen die möglichen Quelldokumente aufgelöst, so dass dann nur vollständige Identifikatoren in der weiteren Verarbeitung vorhanden sind. Analog zu den zu entfernenden Inhalten werden auch die zu modifizierenden Elemente ermittelt.

Da die Menge der Quelldokumente je nach Anwendungsgebiet auch recht umfangreich sein kann, aber eine Variante nicht zwingend immer alle Dokumente generieren soll, bietet der Interpreter für XML die Möglichkeit, sämtliche zu erzeugende Dateien in einer `process.xml` als Teil der Quelldokumente, wie in Listing 3.6 zu sehen, zu definieren. Diese Datei kann in die Merkmalszuordung

[24] Javadoc: de.mheinzerling.dfm.interpreter.xml.XMLInterpreter
[25] Wenn dieser Interpreter Verwendung finden soll, muss der für das Mapping verwendete Editor Identifikatoren in dieser Art bereitstellen.
[26] In Abschnitt 4.3 können so alle Animationen in sämtlichen ODF-Präsentationen mit nur einem Identifikator adressiert und ggf. entfernt werden.
[27] Elemente ohne Merkmalszuordnung bleiben somit erhalten.

einbezogen werden, so dass einzelne Dateien nur bei bestimmten Merkmalen erzeugt werden. Hierzu wird die `process.xml`, sofern vorhanden, vom Interpreter immer zuerst behandelt und basierend auf der generierten Version eine Liste der im Weiteren zu bearbeitenden Dateien erstellt. Alle nicht in dieser Liste enthaltenen Dateien werden nicht im Zielverzeichnis erzeugt.

```
<?xml version="1.0" encoding="UTF-8"?>
<fileList>
  <file uri="[RELATIVER_PFAD_ZUR_DATEI][XMLDATEI]"/>
  ...
</fileList>
```

Listing 3.6
process.xml - Liste der zu erzeugenden Dateien

Nachdem diese Vorbereitungen abgeschlossen sind, werden Quelldateien ohne Änderungen am Inhalt direkt in das Zielverzeichnis kopiert, und Dateien mit Änderungen zunächst kopiert, anschließend die zu löschenden Elemente entfernt und die Modifikationen angewendet. Der Interpreter für XML führt keine Form einer Validierung gegen ein Schema aus. Dies kann entweder in einem Nachbearbeitungsschritt (vgl. Abschnitt 3.6) geschehen oder innerhalb des Editors durch eine eingeschränkte Auswahl der zu löschenden oder zu modifizierenden Elemente.

Für das Erstellen von Berichten oder um Nachbearbeitungsschritte mit zusätzlichen Informationen zu versorgen (vgl. Abschnitt 3.6.6) stellt der Interpreter für XML eine Schnittstelle zur Verfügung, die anderen Komponenten ein Lauschen auf ausgeführte Änderungen erlaubt. Es lässt sich also detailliert verfolgen, welche Elemente entfernt wurden. Weitere Details sind Abschnitt B.2.1 zu entnehmen.

3.6 Nachbearbeitung

Nachdem die Zieldokumente durch den Interpreter entsprechend einer Variante generiert wurden, ist es häufig nötig, diese noch weiter zu bearbeiten. Da der Interpreter selbst nicht zwingend die Validität der erzeugenden Dokumente sicherstellt, kann z. B. eine Reparatur und Prüfung gegen ein Schema in einem *Nachbearbeitungsschritt* erfolgen. Darüber hinaus sind aber auch Aufräumarbeiten oder Anpassungen der Zielordnerstruktur in einem solchen Schritt denkbar.

Nachbearbeitung

Jede Nachbearbeitung wird direkt auf dem Zielordner ausgeführt und ist unabhängig von weiteren Eingaben, einschließlich dem Featuremodell, der Variante oder den Quelldokumenten. Abschnitt 3.6.6 zeigt jedoch eine Möglichkeit auf, wie man zusätzliche Informationen ermitteln und dem Nachbearbeitungsschritt zur Verfügung stellen kann.

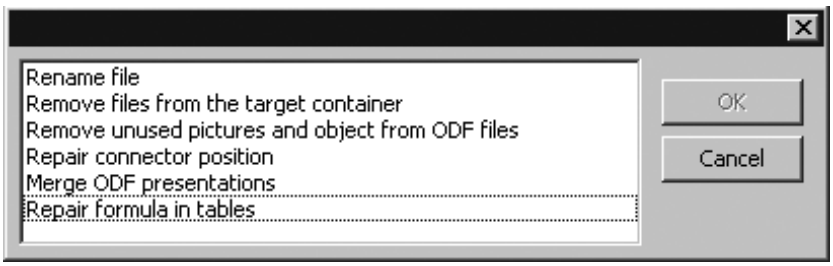


Abbildung 3.15
Liste der Nachbearbeitungsschritte

Im weiteren Verlauf sollen die dem *DocumentFeatureMapper* beigefügten Nachbearbeitungsschritte kurz vorgestellt werden. Abschnitt B.2.6 gibt Hinweise, wie ein eigener Schritt implementiert werden kann. (Abbildung 3.15)

3.6.1 Dateien löschen

Löscht Dateien aus dem Zielordner, die einem beliebigen Muster in Form eines Java-Regulären-Ausdrucks entsprechen.[28]

Eingabe:

pattern Java-Regulärer-Ausdruck; Muster der zu löschenden Dateien

Anwendung: Alle Dateien

3.6.2 Dateien umbenennen

Benennt eine Datei um[29].

Eingabe:

from aktueller Dateiname

to neuer Dateiname

Anwendung: Alle Dateien

3.6.3 Bereinigungen von ODF-Archiven

Damit nicht sämtliche Bilder und Objekte in einem Dokument im Open Document Format bei der Merkmalszuordnung manuell zugeordnet werden müssen, entfernt dieser Nachbearbeitungsschritt die nicht mehr referenzierten Bilder und Objekte aus dem ODF-Archiv, um die Dateigröße zu reduzieren.[30]

Eingabe:

pattern Java-Regulärer-Ausdruck; Muster der zu bereinigenden Dateien

Anwendung: Automatisch auf Dateien im Open Document Format

[28] Javadoc: de.mheinzerling.dfm.postprocessor.filecleanup.FileCleanupPostprocessor
[29] Javadoc: de.mheinzerling.dfm.postprocessor.filerename.FileRenamePostprocessor
[30] Javadoc: de.mheinzerling.dfm.postprocessor.objectcleanup.ObjectCleanupPostprocessor

3.6.4 Zusammenfügen von Präsentationen

Mit Hilfe dieses prototypischen Nachbearbeitungsschrittes kann eine beliebige Anzahl an ODF-Präsentationen hintereinander zu einer Präsentation zusammengefügt werden. Dabei werden zunächst die Bilder und Objekte sowie die automatischen und benutzerdefinierten Formatvorlagen zusammengeführt und mögliche Konflikte aufgelöst. Anschließend werden die einzelnen Folien in das Zieldokument kopiert.[31]

Eingabe:

descriptor Pfad zur Datei im Zielverzeichnis mit detaillierten Informationen über das Zusammenfügen

Anwendung: Automatisch auf Präsentationen im Open Document Format

Beschreibung des Zusammenfügens

Damit die Beschreibung für das Zusammenfügen bereits mit in den Generierungsprozess einfließen kann, ist diese als Bestandteil der Quelldokumente hinzuzufügen. Durch die Zuordnung von Merkmalen kann diese Beschreibung also verändert werden.

Listing 3.7 zeigt eine solche Beschreibung exemplarisch. Zu erkennen ist, dass die vier Präsentationen „f1.odp“ bis „f4.odp“ zu einer einzigen „merge.odp“ zusammengefügt werden sollen. Für das eigentliche Zusammenfügen haben die IDs und die Gruppierung keine Bedeutung, dennoch wird dadurch ein mögliches Mapping vereinfacht, wenn man in einer Variante z. B. gleich mehrere Dateien von der Nachbearbeitung ausschließen möchte. Die Gruppen können hierbei auch beliebig verschachtelt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<mergeDescriptor>
  <merge result="merge.odp" id="m1">
    <group id="g1">
      <source file="f1.odp" id="f1"/>
      <source file="f2.odp" id="f2"/>
    </group>
    <source file="f3.odp" id="f3"/>
    <source file="f4.odp" id="f4"/>
  </merge>
  ...
</mergeDescriptor>
```

Listing 3.7
Rudimentäre Beschreibung zum Zusammenfügen von Präsentationen

[31] Javadoc: ....postprocessor.presentationmerge.PresentationMergePostprocessor

3.6.5 Reparatur von Verbindungen in Grafiken

Im Open Document Format werden in Dokumenten, in denen Verbindungen zwischen Elementen innerhalb einer Grafik eingezeichnet sind, die Andockpunkte dieser Verbindungen für die visuelle Darstellung zwischengespeichert. Damit diese Andockpunkte beim Mapping nicht manuell neu berechnet und zugeordnet werden müssen, kann dieser Nachbearbeitungsschritt aktiviert werden, um den Zwischenspeicher basierend auf den neuen Koordinaten der Elemente zu aktualisieren.[32]
Eingabe: keine
Anwendung: Automatisch auf Dateien im Open Document Format

3.6.6 Reparatur von Formeln in Tabellen

Beim Bearbeiten von Tabellendokumenten werden in vielen Fällen auch ganze Zeilen in einem Zieldokument entfernt. Damit verlieren alle nachfolgend verwendeten Formeln innerhalb der Tabelle ihre Bezugspunkte, da diese über die Zeilennummer (z. B. D7) adressiert werden. Dieser Nachbearbeitungsschritt korrigiert, basierend auf einer Liste der entfernten Zeilen, die Zelladressen in Formeln oder bedingten Formatierungen.[33]
Eingabe: keine
Anwendung: Automatisch in Verbindung mit einer Erweiterung des Interpreters für XML-Dokumente

Ermittlung der entfernten Tabellenzeilen

Da der Interpreter für XML-Dokumente das Konzept einer ODF-Tabellenzeile nicht kennen kann und als Eingabe für die Nachbearbeitung nur das Zielverzeichnis dient, lauscht eine Komponente des Nachbearbeitungsschritts auf das Entfernen von Elementen durch den Interpreter (vgl. Abschnitt 3.5.1). Für jede darunter befindliche Tabellenzeile wird deren Nummer ermittelt und anschließend werden alle als Liste in das Zielverzeichnis geschrieben.[34]

3.7 Automatisierung

Während der durchschnittlichen „Lebensspanne“ einer Dokumentfamilie werden signifikant häufiger Dokumentinstanzen generiert, als die dazugehörigen Merkmalszuordnung verändert. Meist ist es also gar nicht nötig, die komplette grafische Oberfläche des *DocumentFeatureMappers* zu starten, nur um eine

[32] Javadoc: de.mheinzerling.dfm.postprocessor.connectorrepair.ConnectorRepairPostprocessor
[33] Javadoc: de.mheinzerling.dfm.postprocessor.tablerepair.TableRowRepairPostprocessor
[34] Javadoc: ....postprocessor.tablerepair.TableRowRepairXMLInterpreterListener

Instanz zu erzeugen. Hinzu kommt noch, dass sich die Instanzbeschreibungen relativ selten ändern und das Aktualisieren des Dokuments nur nötig ist, weil ein Quelldokument verändert wurde. Eine vollständige Neukonfiguration der Variante, des Interpreters und der Nachbearbeitungsschritte ist dabei unnötig.

```
<?xml version="1.0" encoding="UTF-8"?>
<automation>
  <task>
    <mapping
        class="...FeatureMapping">calc.map</mapping>
    <variant class="...Variant">leer.var</variant>
    <target>target</target>
    <interpreter>...XMLInterpreter</interpreter>
    <postprocessors>
      <postprocessor
          class="...TableRowRepairPostprocessor"/>
      <postprocessor
          class="...FileRenamePostprocessor">
        <property key="from">
          projektplan.ods
        </property>
        <property key="to">
          projektplan_leer.ods
        </property>
      </postprocessor>
      <postprocessor
          class="...ObjectCleanupPostprocessor">
        <property key="pattern"></property>
      </postprocessor>
    </postprocessors>
    ...
  </task>
  ...
</automation>
```

Listing 3.8
Exemplarische Beschreibung zur automatischen Generierung von Dokumentinstanzen

Listing 3.8 zeigt eine solche Spezifikation. Es ist zu erkennen, dass eine Automatisierung aus mehreren Aufgaben bestehen kann. Jede dieser Aufgaben enthält dieselben Informationen wie sie auch über das GUI konfiguriert werden würden. Hierzu zählt zunächst die Datei, die das Mapping enthält, eine Variante und der Interpreter. Für zukünftige Erweiterungen enthält jedes dieser Elemente noch einen Verweis auf die zugehörige Javaklasse[35]. Anschließend können beliebige Nachbearbeitungsschritte mit den im vorherigen Abschnitt vorgestellten Parametern angegeben werden.

[35] Es werden vollständige Bezeichner benötigt. Für die Übersicht wurde hier auf die Pakete verzichtet.

Mit der Möglichkeit, mehrere Aufgaben zur Generierung konkreter Instanzen in einer Beschreibung zusammenzufassen, kann also mit einem Befehl die komplette Dokumentfamilie bei einem Update aktualisiert werden.

Vorstellbar, aber im Rahmen dieser Arbeit nicht weiter untersucht, wäre auch das „Stapeln“ von Aufgaben durch die Anwendung auf den jeweils generierten Zielordnern. Damit ließ sich der hier vorgestellte Ansatz auf beliebige Ebenen von *Metadokumentfamilien* erweitern.

Metadokumentfamilie

Neben Vorteilen für den Benutzer erlaubt die Automatisierung auch die einfache Konfiguration von Testfällen. Exemplarisch können hier die Tests Case3BTest[36] oder Case2Test[37] als Vorlage dienen.

[36] Javadoc: de.mheinzerling.dfm.test.samples.Case3BTest
[37] Javadoc: de.mheinzerling.dfm.test.samples.Case2Test

Kapitel 4

Anwendungsszenarien

Das folgende Kapitel beginnt mit zwei kleineren Beispielen zur Anwendung des *DocumentFeatureMappers* und der Generierung von Dokumenten. Dazu wird zunächst eine Grafik in Form eines OpenOffice.org-Draw-Dokumentes (Abschnitt 4.1) vorgestellt, deren Komplexität eine Zerlegung in überschaubare Einheiten nahe legt. Analog dazu wird im Abschnitt 4.2 die Anwendung eines Featuremodells auf ein Tabellendokument näher betrachtet. Dabei werden zunächst die zugrunde liegenden Szenarien vertieft und jeweils ein geeignetes Featuremodell hergeleitet.

Der Abschnitt 4.3 überträgt dann den hier vorgestellten Ansatz auf eine umfangreiche Sammlung von Präsentationen aus der Erste-Hilfe-Ausbildung des Arbeiter-Samariter-Bundes (ASB) und veranschaulicht so nochmals die Umsetzung an komplexeren Dokumenten mit höheren Anforderungen an das Modell und das Werkzeug. Auch diesem Beispiel geht eine Analyse und die Erarbeitung eines geeigneten Featuremodells voraus.

Für sämtliche Beispiele, im Besonderen das Letzte, wurden bewusst vorhandene, real Verwendung findende Dokumente gewählt, um auch Erkenntnisse über die Praxistauglichkeit des hier zu untersuchenden Ansatzes gewinnen zu können.

Abschließend wird eine allgemeine Featureklassifikation für Dokumente abgeleitet. (Abschnitt 4.5)

4.1 Draw - Featurediagram

Nicht selten kommt es im Rahmen wissenschaftlicher Arbeiten oder Präsentationen zur Erstellung einer Vielzahl komplexer Diagramme oder (Vektor-)Grafiken. Dasselbe Szenario lässt sich aber ebenfalls im Bereich der Softwareentwicklung oder anderer Tätigkeiten mit einem hohen analytischen Anteil ansiedeln. Häufig sind die Grafiken für den Leser oder Zuhörer nicht mehr auf einen Blick zu erfassen, und eine inkrementelle Entwicklung würde das Verständnis deutlich verbessern. Eine Anpassung der Grafiken an diverse Zielgruppen oder die Erzeugung mehrerer Sichten kann analog umgesetzt werden.

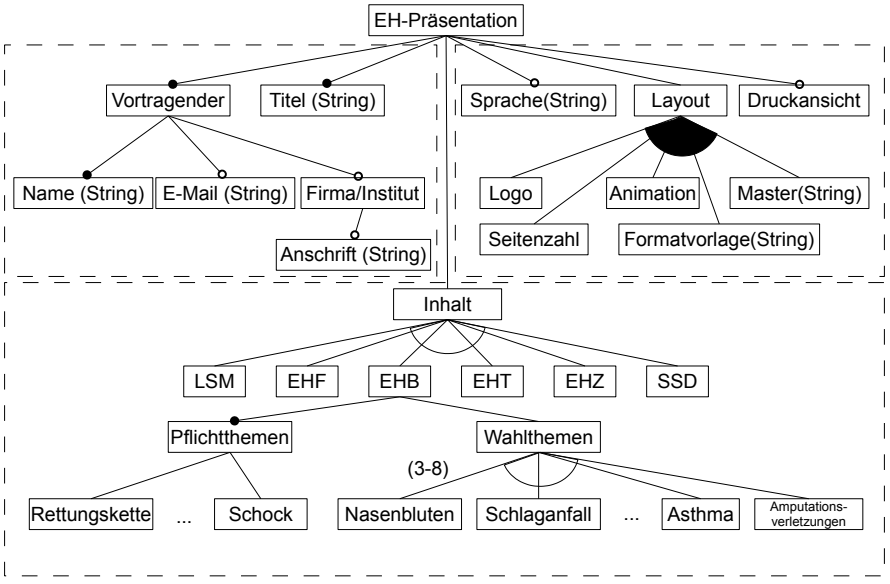


Abbildung 4.1
Komplexes
Featurediagramm

Als Beispiel soll hier ein Featurediagramm dienen, welches im Rahmen dieser Arbeit für den Anwendungsfall in Abschnitt 4.3 entwickelt wurde. Abbildung 4.1 zeigt eine bereits reduzierte Version, die nicht alle Merkmale enthält und dennoch im Allgemeinen schon nicht mehr auf einen Blick erfasst werden kann. Sowohl in schriftlichen Arbeiten, aber im Besonderen in einer mündlichen Präsentation möchte man dieses Diagramm inkrementell aufbauen und den Leser bzw. Zuhörer so durch das Diagramm führen.

Die Grafik verwendet Rechtecke, Verbindungen, Textboxen und benutzerdefinierte Formen (Kreisausschnitte), um die in Abschnitt 2.1.2 vorgestellte Syntax der Merkmalsdiagramme zu realisieren. Der Ansatz ist aber auch auf andere mögliche Grafikelemente übertragbar.

Ohne an dieser Stelle näher auf den eigentlichen Inhalt des Diagramms ein-

gehen zu wollen, soll eine angestrebte Dreiteilung gegeben sein. In einem ersten Schritt soll zunächst der obere linke Quadrant (mit den Metadaten) erscheinen. Anschließen wird der obere rechte Quadrant (mit Angaben zum Layout) und in einem letzten Schritt die untere Diagrammhälfte (mit Angaben zum Inhalt) hinzugefügt. Diese Aufteilung ist relativ willkürlich gewählt und kann selbstverständlich in anderen Szenarien variieren und detaillierter oder gröber ausfallen.

Traditionell würde ein solches Diagramm z. B. zunächst vollständig als Grafik in OpenOffice.org Draw erstellt werden und anschließend durch Entfernen einzelner Elemente, dem Neuanordnen der übrigen Elemente und dem Speichern unter einem anderen Dateinamen erstellt werden. Dieses Vorgehen ist solange unproblematisch, bis ein neues Element hinzukommt oder sich ein vorhandenes in irgendeiner Form ändert.[1] Selbst kleinere Änderungen müssen so in sämtlichen Varianten von Hand übertragen werden.

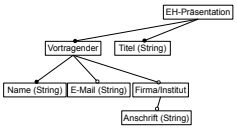
4.1.1 Featuremodell

Um das Generieren dieser Dokumente zu automatisieren, wird ein Featuremodell benötigt, welches die zuvor dargelegten Zielstellungen repräsentiert. Im einfachsten Fall, und dieser ist immer dann zu bevorzugen, wenn ausschließlich die oben erwähnten Veränderungen erwartet werden, könnte Abbildung 4.3a eine mögliche Lösung aufzeigen. Die Elemente des Ursprungsdiagramms werden dabei direkt mit dem gewünschten Ergebnis verknüpft. Somit kann durch die Auswahl des Merkmals „Abschnitt I“ in einer Variante die erste gewünschte Grafik erzeugt werden. Analog können mit der Auswahl des Merkmals „Abschnitt I+II“ bzw. des Merkmals „Alle Abschnitte“ die erwarteten nachfolgenden Grafiken generiert werden. Dieses Vorgehen führt zwar zügig zu einer zufriedenstellenden Lösung, weicht aber den eigentlichen Begriff eines Merkmals stark auf, da sich diese Merkmale nicht mehr direkt auf Eigenschaften des Quelldokuments, sondern auf das daraus resultierende Zieldokument beziehen. Dies wiederum wird aber zum Problem, wenn man weitere Varianten erstellen möchte, die über die zuvor beschriebenen hinaus gehen.[2]

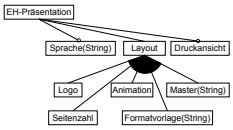
Abbildung 4.3b zeigt ein Featurediagramm, das sich näher am Quelldokument orientiert. Wohingegen das erste Modell nur vier Kombination abdeckt[3], umfasst das zweite Modell sämtliche acht Kombinationen und ist darüber hinaus noch flexibler zu erweitern, wenn weitere Merkmale zum Quelldokument hinzugefügt werden würden.

[1] Einfügen eines neuen Merkmals oder einer neuen Kante, Umbenennen eines Merkmals, Ändern der Kardinalitäten usw.
[2] Auf die detaillierte Betrachtung dieses Beispiels wird daher verzichtet.
[3] leer bzw. Kopfelement, Abschnitt I, Abschnitt I+II, Abschnitt I+II+III

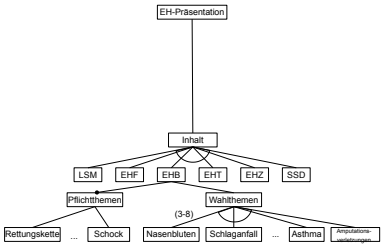
(a) Nur Kopfelement



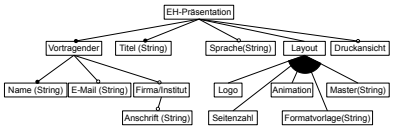
(b) Nur Metadaten



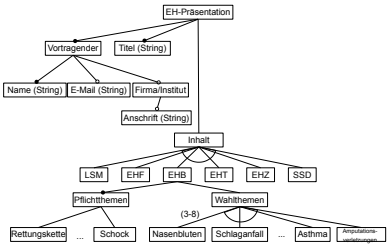
(c) Nur Layout



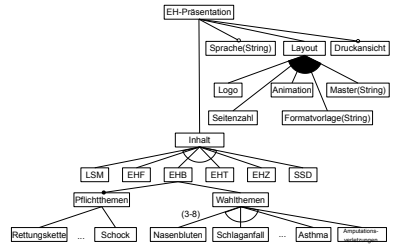
(d) Nur Inhalt



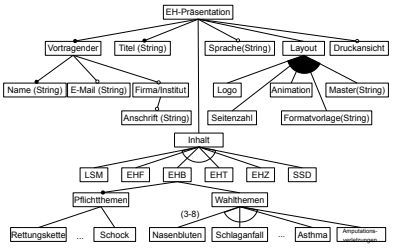
(e) Metadaten und Layout



(f) Metadaten und Inhalt

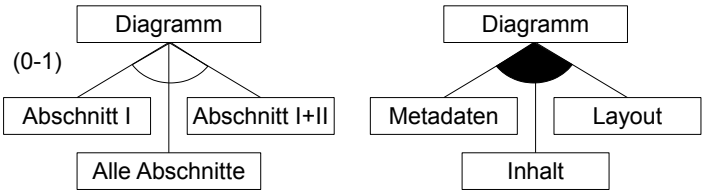


(g) Layout und Inhalt



(h) Komplett

Abbildung 4.2
Übersicht der einfachen Diagramme für Szenario 1



(a) Einfach (b) Besser

Abbildung 4.3
Beispiel Featurediagramme für Szenario 1

4.1.2 Abbildung

Das Featuremodell, alle möglichen Varianten und eine Grundzuordnung der Merkmale zu den Diagrammelementen sind mit Hilfe des *DocumentFeatureMappers* in wenigen Minuten umgesetzt. Abbildung 4.4 am Ende diese Abschnittes stellt nochmals den Inhalt des Editors nach Abschluss der Zuordnung der Merkmale zu Diagrammelementen in einer Übersicht dar. Trotz der zunächst ungewohnten Darstellung des Quelldokuments in einer Baumansicht ist in Kombination mit der OpenOffice.org-Ansicht ein zügiges und präzises Arbeiten möglich, und auch die mentale Verknüpfung der Baumelemente mit Diagrammelementen ist nach kurzer Eingewöhnung ohne weitere Schwierigkeiten gegeben. [4,5]

Mit der so erstellten Konfiguration ist es bereits möglich, Dokumente zu erstellen, die eine beliebige Kombination der drei Merkmale „Metadaten“, „Layout“ und „Inhalt“ enthalten (Abbildung 4.2). Unter der ursprünglichen Zielstellung, die Grafik in drei Schritten aufzubauen, wäre dies wieder eine zufriedenstellende Lösung.

4.1.3 Verfeinerung und Varianten

Unter der Annahme, dass der erste Schritt (Abbildung 4.5[6]) aber nicht Teil einer Präsentation ist, sondern in einer Arbeit abgedruckt werden soll, ist die Anordnung des Diagramms noch nicht optimal.

Über das Entfernen von Elementen hinaus ist es nun nötig, die weiteren Elemente zu modifizieren (vgl. Abschnitt 3.2.4) und deren Position im Dokument zu verändern, um eine ansprechende und kompakte Grafik zu erhalten. Da

[4] Das Beispiel liegt der Arbeit bei.
[5] Hinweis: Die Zuordnung der Rechtecke ist aufgrund der Angabe des enthaltenen Texts ohne weiteres möglich. Zusätzlich erhalten diese Einträge die interne Identifikation, was wiederum eine Zuordnung der einzelnen Verbindung ermöglicht. Die Großbuchstaben in Klammern geben dabei die Himmelsrichtung des Anschlusses an das Rechteck an. Die Rahmen und benutzerdefinierten Formen (Kreisausschnitte) lassen sich, sofern nicht durch den Text identifizierbar, über die nicht in der Abbildung 4.4 enthalten Attribute zur horizontalen und vertikalen Positionierung zuordnen.
[6] Vergrößerte Darstellung der Abbildung 4.2b

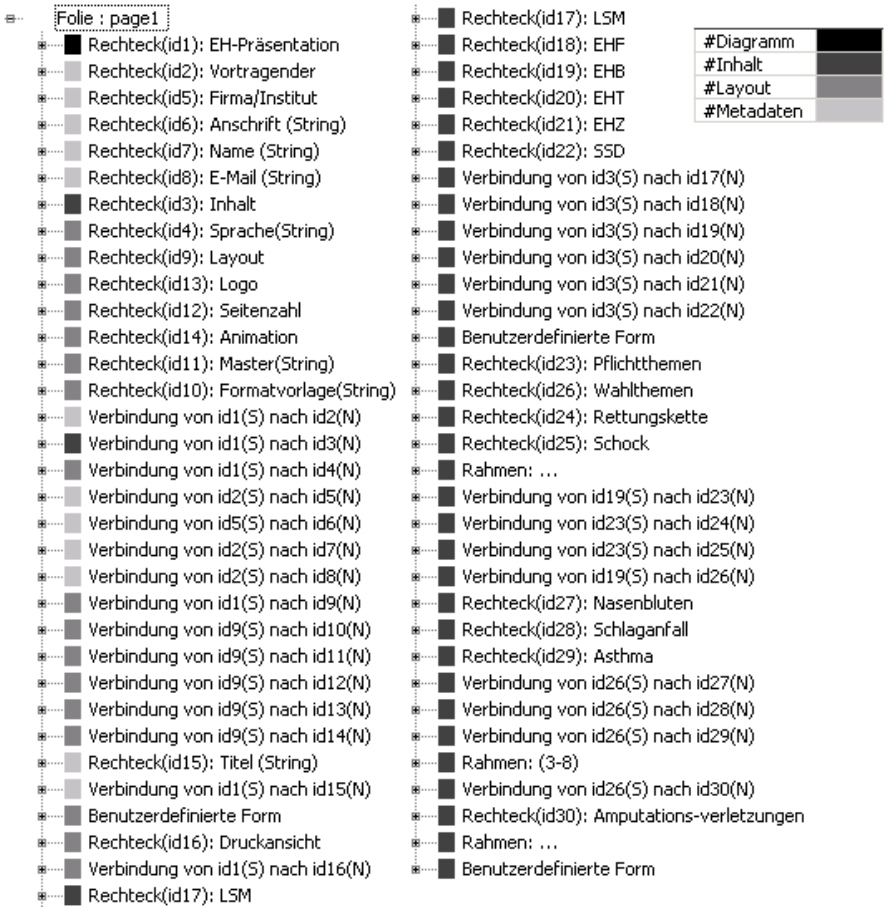


Abbildung 4.4
Merkmalsgrundzuordnung für Szenario 1

diese Modifikationen gewöhnlich nur in einigen Varianten Anwendung finden, können einzelne Merkmale des Featuremodells zu komplexen logischen Bedingungen verknüpft werden, um das Auslösen der Modifikation zu steuern. Da sich die weitere Betrachtung auf die Optimierung der vier ursprünglich benötigten Grafiken beschränkt, werden die Ausdrücke „Nur Diagramm“[7], „Nur Metadaten“[8] und „Metadaten und Layout“[9] benötigt (vgl. Abbildung 4.2 a/b/e).

Um die Grafik in Abbildung 4.5 ansprechender zu gestalten, gibt es wiederum mehrere Ansätze. Für einen besseren Überblick soll angenommen werden, dass kein Wert darauf gelegt wird, das A4-Format der ursprünglichen Grafik beizubehalten. Diese Annahme ermöglicht das Anpassen der Grafik mit nur drei Modifikationen. Wenn der Ausdruck „Nur Metadaten“ zu wahr ausgewertet wird, soll das Rechteck „EH-Präsentation“ nach links verschoben und weiterhin das A4-Format in Höhe und Breite beschnitten werden. Mit diesen Angaben

[7] Diagramm $\wedge \neg$ Metadaten $\wedge \neg$ Layout $\wedge \neg$ Inhalt
[8] Metadaten $\wedge \neg$ Layout $\wedge \neg$ Inhalt
[9] Metadaten $\wedge$ Layout $\wedge \neg$ Inhalt

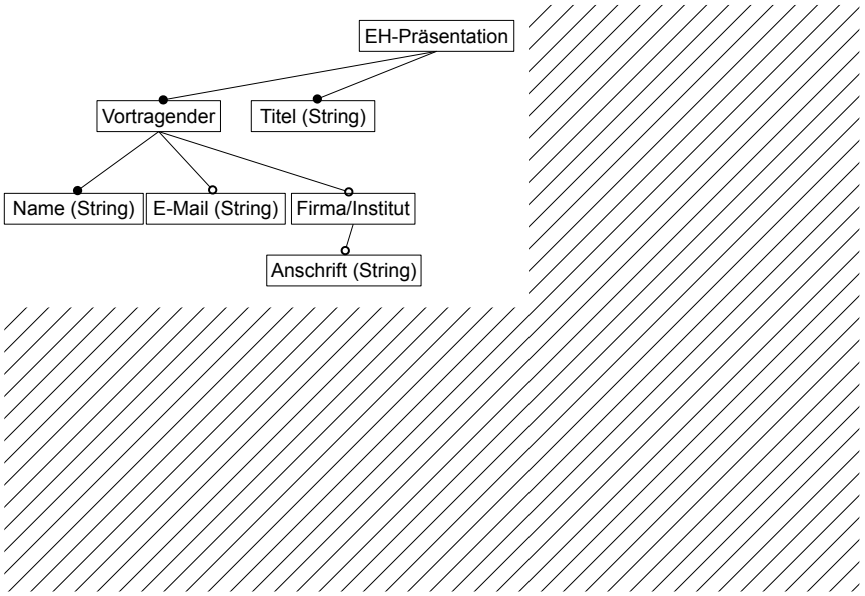


Abbildung 4.5
Einfache Version der Variante „nur Metadaten“ für Szenario 1

kann die Grafik in Abbildung 4.6 generiert werden.

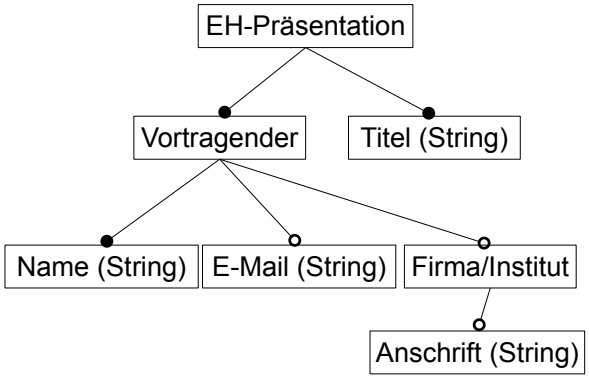


Abbildung 4.6
Opt. Version der Variante „nur Metadaten“ für Szenario 1

Ähnliche Überlegungen liegen auch der Übersicht sämtlicher weiterer Modifikationen (Tabelle 4.1) zu Grunde, die im Folgenden näher erläutert werden sollen.

Die Änderung der Position der einzelnen Elemente oder der Seitenabmessungen ist direkt über eine Auswahl der entsprechenden Eigenschaft in der PropertyView (vgl. Abschnitt 3.3.1) möglich. Bei der Anpassung der Werte ist nur darauf zu achten, dass Positionsangaben innerhalb von OpenOffice.org abzüglich des Randes angegeben sind und beim Übertragen um diesen erweitert werden

Eigenschaft	Ausdruck	Neuer Wert
Seitenbreite	Nur Diagramm	8 cm
	Nur Metadaten	16 cm
Seitenhöhe	Nur Diagramm	1.5 cm
	Nur Metadaten	5.5 cm
EH-Präsentation		
X-Pos	Nur Diagramm	1.5 cm
	Nur Metadaten	5.5 cm
Y-Pos	Nur Diagramm	1 cm
Schwarzer Kreisausschnitt		
Y-Pos	Metadaten und Layout	7.8 cm
Bogengrenzen[a]	Metadaten und Layout	'153,72.. 35,11..'
Rechtecke mit[b]		
Y-Pos=4.1 cm	Metadaten und Layout	8 cm
Y-Pos=7.1 cm	Metadaten und Layout	12 cm
Y-Pos=9.1 cm	Metadaten und Layout	15 cm

Tabelle 4.1
Modifikationen für Szenario 1

[a] Derzeit nicht in der Benutzeroberfläche enthalten
[b] Einschränkungen in Verwendung, siehe Fließtext

müssen.[10]

Formal ist es weiterhin möglich, Elemente des Quelldokuments nicht nur über eine eindeutige Identifikation zu adressieren, sondern auch eine Gruppe von Elementen über beliebige Eigenschaften auszuwählen. Dieses Vorgehen wird im aktuellen Prototyp durch eine Expertenansicht (vgl. Abschnitt 3.2.5) unterstützt, setzt aber ein erhöhtes Verständnis des Open Document Formats voraus, um die Identifikatoren manuell zu ermitteln. Die Auswahl sämtlicher Rechtecke, die auf einer Höhe von 4,1 cm beginnen (Vortragender, Titel, Sprache, Layout und Druckansicht), ist z. B. mit dem Identifikator in Listing 4.1 zu erreichen.

Listing 4.1
Identifikator für die zweite Reihe der Merkmale in Szenario 1

```
...odg!/content.xml#//draw:rect[@svg:y="4.1cm"]/@svg:y
```

Der Abschnitt vor der Raute beschreibt zunächst wieder die Quelldatei. Der hintere Teil in Gestalt eines XPath-Ausdruckes wählt zunächst sämtliche Rechtecke aus, deren Y-Koordinate exakt „4.1cm“ ist. Von diesen soll dann genau diese Y-Koordinate geändert werden. Analog können die anderen beiden oder weitere komplexere Ausdrücke hergeleitet werden.

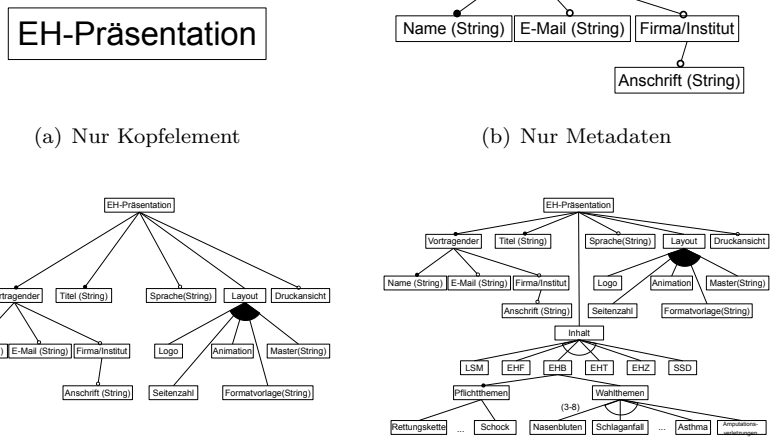
Diese Vorgehensweise ist aber mit Vorsicht einzusetzen und es gilt, sich vor der Anwendung über die Konsequenzen klar zu werden. Die Adressierung über die horizontale Koordinate hat zunächst den Vorteil, dass die Modifikation nicht allen Rechtecken einzeln zugeordnet werden muss. Selbst wenn im Quelldoku-

[10] In diesem Beispiel ist der Rand 1 cm breit. Wenn die generierten Dokumente in OpenOffice.org geöffnet werden, sind die Positionsangaben somit jeweils um 1 cm reduziert.

ment zukünftig noch weitere Rechtecke auf derselben Höhe eingefügt werden und diese in der Variante vorhanden sind, werden diese ebenfalls mit verschoben, ohne dass das Mapping nochmals angepasst werden muss. Nachteile ergeben sich, wenn dieses Verhalten nicht erwünscht ist, oder wenn die Möglichkeit besteht, dass sich im Quelldokument diese Höhe „jemals“ ändert und der Identifikator damit ins Leere läuft. Unter diesem Aspekt wäre dann eine stabilere Identifikation über die Einzelelemente zu bevorzugen, oder das Mapping muss ebenfalls angepasst werden.

Das Generieren der Zieldokumente mit der zuvor hergeleiteten Abbildung würde trotzdem zu einem unansehnlichen Ergebnis führen, da das Open Document Format zwar den Verbindungen ein Start- und Zielelement zuordnet, aber parallel dazu auch die exakte Position der Ansatzpunkte speichert. Somit wären die Verbindungen noch an ihrer ursprünglichen Position. Die Ansatzpunkte werden erst bei der Veränderung innerhalb von OpenOffice.org neu berechnet. Damit diese nun weder im Mapping angegeben, noch durch das Laden in OpenOffice.org repariert werden müssen, stellt der *DocumentFeatureMapper* ein Nachbearbeitungswerkzeug zur Verfügung, das automatisch nach dem Generieren ausgeführt werden kann und die Verbindungen neu berechnet (vgl. Abschnitt 3.6.5).

Abbildung 4.7 zeigt die generierten, im Rahmen der Zielstellung interessanten und optimierten Dokumente.



(a) Nur Kopfelement

(b) Nur Metadaten

(c) Metadaten und Layout

(d) Komplett

Abbildung 4.7
Übersicht der optimierten Diagramme für Szenario 1

4.2 Calc - Einfacher Projektplan

Aufbauend auf den Erkenntnissen des vorangegangen Anwendungsfalles bildet nun ein Tabellendokument die Grundlage für die weiteren Betrachtungen. Hierbei liegt ein besonderer Schwerpunkt auf den Besonderheiten, die der Umgang mit Tabellen mit sich bringt. Als Ausgangspunkt dient ein einfacher Projekt- bzw. Zeitplan (Abbildung 4.8) wie er zur Planung dieser Arbeit verwendet wurde.

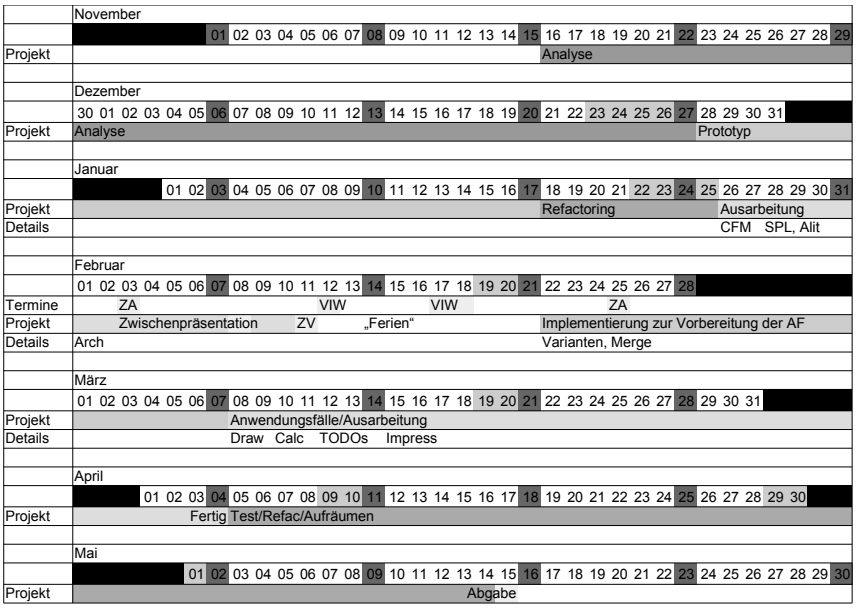


Abbildung 4.8
Einfacher Projektplan

4.2.1 Featuremodell

Um ein geeignetes Featuremodell für die zu erzeugende Dokumentfamilie abzuleiten, gilt es zunächst wieder, einen Überblick über mögliche Zieldokumente zu bekommen. Abbildung 4.8 ist bereits selbst die erste Variante des Quelldokuments, welches in seiner ursprünglichen Fassung nicht in Graustufen sondern mehrfarbig erstellt wurde. Um die Anpassung der Farben nicht dem Drucker zu überlassen und damit unleserliche Abschnitte zu riskieren, wird ein Merkmal benötigt, das die Farbigkeit repräsentiert. Ebenfalls der ursprünglichen, für die Bildschirmausgabe optimierten Farbgebung, sind die schwarzen Balken zu Beginn und Ende eines Monats geschuldet. Da diese nicht bloß beim Druck in Graustufen, sondern auch beim farbigen Druck unangenehm sind, soll dies über ein eigenes Merkmal abgedeckt werden.

In der Tabelle in dieser Form sind auch Informationen enthalten, die zwar für die Erstellung des Plans nötig und hilfreich waren, aber eigentlich nicht in

einer öffentlichen Version enthalten sein sollten. So enthält die Zeile „Termine“ einige private Termine, die das Projekt beeinflussen können, aber nicht zuletzt aufgrund der Abkürzungen für die meisten Leser von geringem Interesse sind. Selbiges gilt auch für die hellgrau unterlegten Wochentage, die kennzeichnen, dass man zu dieser Zeit nicht in der Stadt ist bzw. war. Dieser zunächst simple Grundgedanke lässt sich aber bei anderen Dokumenten auch auf interne oder sogar geheime Informationen erweitern. Das Merkmal soll kurz als „Auswärts“ bezeichnet werden, hat aber in der Anwendung die Bedeutung von „Auswärts verstecken“.

Aber auch die direkt projektbezogenen Informationen lassen sich anhand einer möglichen Zielgruppe in zwei weitere Merkmale aufspalten. Soll ein solcher Projektplan einem Kunden präsentiert werden, würde man vermutlich auf die Zeilen beginnend mit „Details“ eher verzichten wollen und nur die Zeilen beginnend mit „Projekt“ zeigen, um das Dokument überschaubarer zu gestalten.

Sobald ein Plan über die hier angegeben sechs Monate hinaus geht, kommt weiterhin der Wunsch auf, nur Teile davon weiterzugeben. Für eine einfachere Umsetzung zerfällt diese Anforderung in zwei Merkmale, eines, das alle Monate anzeigt und ein weiteres, das basierend auf einer Liste, die Auswahl von Monaten erlaubt.

Abbildung 4.9 fasst diese Erkenntnisse nochmal in einem Featurediagramm zusammen.

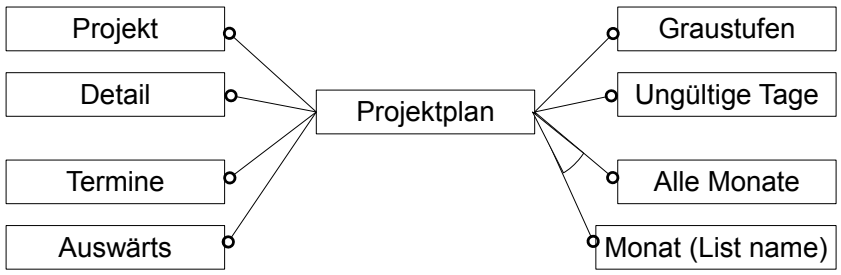


Abbildung 4.9
Featurediagramm für Szenario 2

4.2.2 Abbildung

Damit die eigentliche Zuordnung der Merkmale zu Dokumentelementen vereinfacht wird, ist es hilfreich, aber nicht zwingend nötig, wenn das Dokument sauber unter Nutzung von Formatvorlagen erstellt wurde. Ist dies nicht gewährleistet und kann das Originaldokument verändert werden, gilt es zu prüfen, ob es sich lohnt, diese nachträglich in ein Dokument einzuführen. Durch die Identifikation über Formatvorlagen sind zukünftige Erweiterungen des Originaldokuments mit

signifikant geringerem Aufwand möglich.

Sind keine eigenen Formatvorlagen vorhanden, muss versucht werden, mit Hilfe des Editors die automatische Formatvorlage zu identifizieren, die das Erscheinungsbild eines Dokumentelements beschreibt[11]. Auf diese Formatvorlage kann dann über einen Namen aus Kleinbuchstaben gefolgt von Zahlen zugegriffen und die Eigenschaft verändert werden. Neben dem Aufwand, diese manuelle Identifikation im pessimistischsten Fall für jede Zelle[12] einzeln machen zu müssen, kann jede Änderung am Layout des Originaldokuments die Namen der Formatvorlagen verändern und das Mapping so unbrauchbar machen.

Farbigkeit

Da das Beispieldokument eigene Formatvorlagen, enthält soll dieser Weg nicht weiter verfolgt werden. Die benutzerdefinierten Formatvorlagen können einfach über den Editor ausgewählt und wie gewohnt um Modifikationen erweitert werden. Tabelle 4.2 zeigt die Modifikationen, die nötig sind, alle Formatvorlagen auf Graustufen umzustellen bzw. die schwarzen Felder transparent zu machen.

Eigenschaft	Ausdruck	Neuer Wert[a]
Hintergrundfarbe der Zelle in Formatvorlage		
Analyse	Graustufen	#999999
Ausarbeitung	Graustufen	#DDDDDD
Implementierung	Graustufen	#CCCCCC
QS	Graustufen	#AAAAAA
Wichtig	Graustufen	#EEEEEE
Auswärts	Graustufen	#CCCCCC
Invalid	Ungültige Tage	transparent

Tabelle 4.2
Modifikationen am Layout für Szenario 2

[a]Hexadezimaler RGB-Farbcode wie er auch in HTML Anwendung findet

Inhalte

Für die Abbildung der Merkmale „Projekt“, „Details“ und „Termine“ stehen wieder zwei alternative Ansätze zur Verfügung. Es ist zunächst möglich, die Tabellenzeilen, die entfernt werden sollen, im Editor direkt zu selektieren und diese mit den Merkmalen zu verknüpfen. Da die Zeilen dann nur über ihre Zeilennummer identifiziert werden können, treten hier wieder Probleme auf, wenn das Quelldokument verändert wird. Fügt man dem Dokument hingegen nur am Ende neue Zeilen hinzu, muss man wiederum das Mapping erweitern.

[11]Selbstverständlich kann eine Tabellenzelle auch über ihre Position identifiziert werden, aber Formatvorlagen fassen gewöhnlich gleichartige Zellen zusammen und reduzieren so den Umfang des Mappings.

[12]Jeder andere Rand, jede andere Farbe, jeder andere Datentyp usw. führt zu einer eigenen automatischen Formatvorlage

Die elegantere Möglichkeit ist die Spezifikation eines eigenen Identifikators (Listing 4.2) mit Hilfe der Expertenansicht.

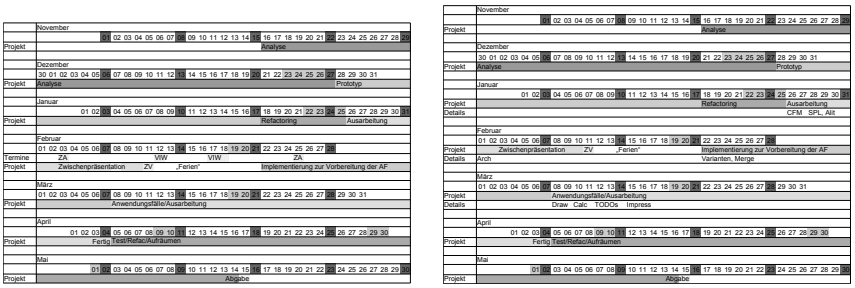
```
....ods!/content.xml#//table:table-row/
  table:table-cell[1][text:p="Projekt"]/..
```

Listing 4.2
Identifikator für die Tabellenzeile beginnend mit „Projekt“

Dieser erkennt auch neu hinzugefügte Zeilen und ist unanfällig gegen das Einfügen von Zeilen. Der erste Teil vor der Raute definiert wieder das Dokument. Im zweiten Teil werden die Zeilen ausgewählt, deren erste Zelle einen Absatz mit dem Inhalt „Projekt“ enthält. Analog können die Identifikatoren für „Details“ und „Termine“ erstellt werden

Generiert man nun eine Variante, z. B. ohne die „Details“, führt dies zum Entfernen einzelner Tabellenzeilen. Damit einher geht eine neue fortlaufende Nummerierung der verbliebenen Tabellenzeilen. Finden in einem solchen Tabellendokument dann Formeln Anwendung[13], führt dieses Verhalten dazu, dass sich die Referenzen auf falsche Zellen beziehen und falsche oder ungültige Berechnungen durchgeführt werden. Ähnlich der Reparatur der Verbindungen im vorangegangen Beispiel bietet der *DocumentFeatureMapper* einen geeigneten Nachbearbeitungsschritt (vgl. Abschnitt 3.6.6) an, um dies automatisch korrigieren zu können.

Abbildung 4.10 zeigt zwei generierte Varianten in Graustufen, ohne die schwarzen Balken, über den kompletten Zeitraum und ohne Details bzw. Termine.



(a) Ohne Details

(b) Ohne Termine

Abbildung 4.10
Übersicht der Tabellen ohne „Details“ oder „Termine“ für Szenario 2

Merkmal „Auswärts“

Bei der Umsetzung des Merkmals „Auswärts“ gibt es drei alternative Ansätze, die ein unterschiedliches Verhältnis von Stabilität bei Erweiterungen und

[13] In diesem Beispiel das Berechnen des Datums und die bedingte Formatierung der Sonntage

Sicherheit bzw. Rekonstruierbarkeit der versteckten bzw. entfernten Informationen aufweisen.

Wird einer Zelle eine benutzerdefinierte Formatvorlage zugewiesen und anschließend, wie in diesem Beispiel, die ganze Tabellenzeile mit einem Rand versehen, wird innerhalb von OpenOffice.org eine neue automatische Formatvorlage angelegt. Die Formatvorlage enthält einen Verweis auf die übergeordnete Formatvorlage und die neuen Informationen bezüglich des Aussehens des Randes. Dies bedeutet konkret, dass die Zelle keinen Verweis auf die Formatvorlage „Auswärts“, sondern auf eine automatische Formatvorlage z. B. „ce123“ enthält. Hier kann es also wieder passieren, dass eine Veränderung des Quelldokuments zur Vergabe von neuen automatischen Bezeichnern führt und dass ein Mapping, welches sich darauf bezieht zunächst nicht mehr gültig ist.

Der erste sichere, aber instabile Ansatz wäre somit die Ermittlung der automatischen Formatvorlagen der entsprechenden gekennzeichneten Zellen und analog dazu die Ermittlung der Formatvorlage von Zellen ohne diese Information. Mit Hilfe von Modifikationen können dann die zu entfernenden Formatvorlagen durch die anderen ersetzt werden. Dies hat zunächst den entscheidenden Vorteil, dass selbst bei der Inspektion der internen XML-Dokumente keine Rückschlüsse auf das Originaldokument möglich sind.

Dem gegenüber steht ein Vorgehen, das Anwendung finden kann, wenn das Dokument ausschließlich in gedruckter Form verbreitet wird und ein Zugriff des Empfängers auf die interne Dokumentstruktur nicht möglich ist. Dann genügt es natürlich, die Information ähnlich dem Merkmal „Graustufen“ einfach direkt aus der Formatvorlage zu entfernen. Das heißt, die Hintergrundfarbe der Formatvorlage wird auf transparent gesetzt und die „Auswärts“-Information ist nicht mehr sichtbar. Somit deckt diese Möglichkeit auch zukünftige Änderungen und neue Tabellenzellen mit der Formatvorlage „Auswärts“ ab.

Für einen möglichst flexiblen Einsatz weisen beide Ansätze einige Unzulänglichkeiten auf. Im ersten Fall macht potenziell jede Erweiterung das Mapping zunächst unbrauchbar und die zweite Lösung ist ausschließlich für gedruckte Dokumente anwendbar. Um diesem zu begegnen, soll noch eine dritte Möglichkeit näher betrachtet werden.

Wie bereits erwähnt, erhält die erzeugte automatische Formatvorlage eine Referenz auf die übergeordnete benutzerdefinierte Formatvorlage. Diese kann mit einem geeigneten Identifikator (Listing 4.3) innerhalb einer Modifikation ausgetauscht werden.[14]

Durch das Austauschen der übergeordneten Formatvorlage beim Generieren einer entsprechenden Variante existieren dann zwei automatische Formatvor-

[14] Hier: neuer Wert „Default“

```
....ods!/content.xml#//style:style
  [@style:parent-style-name="Auswärts"]/
  @style:parent-style-name
```

Listing 4.3 Identifikator für automatische Formatvorlagen, die „Auswärts“ erweitern

lagen mit demselben Erscheinungsbild[15] und unterschiedlichen Namen. Ist das Originaldokument zuvor sauber unter Nutzung von Formatvorlagen erstellt worden, fallen zwei fast identische Formatvorlagen in der internen Struktur aber auf, lassen Rückschlüsse auf den Inhalt des Originaldokuments zu und bieten so noch nicht die Sicherheit aus dem ersten Ansatz. Um diese Sicherheit herzustellen, existiert derzeit nur die Möglichkeit, das generierte Dokument in OpenOffice.org zu öffnen und anschließend zu speichern. Dadurch werden die doppelten Formatvorlagen entfernt und die ursprüngliche Markierung ist nicht mehr ermittelbar. In einer nächsten Version des *DocumentFeatureMappers* wird dies aber auch direkt über einen eigenen Nachbearbeitungsschritt möglich sein.

Zeitraum

Für die Auswahl der Monate ist der Aufwand ein wenig umfangreicher. Der aktuelle Prototyp unterstützt keine Identifikation von Elementen über Attribute der Variantenfeatures (vgl. Abschnitt 3.2.6). Weiterhin besteht ein Monat aus einer unterschiedlichen Anzahl an Zeilen in der Tabelle. Somit ist ein wünschenswerter Identifikator der Art

$\forall x \in$ `Variante.Monat.name Wähle die Tabellenzeilen` i `bis` j `, deren erste Zeile` i `in der zweiten Zelle einen Absatz mit dem Text` x `enthält und deren letzte Zeile` j `vollständig leer ist.` [16]

derzeit leider nicht möglich.

Für jeden Monat muss ein eigener Ausdruck der Art `Alle Monate` $\vee$ `Monat (`x`)` angelegt werden, und diesem können dann über die grafische Oberfläche den Tabellenzeilen zugeordnet werden. Abbildung 4.11 zeigt die ursprünglichen Merkmale und die neu hinzugekommen Ausdrücke. Man erkennt, dass die ersten vier Tabellenzeilen dem Ausdruck „November“ zugeordnet sind. Dieser Ausdruck wiederum verknüpft die Merkmale `#Alle Monate` und `#Monat (0911)` disjunktiv. [17]

[15] Hier: die normalen Tage und die Tage, die markiert waren

[16] Das Attribut „name“ des Merkmals „Monat“ in einer Variante ist hier als Liste von Monaten definiert.

[17] November, Dezember usw. sind selbst keine Merkmale, sondern Bezeichner für komplexe Ausdrücke, die wie ein aktiviertes Merkmal behandelt werden, wenn der Ausdruck zu wahr ausgewertet werden kann, also z. B. „0911“ in der Liste der anzuzeigenden Monate enthalten

Abbildung 4.11
Merkmalszuordnung für Szenario 2

4.2.3 Variante

Nachdem sämtliche Merkmale zugeordnet wurden, können Zieldokumente mit beliebigen Merkmalskombinationen generiert werden. In Abbildung 4.12 ist eine Variante in Graustufen, ohne die schwarzen Balken, mit den entfernten Angaben zur Abwesenheit und beschränkt auf die Monate November, Dezember und Mai abgebildet. Das Beispiel liegt der Arbeit bei, so dass weitere Kombinationen individuell ausprobiert werden können. Wichtig ist dabei, den Nachbearbeitungsschritt zur Reparatur der Formeln nicht zu vergessen, da anderenfalls die Tabellenzeilen mit den einzelnen Tagen, aber auch die bedingte Formatierung der Sonntage zerstört werden.

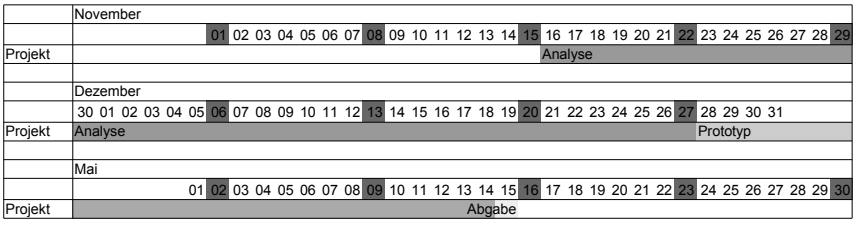


Abbildung 4.12
Projektplan der Monate November, Dezember und Mai für Szenario 2

ist.

4.3 Impress - ASB-Erste-Hilfe-Ausbildung

Die Basis für die weiteren Betrachtungen bilden die Präsentationen der Erste-Hilfe-Ausbildung des Arbeiter-Samariter-Bundes[18]. Nach einem kurzen Einblick über die Hintergründe und Rahmenbedingungen der Präsentationen soll wieder ein geeignetes Featuremodell aufgezeigt werden, um abschließend drei konkrete Varianten umzusetzen. Dieses Anwendungsszenario steigert die Komplexität im Vergleich zu den vorangegangen Beispielen deutlich erhöht.

4.3.1 Quelldateien und Motivation

Sämtliche Lehrmaterialien der Erste-Hilfe-Ausbildung des ASBs werden zentral durch die Bundesgeschäftsstelle in Köln allen Ausbildern in den Regional- und Kreisverbänden zur Verfügung gestellt. Diese Menge kann aber für die folgenden Betrachtung auf eine Auswahl von 20 Präsentationsdokumenten reduziert werden. Den Kern bilden dabei acht Präsentationen, die jeweils eine Doppelstunde eines Kurses Lebensrettende Sofortmaßnahmen (Nachweis für PKW-Führerschein) bzw. Erste-Hilfe (Nachweis für LKW-Führerschein) abdecken. Hinzu kommen zehn weitere Dokumente, die jeweils einige wenige Folien enthalten, um Ergänzungen oder Vertiefungen anzubieten. Ein weiterer Foliensatz enthält Informationen über das Tätigkeitsfeld des ASBs, der bei der Vorstellung und Einleitung gezeigt werden kann. Ebenfalls für die Einleitung oder aber für den Abschluss geeignet steht ein letzter Foliensatz mit Hinweisen zu weiteren Terminen und anderen Kursen zur Verfügung.

Auf den hier kurz umrissenen Folien basieren auch einige weitere Kurse, die im zeitlichen Umfang oder den inhaltlichen Schwerpunkten variieren. Diesem Sachverhalt wurde bisher beim Erstellen durch den ASB mit einer Vielzahl an einzelnen Präsentationen Rechnung getragen, die eine Kombination der Inhalte erlauben soll. Dieser Ansatz weist zunächst den rein optischen Mangel auf, dass ein wenig technisch versierter Ausbilder während seines Vortrages immer zwischen Präsentationen hin und her springen bzw. neue Präsentationen öffnen muss. Sobald ein Ausbilder aber in der Lage ist, diese Präsentationen auch zu bearbeiten, d. h. die Zusatzthemen direkt zwischen die anderen Folien einzufügen oder vier Doppelstunden- zu einer Tagespräsentation zusammenzufassen, führt dies unweigerlich zu einer Vielzahl an Varianten der Originaldokumente. Hinzu kommen noch regionale Anpassungen oder z. B. im Rahmen der Erste-Hilfe-Ausbildung in Betrieben bei jedem Betrieb eigene den Gefahrensituationen angepasste Schwerpunkte und Zusatzthemen. Im Verlauf dieses Abschnittes werden noch weitere Varianten aufgezeigt, aber jede neue Variante, die nach der

[18] http://www.asb.de

Methode „Kopieren und Anpassen“ durch einen Ausbilder erstellt wird, führt zu dem Hemmnis, ein mögliches zentrales Update der Quelldateien auf alle seine Varianten zu übertragen. Im pessimistischsten Fall leidet darunter die Qualität der Ausbildung und führt bei den Kursteilnehmern zu Verwirrungen, wenn der Ausbilder zwar auf dem aktuellen Stand ist, seine unterstützende Präsentation aber einige Jahre alt und er genötigt ist, jede Differenz verbal zu kommunizieren.

Exemplarisch zu den 20 zentralen Foliensätzen komme noch fünf weitere Dateien, die Inhalte aus anderen ASB-Präsentationen enthalten oder eine Titelseite sowie Informationen über den Ausbilder bzw. den Vortragenden bereit stellen. Die Letzteren, aber auch die Termine und Hinweise auf andere Kurse, unterliegen dabei noch einer deutlich höheren Veränderlichkeit als die eigentlichen Kursinhalte.

4.3.2 Featuremodell

Dem Featuremodell, welches im folgenden erarbeitet werden soll, liegt der Wunsch zugrunde, für jeden Kurstermin einen separaten Ordner mit allen benötigten Materialien, und nur diesen, generieren zu können. Ein solcher Ordner, oder besser ausschließlich die Variantenbeschreibung, kann dann archiviert werden, erfordert aber selbst keine Veränderungen mehr.

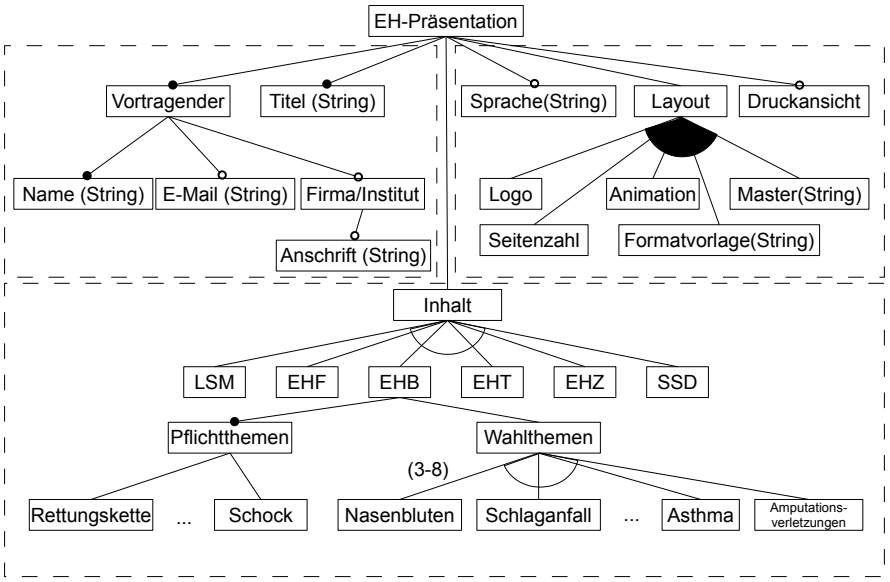
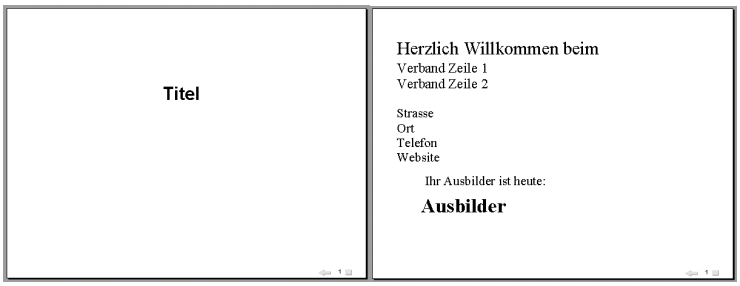


Abbildung 4.13
Mögliches Merkmalsdiagramm

Beim Entwurf eines Featuremodells gibt es nicht *die* richtige Lösung. Inwieweit ein Modell geeignet ist, hängt wesentlich vom Einsatzgebiet ab. So beschreibt der erste Entwurf des Merkmalsdiagramms (Abbildung 4.13), das auch

als Beispiel in Abschnitt 4.1 diente, zwar das Szenario korrekt, erhöht aber durch die duplizierten Themenmerkmale den Aufwand, ein Mapping zu erstellen. Andererseits legt man keinen Wert darauf, nur ein Featuremodell und damit jeweils eine vollständige Variantenbeschreibung zu erstellen, könnte man die 25 Dateien in mehrere unabhängige Teile zerlegen und damit auch die nötigen Featuremodelle vereinfachen.



(a) Titel (b) Vorstellung

Abbildung 4.14
Titel- und Vorstellungsfolie aus Szenario 3

Abbildung 4.14 zeigt die zwei zusätzlichen Folien für den Titel und die Vorstellung des Vortragenden[19]. Aus den veränderlichen Bereichen resultieren die Merkmale Vorstellung, Name, Verband, Anschrift und Titel, wie in Abbildung 4.15 mit entsprechenden Attributen dargestellt.

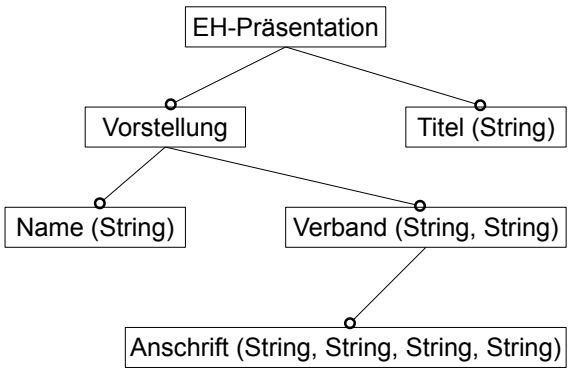
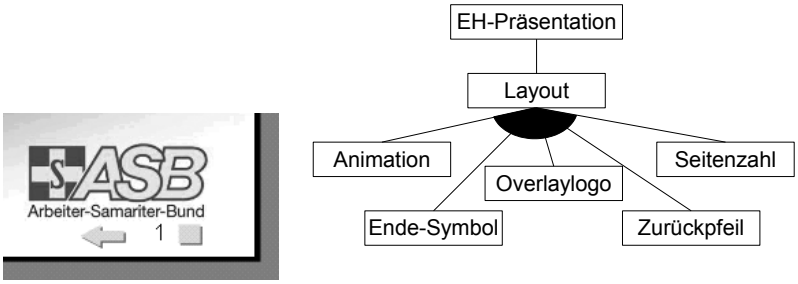


Abbildung 4.15
Merkmale der Titel- und Vorstellungsfolie für Szenario 3

In Abbildung 4.16a ist der untere rechte Ausschnitt einer jeden Folie zu sehen. Die Seitenzahl, der Zurückpfeil und das Logo[20] werden durch den Folienmaster beigesteuert. Bei folienfüllenden Grafiken wird das verdeckte Logo durch ein optisch identisches Overlaylogo innerhalb der Folie wieder in den Vordergrund gebracht. Abschließend markiert das Rechteck das Ende aller Ani-

[19] Das Hintergrundbild wurde für den Druck entfernt
[20] Dies soll generell nicht entfernbar sein.



(a) Folie (b) Diagramm

Abbildung 4.16
Merkmale der Gruppe Layout für Szenario 3

mationen auf einer Folie und signalisiert, dass der nächste Tastendruck oder Mausklick zur nächsten Folien führt. All diese Elemente sollen in beliebiger Kombination und nach den Vorlieben des Vortragenden ausgeblendet werden können. Das Merkmalsdiagramm in 4.16b fasst die Merkmale nochmals zusammen und bietet auch die Möglichkeit, Animationen völlig zu deaktivieren.

Der nächste Komplex an Merkmalen fasst die Kurse und sämtliche Themen zusammen. Die drei bereits erwähnten Kursen sollen hier exemplarisch um das Erste-Hilfe-Training (EHT), Erste-Hilfe-Zweiradfahrer (EHZ), Erste-Hilfe-Kindernotfälle (EHK) und den Schulsanitätsdienst (SSD) erweitert werden. Dem gegenüber stehen sämtliche Themen, wie in Abbildung 4.17 dargestellt. Die Angabe in geschweiften Klammern gibt einen Hinweis auf die reale Anzahl der untergeordneten Merkmale, auf deren Angabe zu Gunsten der Übersicht verzichtet wurde. Die Bezeichnungen der Themen-Merkmale resultieren aus der ASB-internen Einteilung der Präsentationen. Die Unterscheidung in Zusatzthemen und eigene Themen ist im weitesten Sinn willkürlich. Auch eine weitere Unterteilung der Kernthemen wäre möglich. Es wurde in der Grafik ebenfalls darauf verzichtet, die Abhängigkeiten zwischen den Kursen und den Themen darzustellen, diese können bei Interesse aus den dieser Arbeit beigefügten Beispieldaten entnommen werden. Es besteht z. B. eine Abhängigkeit zwischen dem Kurs LSM und dem Thema Notruf, d. h. eine Variante mit dem Merkmal LSM muss auch das Merkmal Notruf enthalten.

Weitere Merkmale ergeben sich für den Foliensatz, der die Hinweise auf nächste oder andere Kurse enthält. Abbildung 4.18a zeigt die darin enthaltenen Folie für den Kurs LSM. Ordnet man die variablen Bereiche[21] wieder Attributen zu, so erhält man das Diagramm in 4.18b. Auch hier wurde für die Übersichtlichkeit auf die Angabe der anderen 13 möglichen Kurse verzichtet.

Den Abschluss bilden drei weitere Merkmale, die wieder den unterschiedlichen Vorlieben der einzelnen Ausbilder Rechnung tragen sollen. Zunächst finden

[21] Uhrzeit, Termine, Kosten, Ort und URL

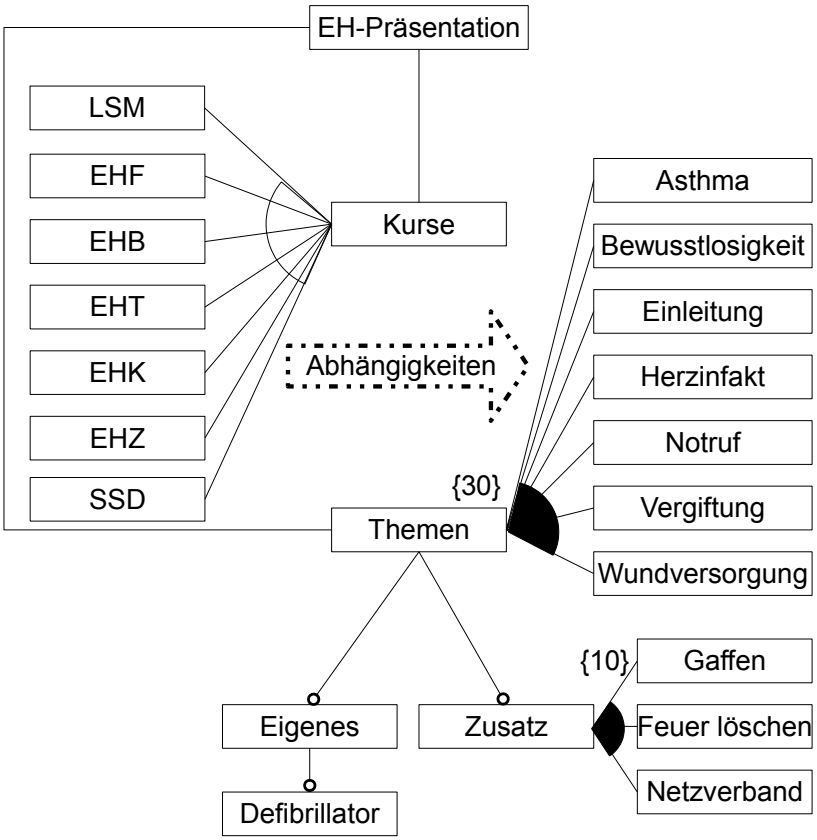
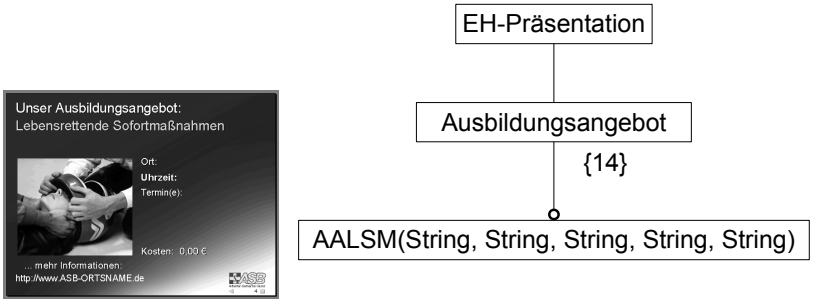


Abbildung 4.17
Merkmale der Kurse und Themen für Szenario 3

sich über alle Foliensätze hinweg zwischen den Themen vereinzelt Folien, die als Trennung dienen sollen, aber nicht immer auch eine Pause erfordern und so meist schnell weitergeklickt werden. Darüber hinaus enthalten die Präsentationen an geeigneten Stellen Querverweise auf die Tätigkeiten des ASBs. Diese Unterbrechungen kommen aber wiederum nicht allen Ausbildern entgegen, so dass der gewünscht positive PR-Effekt nicht eintritt oder sich gar in den negativen Eindruck einer „Werbeveranstaltung“ umkehren kann. Dem Vortragenden die Wahl zwischen den eingebetteten Hinweisen oder dem alternativen, separaten und zusammenhängen PR-Foliensatz zu geben, kann diesem Problem begegnen. Sollen in einer Variante mehrere Foliensätze zu einem verschmolzen werden, erweisen sich die jeweils erste und letzte Folie der Kernpräsentationen als überflüssig, diese seien unter dem Merkmal „Randfolien“ zusammenfasst.

Abbildung 4.19 stellt diese Merkmale nochmals grafisch dar und ergänzt noch zwei Merkmale, die Veränderungen im Vergleich zum Originaldokument



(a) Folie LSM (b) Diagramm

Abbildung 4.18
Merkmale der Ausbildungsangebote für Szenario 3

zulassen, ohne diese am Original vornehmen zu müssen. Dies ist insbesondere dazu gedacht, mögliche Mängel oder Erweiterungen bis zur nächsten Version des zentralen Foliensatzes zu überbrücken. Führt man diesen Gedanken weiter, können die auf diese Weise eingefügten Änderungen aber auch als konkreter Hinweis für die Entwickler des Foliensatzes dienen, so dass keine Änderungen in den Dateien ermittelt oder unpräzise verbale Beschreibungen interpretiert werden müssen.

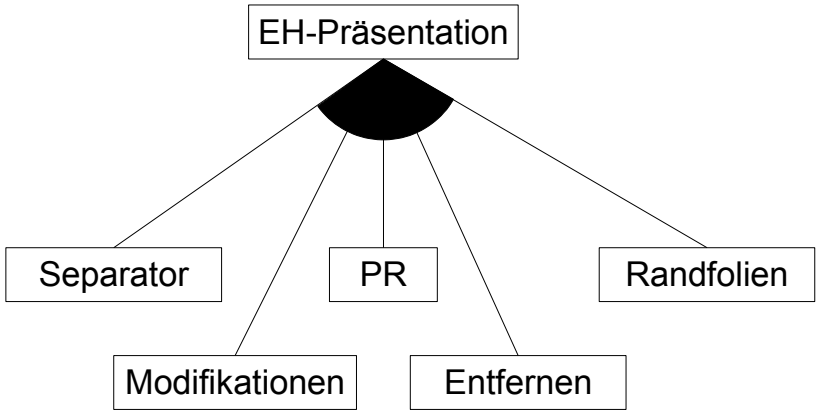


Abbildung 4.19
Weitere Merkmale für Szenario 3

Selbstverständlich sind weitere Merkmale basierend auf den Vorlieben der Vortragenden bzw. den Anforderungen der Zielgruppe[22] möglich. Auch eine feingranularere Einteilung der Themen bietet sich an. Im Rahmen dieser Arbeit soll aber das vorgestellte Modell und die damit einhergehende Komplexität genügen.

[22] Kinder, Ältere Menschen oder Menschen mit Beeinträchtigungen

4.3.3 Abbildung

Im Gegensatz zu den beiden vorangegangen Beispielen sind die Foliensätze der Erste-Hilfe-Ausbildung unabhängig von und weit vor dieser Arbeit entstanden. Auch wenn in einer späteren Anwendung die Entwickler der Originalpräsentationen eine Optimierung für eine einfachere und stabilere Abbildung anstreben können, sollen diese zunächst unverändert bleiben. Die größten Unannehmlichkeiten diesbezüglich folgen derzeit aus einer nicht durchgängigen Vergabe von Folientiteln innerhalb der Präsentationen, womit eine Identifikation meist nur über die fortlaufende Foliennummer, mit den bereits erläuterten Problemen beim Einfügen weiterer Folien, möglich ist. Dies ist aber insofern kein Problem, als der *DocumentFeatureMapper* für die spätere Vergabe oder Änderung der Titel ein Reparaturwerkzeug (vgl. Abschnitt3.4.2) anbietet.

Auf den Großteil der Zuordnung der Merkmalen zu Dokumentelementen soll an dieser Stelle nicht im Detail eingegangen werden, da diese, wie z. B. die generelle Zugehörigkeit von Folien zu Themen, einfach über die Abbildungsansicht in dem GUI erfolgt. Abbildung 4.20 zeigt exemplarisch die Zuordnung für die Präsentation der ersten Doppelstunde.

Analog zu den vorangegangenen Beispielen können über die Expertenansicht wieder präzisere Identifikatoren angegeben werden. Dies bietet sich insbesondere bei Merkmalen an, die in mehreren Dokumenten ähnlich auftreten. Um dies auch innerhalb der Identifikatoren repräsentieren zu können, bietet der *DocumentFeatureMapper* die Möglichkeit, innerhalb der realen Pfad- und Dateibezeichnung reguläre Ausdrücke zu verwenden. Somit können, wie in Tabelle 4.3 dargestellt, jeweils alle acht Grunddokumente mit einem Identifikator angesprochen werden. Dies funktioniert insbesondere in diesem Beispiel problemlos, da alle Dokumente auf demselben Master aufsetzen. Der URI 1 beschreibt die jeweils erste und letzte Seite einer Präsentation. Das jeweilige Elternelement aller Animationen auf einer Folie wird mit 2 beschrieben. Das Ende-Symbol (3) kann über seine Position identifiziert werden. Die Identifikatoren 4 und 5 beschreiben die Seitenzahl und den Zurückpfeil im Folienmaster. Die übrigen drei URIs behandeln die Folien, die nur aus einem Bild bestehen und das Logo, die Seitenzahl und den Pfeil nochmal enthalten. Gerade bei diesem Fall wird nochmal die Aufwandsersparnis sichtbar, da bei einer einzelnen Identifikation die betreffenden Folien zunächst erst einmal „manuell“ gesucht werden müssten.

Eine weitere interessante Art der Modifikation, die in den vorangegangenen Beispielen noch keine Anwendung gefunden hat, ist die Verwendung von Platzhaltern, um Attributinhalte von Merkmalen aus der Variantenspezifikation direkt in eine Modifikation einzubinden. Dies ist die Grundlage für sowohl die Titel- und Vorstellungsfolie als auch für die Terminfolien. Auf die Anwen-

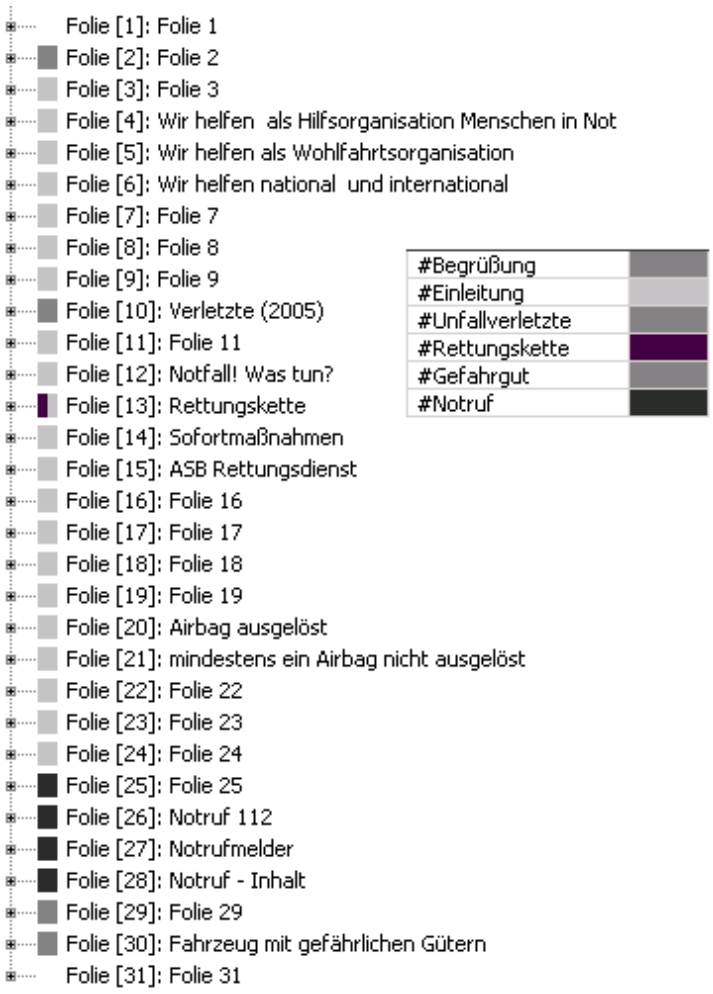


Abbildung 4.20
EH – 1. Doppelstunde für Szenario 3

dung soll erst im nächsten Abschnitt näher eingegangen werden, aber während der Zuordnung der Merkmale und Auszeichnung der Modifikation kann innerhalb der Ersetzung ein Ausdruck z. B. der Form `Ort: {AALSM.ort}` angegeben werden. AALSM ist dabei das Merkmal für die Folie „Ausbildungsangebot Lebensrettende Sofortmaßnahmen“ und „ort“ eines der Attribute des Merkmals, welches in einer konkreten Variante instantiiert wird. Die Modifikation wird dabei immer auf einen Absatz angewendet, der einfach über das GUI ausgewählt werden kann. Auf eine vollständige Liste soll an dieser Stelle wieder verzichtet werden. Die Arbeitsschritte sind für alle 13 weiteren Ausbildungsangebote sowie für die Attribute der Titel- und Vorstellungsseite analog anzuwenden[23].

Um der ursprünglichen Zielstellung gerecht zu werden und nur Dokumente

[23] Ähnlich den Monaten im vorangegangenen Beispiel kann hier durch eine spätere Möglichkeit der Ersetzung der Attribute auch innerhalb des Identifikators Redundanz und Instabilität entfernt werden.

Merkmal	Identifikator
Randfolien (1)	`regexp:source/EH-[1-8]\.odp !/content.xml#//draw:page[1]| //draw:page[last()]`
Animation (2)	`regexp:source/(.*)\.odp`[a] `!/content.xml#//draw:page/anim:*`
Ende-Symbol (3)	`!/content.xml#//draw:page/draw: custom-shape[@svg:x="23.918cm"]`
Zurückpfeil (4)	`!/styles.xml#/office:document-styles/ office:master-styles/style:master -page/draw:custom-shape[.//draw: enhanced-geometry[@draw:type= "right-arrow"]]`
Seitenzahl (5)	`!/styles.xml#/office:document-styles/ office:master-styles/style:master -page/draw:custom-shape [.//text:page-number]`
Overlaylogo (6)	`!/content.xml#//draw:page/draw:frame [./draw:image[@xlink:href= "Pictures/1..3C.jpg"]]`
Zurückpfeil (7)	`!/content.xml#//draw:page/draw:custom -shape[.//draw:enhanced-geometry [@draw:type="right-arrow"]]`
Seitenzahl (8)	`!/content.xml#//draw:page/draw:custom -shape[.//text:page-number]"`

[a]Jedem Identifikator vorangestellt

Tabelle 4.3
Identifikatoren für Szenario 3

im Zielordner zu erhalten, die auch benötigt werden, existiert die Möglichkeit, ein `process.xml` im Quellordner zu platzieren, welches eine Liste aller zu generierende,n Dokumente enthält (vgl. Abschnitt 3.5). Dieses Dokument kann selbst in das Mapping einbezogen werden und wird vom Interpreter zuerst ausgewertet. Basierend auf diesem Mechanismus enthält dieses Dokument also z. B. nur einen Verweis auf das Präsentationddokument mit der Titelfolie, wenn das Merkmal „Titel“ auch in der Variante vorhanden ist. Die Zuordnung erfolgt analog für sämtliche Quelldokumente.

Auf Grundlage des zuvor erstellten Modells und dem darauf aufbauenden Mapping lassen sich unzählige sinnvolle Varianten generieren. Im Folgenden sollen drei davon näher betrachtet werden.

4.3.4 Variante 1

In der ersten Variante wird im Rahmen einer anderen Veranstaltung ausschließlich der Hinweis auf die nächsten Termine der Erste-Hilfe-Ausbildung benötigt. Dazu könnte eine Variante wie in Abbildung 4.22 erstellt werden. Diese enthält

Abbildung 4.21
Vollständiges Zieldokument der Variante 1 für Szenario 3

bloß die beiden Merkmale für die Seiten der Ausbildungsangebote für die Lebensrettenden Sofortmaßnahmen und den Erste-Hilfe-Lehrgang und ordnet die entsprechenden Attribute zu. Auf sämtliche anderen Merkmale soll verzichtet werden.

⊟ Feature EH-Präsentation	■
⊟ Group 0	>=0
⊞ Feature Vorstellung	☐
⊞ Feature Titel	☐
⊞ Feature Layout	☐
⊞ Feature Themen	☐
⊞ Feature Kurse	☐
⊟ Feature Ausbildungsangebot	■
⊟ Group 1	>=1
⊟ Feature AALSM	■
Attribute uhrzeit	8:00
Attribute termine	20.03.2010
Attribute kosten	20
Attribute ort	Merseburg
Attribute url	http://www.asb-saalekreis.de
⊟ Feature AAEHF	■
Attribute uhrzeit	8:00
Attribute termine	20. und 22.03.2010
Attribute kosten	35
Attribute ort	Merseburg
Attribute url	http://www.asb-saalekreis.de
⊞ Feature AAEHB	☐
...	☐
⊞ Feature AASL	☐
Feature Separator	☐
Feature PR	☐
Feature Randfolien	☐
Feature Modifikationen	☐
Feature Entfernen	☐

Abbildung 4.22
Variante 1 für Szenario 3

Das Generieren dieser Variante führt zu einem Ordner, der nur den Foliensatz der Ausbildungsangebote enthält[24]. Dieser wiederum enthält nur die beiden Folien wie sie in Abbildung 4.21 dargestellt sind[25]. Die anderen Angebote, die weiteren Folien, die Seitenzahlen und die Navigationspfeile wurden entfernt.

[24] Hinzu kommt noch der PR-Foliensatz, dem kein Merkmal zugeordnet wurde und somit immer erzeugt wird.
[25] Der Hintergrund wurde für den Druck entfernt.

Da der Interpreter selbst keine Optimierung vornimmt, kann in einem Nachbearbeitungsschritt (vgl. Abschnitt 3.6.3) die Dateigröße des Zieldokuments reduziert werden, indem nun nicht mehr referenzierte Grafiken aus dem Dokument entfernt werden.

Basierend auf dieser Variante wäre es auch möglich, die Generierung solcher Foliensätze zu automatisieren. Ein kleines Script könnte die nächsten Termine aus einer Datenbank abfragen und daraus eine Variantenbeschreibung erstellen. Diese kann dann als Eingabe für das in Abschnitt 3.7 vorgestellte Werkzeug zur Automatisierung verwendet werden.

4.3.5 Variante 2

Die Variante 2 greift noch einmal den ursprünglichen Gedanken, in der Zielpräsentation nicht zwischen zu vielen Dokumenten wechseln zu müssen, auf. Als Beispiel soll hier der Kurs Erste-Hilfe-Lehrgang dienen. Dieser wird im Allgemeinen an zwei Tagen zu je vier Doppelstunden angeboten. Zu den Kursfolien soll noch ein Foliensatz mit der Titel- und Vorstellungsfolie hinzugefügt werden, so dass sich im Zielverzeichnis nach Abschluß der Generierung drei Dateien befinden.

In Abbildung 4.24 sind die vollständige Variantenbeschreibung und die Merkmalsattribute dargestellt. Darüber hinaus ist auch aufgezeigt, wie sich der Varianteneditor verhält, wenn noch nicht alle nötigen Themen-Merkmale für das Kurs-Merkmal „EHF“ ausgewählt wurden. Auf sämtliche Zusatzfolien, wie auch alle Layout-Merkmale wurde verzichtet.

Basierend auf dieser Variante werden die Titel- und die Vorstellungsfolie wie in Abbildung 4.23 erzeugt[26].

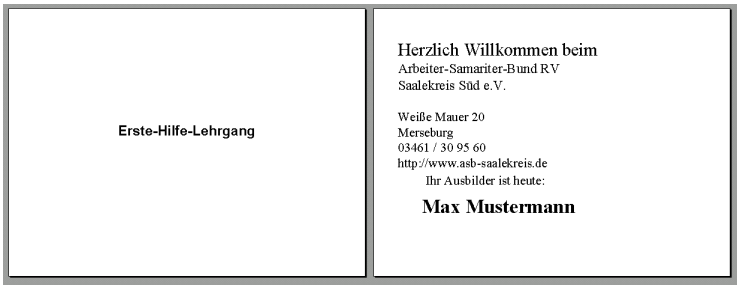


Abbildung 4.23
Erstes vollständiges Zieldokument der Variante 2 für Szenario 3

[26] Für den Druck wurde wieder auf den Hintergrund verzichtet

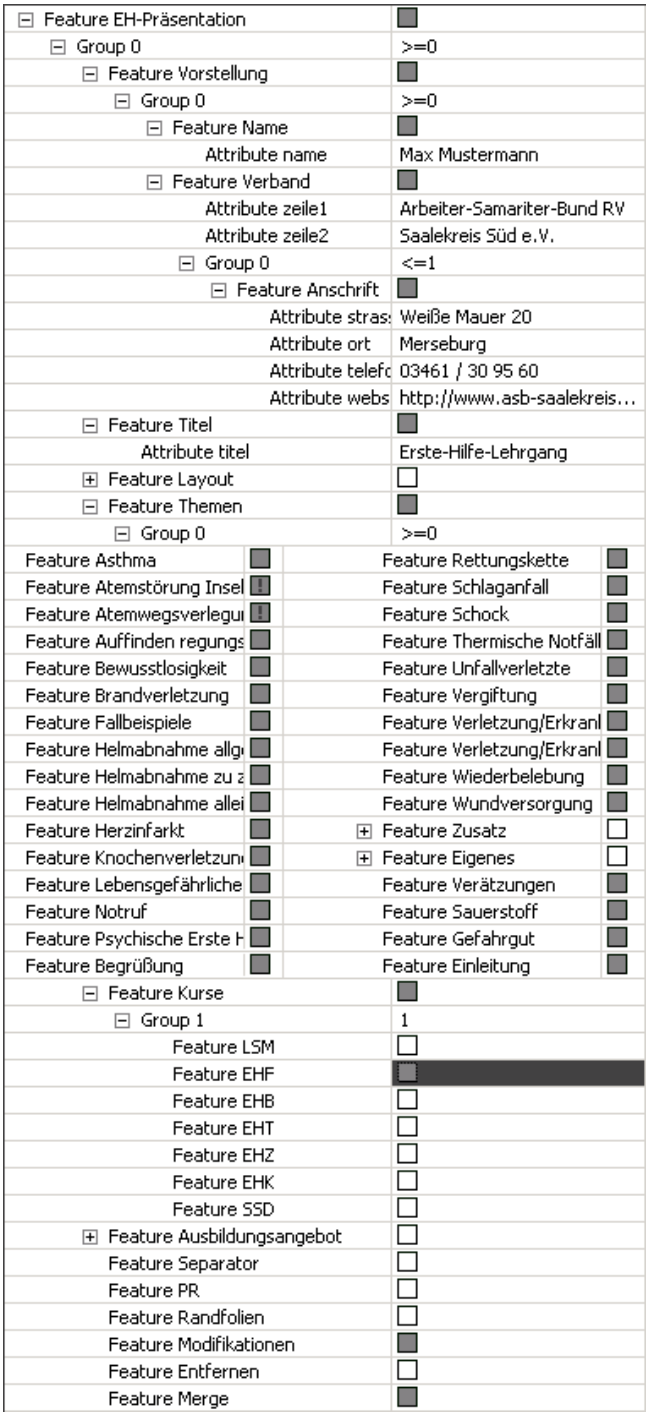


Abbildung 4.24
Variantenbeschreibung 2 für Szenario 3

Exemplarisch wurde auch dem Merkmal “Modifikationen“ eine Änderung des Folientextes zugeordnet. So hat sich zum Beispiel die Formulierung „Kopf nach hinten neigen und Kinn anheben“ auf der linken Folie in Abbildung 4.25 als für die Praxis im Lehrbetrieb ungeeignet erwiesen. Vereinzelt interpretierten Lehrgangsteilnehmer diese Anweisung so, dass sie den Kopf ausschließlich am Kinn bewegen, welches zu einer unnötigen Belastung der Wirbelsäule des Unfallopfers führen kann[27]. Der zweite Satzteil soll also entfernt werden, bis diese Änderung in das zentrale Originaldokument einfließt.

Abbildung 4.25
Modifikation in Variante 2 für Szenario 3

Für das Zusammenfügen zu zwei Tagespräsentationen bietet der *DocumentFeatureMapper* einen prototypischen Nachbearbeitungsschritt an (vgl. Abschnitt 3.6.4). In einer späteren Version wird dann nicht nur das hintereinander Anordnen von Präsentationen möglich sein, sondern auch das direkte Einfügen in die Präsentation bzw. das Ändern der Reihenfolge der einzelnen Folien. Ein weiterer Nachbearbeitungsschritt (vgl. Abschnitt 3.6.1) steht zur Verfügung, um die nun nicht mehr benötigten Einzelpräsentationen zu löschen.

Wie bereits zu Beginn des Abschnittes erwähnt, würde aber auch dieses Szenario stark davon profitieren, die Originalfolien durch eine striktere Nutzung von Formatvorlagen, das konsequente Verwenden von Folientiteln und später, mit den erweiterten Möglichkeiten des Zusammenfügens von Präsentationen, auch kleineren „Themenpräsentationen“ zu optimieren.

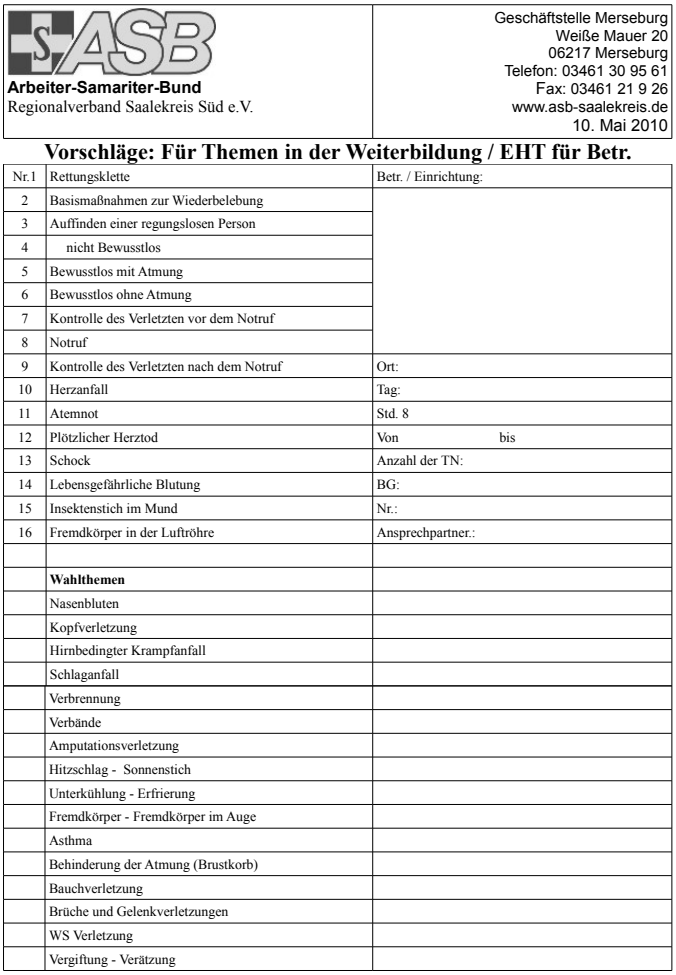
4.3.6 Variante 3

Die letzte Variante im Rahmen dieser Arbeit greift ein Szenario im Zusammenhang mit dem Kurs Erste-Hilfe-Training in Betrieben auf.

Durch eine Vielzahl unterschiedlicher Tätigkeiten innerhalb eines Betriebes verbessert sich die Qualität der Erste-Hilfe-Ausbildung erheblich, wenn auf die damit einhergehenden Notfallsituationen verstärkt eingegangen und kein universeller Foliensatz für alle Arten von Betrieben präsentiert wird. In einem Tischlereibetrieb könnte der Schwerpunkt eher auf Verbänden und Amputationverletzungen liegen, wohingegen dieses Wissen in Betrieben mit überwiegender

[27] Beispiel geht auf Erfahrungen eines Ausbilders zurück

Tätigkeit am Schreibtisch weniger von Nutzen ist. In chemisch geprägten Betrieben bietet sich ein Schwerpunkt auf Vergiftungen und Verätzungen an und im Umgang mit Kindern vielleicht eine Konzentration auf Asthma oder Nasenbluten.



Arbeiter-Samariter-Bund
Regionalverband Saalekreis Süd e.V.

Geschäftstelle Merseburg
Weiße Mauer 20
06217 Merseburg
Telefon: 03461 30 95 61
Fax: 03461 21 9 26
www.asb-saalekreis.de
10. Mai 2010

Vorschläge: Für Themen in der Weiterbildung / EHT für Betr.

Nr.1	Rettungsklette	Betr. / Einrichtung:
2	Basismaßnahmen zur Wiederbelebung	
3	Auffinden einer regungslosen Person	
4	nicht Bewusstlos	
5	Bewusstlos mit Atmung	
6	Bewusstlos ohne Atmung	
7	Kontrolle des Verletzten vor dem Notruf	
8	Notruf	
9	Kontrolle des Verletzten nach dem Notruf	Ort:
10	Herzanfall	Tag:
11	Atemnot	Std. 8
12	Plötzlicher Herztod	Von bis
13	Schock	Anzahl der TN:
14	Lebensgefährliche Blutung	BG:
15	Insektenstich im Mund	Nr.:
16	Fremdkörper in der Luftröhre	Ansprechpartner.:
	Wahlthemen	
	Nasenbluten	
	Kopfverletzung	
	Hirnbedingter Krampfanfall	
	Schlaganfall	
	Verbrennung	
	Verbände	
	Amputationsverletzung	
	Hitzschlag - Sonnenstich	
	Unterkühlung - Erfrierung	
	Fremdkörper - Fremdkörper im Auge	
	Asthma	
	Behinderung der Atmung (Brustkorb)	
	Bauchverletzung	
	Brüche und Gelenkverletzungen	
	WS Verletzung	
	Vergiftung - Verätzung	

Abbildung 4.26
Formular
Erste-Hilfe-Training in
Betrieben für Szenario 3

Um diesen Anforderungen gerecht zu werden, wird im ASB-Regionalverband Saalekreis Süd e.V. bei dem Kurs EHB dem Betrieb immer das Formular in Abbildung 4.26 zur Verfügung gestellt, um individuelle Schwerpunkte festlegen zu können. Mit dem zuvor abgeleiteten Featuremodell ist es nun mit deutlich geringerem Aufwand möglich, eine entsprechende Variante zu erzeugen, da weder

von Hand Folien gelöscht werden müssen, noch der Vortragende diese einfach bei der Präsentation überspringt.

Da sich der eigentliche Ablauf nicht von den vorangegangenen Varianten unterscheidet, soll hier auf eine detaillierte Darstellung verzichtet werden. Auch weitere Varianten sind mit Hilfe des Varianteneditors des *DocumentFeatureMappers* in kurzer Zeit erstellt, nachdem die einmalig etwas umfangreiche Zuordnung der Merkmale erfolgt ist.

4.4 Writer

Auf ein gesondertes Beispiel für ein Textdokument soll an dieser Stelle verzichtet werden. Sowohl die Motivation als auch der Grundgedanke der Featuremodelle können analog zu den vorangegangenen Beispielen abgeleitet werden. Abschnitt 4.5 stellt unterstützend einen Überblick über mögliche Merkmale bereit.

4.5 Generische Featureklassifikation für Dokumentfamilien

Die Anwendungsfälle haben gezeigt, dass die Featuremodelle für ein sinnvolles Mapping für jedes Szenario individuell ermittelt werden müssen. Ein zu allgemeines Modell kann die Abbildung der Merkmale auf Elemente des Quelldokuments unnötig verkomplizieren. Auch wenn es unter diesem Aspekt nicht *die* eine, alle Dokumente umfassende Klassifikation gibt, können dennoch vier Gruppen von Merkmalen unterschieden werden, die als Ausgangspunkt für die Erstellung eines spezifisches Featuremodell dienen können.

Abbildung 4.27 stellt die vier Gruppen grafisch in einer Übersicht dar und kann so als Grundlage für ein geeignetes Featuremodell für ein spezifische Dokumentfamilie dienen.

4.5.1 Layout und Satz

Die erste Gruppe, die unter der Bezeichnung „Layout und Satz“ zusammengefasst werden kann, enthält sämtliche Merkmale, die das Erscheinungsbild des Dokuments definieren. Darunter fällt z. B. die Wahl bestimmter Formatvorlagen oder Folienmaster, die Sichtbarkeit von Seitenzahlen und Grafiken oder die Aktivierung von Animationen und Folienübergängen. Ebenso gehören das Seitenformat und die Seitengröße oder sogar die Anordnung von Elementen innerhalb des Dokuments in diese Gruppe.

4.5.2 Zielgruppe und Anwendungsgebiet

In einer weiteren Gruppe lassen sich Merkmale zusammenfassen, die ein Dokument für eine bestimmte Zielgruppe oder ein spezielles Anwendungsgebiet anpassen. Hierzu gehört z. B. die Sprache, als auch deren Komplexität, die Anpassung der Farben an einen Graustufendruck oder die Verwendung stärkerer Kontraste. Auch der Zustand eines Dokuments (Entwurf, Geheim oder Öffentlich) oder das gewünscht Zielformat sind als Merkmale dieser Gruppe zuzuordnen.

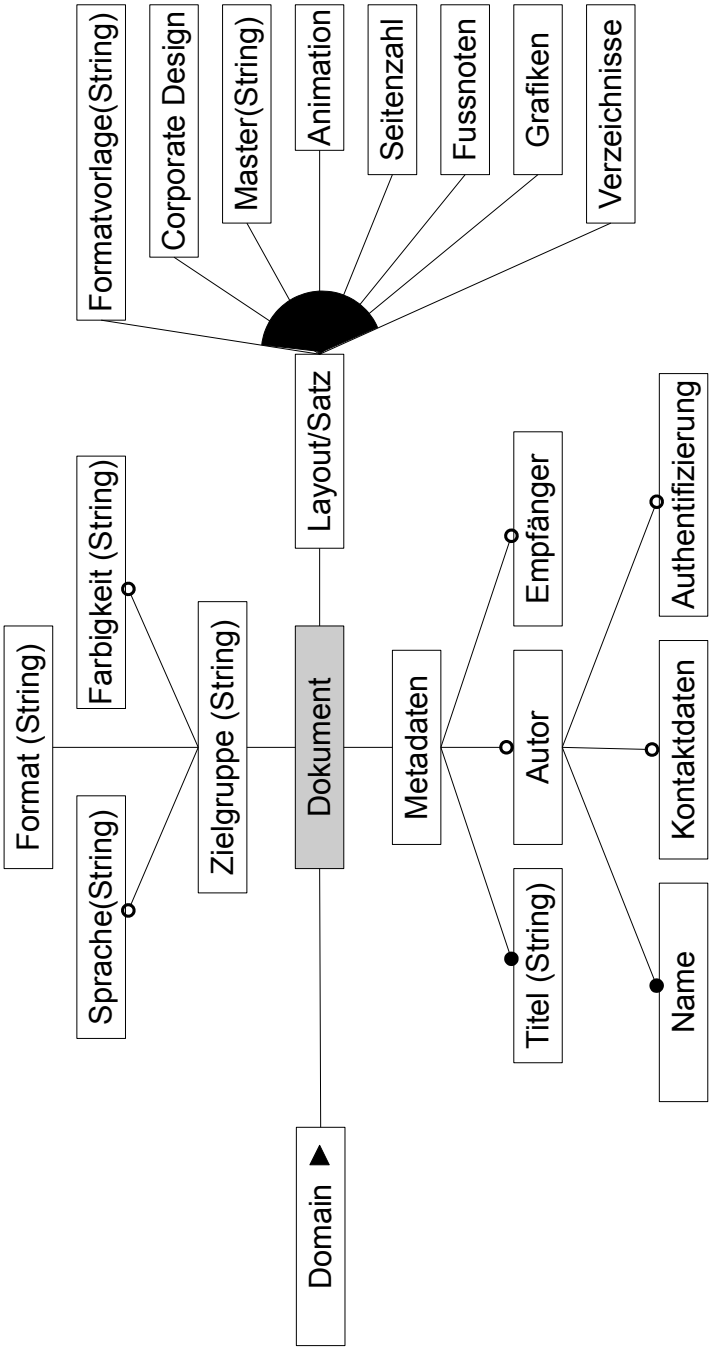


Abbildung 4.27
Einordnung von Merkmalen beliebiger Dokumente

4.5.3 Metadaten

Die Gruppe der „Metadaten“ beinhaltet Merkmale den Autor betreffend oder den Titel des generierten Dokuments. Die Versionsnummer oder eine fachliche Klassifikation wird von diesen Gruppen ebenfalls abgedeckt.

4.5.4 Domainspezifische Merkmale

Mit Abschnitt 4.3 konnte ein Szenario aufgezeigt werden, welches eine Vielzahl von untereinander verknüpften Merkmalen enthält, die direkt aus der Domain entnommen sind. Aber auch bei den kleineren Beispielen gehören die meisten Merkmale dieser Gruppe an.

Kapitel 5

Zusammenfassung, Bewertung und Ausblick

Zusammenfassung

Im Rahmen dieser Diplomarbeit konnte zunächst gezeigt werden, dass sich Dokumente und Dokumentfamilien analog zu den Softwareproduktlinien mit Hilfe von Featuremodellen beschreiben lassen.

Basierend auf dieser Erkenntnis wurde der templatebasierte Ansatz nach Czarnecki zur Generierung konkreter Modellinstanzen erfolgreich auf das Erzeugen spezifischer Dokumente einer Dokumentfamilie übertragen. Hierzu wurde ein Werkzeug entwickelt, welches dem Benutzer durch eine detaillierte Visualisierung der Quelldokumente eine einfache Zuordnung der Merkmale des Featuremodells zu Elementen des Dokuments erlaubt.

Das Werkzeug bietet eine Vielzahl an Schnittstellen an, die eine Erweiterung auf andere Dokumentformate, neben dem Open Document Format, ermöglichen. Darüber hinaus existiert ein prototypischer Nachbearbeitungsschritt, der Ansätze für eine aspektorientierte Weiterentwicklung aufzeigt.

Bewertung

Neben den theoretischen Möglichkeiten konnte mit den Anwendungsszenarien die Praxistauglichkeit des in dieser Arbeit vorgestellten Ansatzes untersucht werden. Es hat sich gezeigt, dass das Werkzeug bereits in seiner jetzigen Form produktiv eingesetzt werden kann. Für die leistungsstarken Funktionen, die derzeit noch ein überdurchschnittliches Verständnis der Struktur des Open Document Formats voraussetzen, gilt es noch, eine geeignete Abstraktion und Visualisierung zu finden, die dem Endnutzer entgegen kommt. Insbesondere die Identifikation von Dokumentelementen hat sich als kritischer Punkt herausgestellt, der gerade in Verbindung mit OpenOffice.org eine erhöhte Aufmerksamkeit fordert und dem im Einzelfall auch mit einem erhöhten Zeiteinsatz zu begegnen ist. Mit ähnlichen Aufwand ist auch bei zukünftigen Erweiterungen zu rechnen, wenn das Quellformat nicht selbst eine eindeutige Identifikation für sämtliche relevante Elemente liefert. Dennoch konnte gezeigt werden, dass akzeptable Kompromisse in Abhängigkeit der konkreten Anwendung existieren und die Vorteile der Merkmalsmodelle im Vergleich zum klassischen manuellen Erstellen von Dokumentfamilien deutlich überwiegen.

Parallel zu der eigentlichen Aufgabenstellung war es möglich die Überlegungen, die Grundlage für den FeatureMapper waren, und die mit dieser Arbeit neu entstandene Anforderungen in eine solide gemeinsame Architektur zu überführen. Eine umfangreichere Testabdeckung und auf Erweiterung ausgelegte Struktur (Anhang B) bilden eine konsolidierte Basis für die weitere Arbeit an beiden Forschungsschwerpunkten. Selbstverständlich konnte der FeatureMapper im Rahmen dieser Arbeit noch nicht vollständig integriert werden und so wird

z. B. die Validität der ODF-Zieldokumenten bisher auch nur indirekt über die Auswahlmöglichkeiten im Editor sichergestellt oder auf die Visualisierung einer Realisierung gänzlich verzichtet.

Ausblick

Im Hinblick auf die Weiterführung dieser Arbeit ergeben sich noch zwei interessante Aufgabenstellungen, deren Umsetzung die Akzeptanz durch einen Benutzer und damit die Praxistauglichkeit nochmals steigern können.

Die Integration von OpenOffice.org innerhalb des *DocumentFeatureMappers* ist derzeit noch rudimentär. Eine bidirektionale Synchronisation der Auswahl zwischen dem detaillierten Baumeditor und der Visualisierung des Dokuments in der OpenOffice.org-Ansicht ist aufgrund der Differenz des Open Document Formats zur Repräsentation der Struktur innerhalb von OpenOffice.org nicht trivial. Hier ist noch eine geeignete performante Lösung zu finden.

Der zweite Schwerpunkt, auf den im Rahmen dieser Arbeit nur bedingt eingegangen werden konnte, ist die Gewährleistung der Stabilität der Abbildung der Merkmale auf das Quelldokument bei Veränderungen. Es wurden erste Ansätze zu Stabilisierung oder Reparatur aufgezeigt, diese gilt es aber, im Besonderen unter dem Aspekt der Automatisierung, noch weiter zu untersuchen und darüber hinaus auch unabhängig vom Open Document Format zu generalisieren.

Ergänzend zu diesen beiden Aufgaben ist mit einer Ausdehnung auf andere Dokumentformate oder einer weiteren praktischen Erprobung mit neuen Reparatur- und Nachbearbeitungsschritten zu rechnen. Ebenso bietet der Nachbearbeitungsschritt für das Zusammenfügen von Präsentationen noch Potential, hin zu einer detaillierten Aspektorientierung, durch ein feingranulareres und allgemeineres Einfügen.

Nicht zuletzt gilt es auch, die Fähigkeiten der Validierung des FeatureMappers in die allgemeinere Architektur des *DocumentFeatureMappers* zu integrieren.

Anhang A

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[Berners-Lee 2005] Berners-Lee, T.: *Uniform Resource Identifier (URI): Generic Syntax*. New York, NY, USA: , 2005. – URL `http://tools.ietf.org/html/rfc3986`

[Brown 2008] Brown, Alex: *ODF 1.0 and OpenOffice.org: a conformance smoke test*. 2008. – URL `http://www.griffinbrown.co.uk/blog/PermaLink.aspx?guid=f0384bed-808b-49a8-8887-ea7cde5caace`

[Brownsword u. a. 1996] Brownsword, L. ; Clements, P. ; Olsson, P.: Successful Product Line Engineering: A Case Study. In: *Software Technology Conference, Salt Lake City*, 1996

[Czarnecki 2005] Czarnecki, Krzysztof: Mapping features to models: A template approach based on superimposed variants. In: *GPCE 2005 - Generative Programming and Component Enginering. 4th International Conference*, Springer, 2005, S. 422–437

[Czarnecki u. a. 2005] Czarnecki, Krzysztof ; Helsen, Simon ; Eisenecker, Ulrich W.: Formalizing cardinality-based feature models and their specialization. In: *Software Process: Improvement and Practice* 10 (2005), Nr. 1, S. 7–29

[Czarnecki und Kim 2005] Czarnecki, Krzysztof ; Kim, Chang Hwan P.: Cardinality-Based Feature Modeling and Constraints: A Progress Report. (2005)

[Gacek und Anastasopoules 2001] Gacek, Critina ; Anastasopoules, Michalis: Implementing product line variabilities. In: *SIGSOFT Softw. Eng. Notes* 26 (2001), May, S. 109–117. – URL `http://doi.acm.org/10.1145/379377.375269`. – ISSN 0163-5948

[Gamma u. a. 1993] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. 1993

[Harris 1997] Harris, Stephen L.: Managing virtual documents: correctness by design. In: *SIGDOC '97: Proceedings of the 15th annual international conference on Computer documentation*. New York, NY, USA : ACM, 1997, S. 131–135. – ISBN 0-89791-861-4

[Heidenreich 2009] Heidenreich, Florian: Towards Systematic Ensuring Well-Formedness of Software Product Lines. In: *In Proceedings of the 1st Workshop on Feature-Oriented Software Development*. New York, NY, USA : ACM, oct 2009, S. 69–74. – ISBN 978-1-60558-567-3

[Hunt und Thomas 1999] Hunt, Andrew ; Thomas, David: *The pragmatic programmer: from journeyman to master*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 0-201-61622-X

[Kang u. a. 1990] Kang, K. C. ; Cohen, S. G. ; Hess, J. A. ; Novak, W. E. ; Peterson, A. S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study / Carnegie-Mellon University Software Engineering Institute. November 1990. – Forschungsbericht

[Kang u. a. 1998] Kang, Kyo C. ; Kim, Sajoong ; Lee, Jaejoon ; Kim, Kijoo ; Kim, Gerard J. ; Shin, Euiseob: FORM: A feature-oriented reuse method with domain-specific reference architectures. In: *Annals of Software Engineering* 5 (1998), S. 143–168

[Kang u. a. 2002] Kang, Kyo C. ; Lee, Jaejoon ; Donohoe, Patrick: Feature-Oriented Product Line Engineering. In: *IEEE Software* 19 (2002), S. 58–65. – ISSN 0740-7459

[Kiczales u. a. 1997] Kiczales, G. ; Lamping, J. ; Menhdhekar, A. ; Maeda, C. ; Lopes, C. ; Loingtier, J.M. ; Irwin, J.: *Aspect-Oriented Programming, volume 1241*. 1997

[Kopcsek 2007] Kopcsek, Jan: Entwurf und Implementierung eines Werkzeugs zur featuregetriebenen Modellierung mittels domänenspezifischer Sprachen / TU Dresden. 2007. – Forschungsbericht

[Niederhausen u. a. 2009] Niederhausen, Matthias ; Karol, Sven ; Assmann2, Uwe ; Meissner, Klaus: HyperAdapt: Enabling Aspects for XML. (2009)

[Northrop 2009] Northrop, Linda: Software Product Lines Essentials. (2009)

[OpenOffice.org 2009a] OpenOffice.org: *Market Share Analysis*. 2009. – URL `http://wiki.services.openoffice.org/w/index.php?title=Market_Share_Analysis&oldid=145776`

[OpenOffice.org 2009b] OpenOffice.org: *OpenOffice.org Developer's Guide*. 2009. – URL http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/FirstSteps

[Steinberg u. a. 2009] Steinberg, Dave ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed ; Gamma, Erich (Hrsg.) ; Nackman, Lee (Hrsg.) ; Wiegand, John (Hrsg.): *EMF: Eclipse Modeling Framework (2nd Edition) (Eclipse)*. 2nd Revised edition (REV). Addison-Wesley Longman, Amsterdam, January 2009. – URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0321331885. – ISBN 0321331885

[Walter 2002] Walter, Cornelius: Softwareproduktlinien: Einführung und Überblick. (2002)

[Weir 2008] Weir, Rob: *ODF Validation for Dummies*. 2008. – URL http://www.robweir.com/blog/2008/05/odf-validation-for-dummies.html

Anhang B

Dokumentation

Im Anhang B werden Fragestellungen im Zusammenhang mit der konkreten Implementierung behandelt. Weiterhin wird ein Überblick über die mögliche Verwendung des in der Arbeit vorgestellten Werkzeugs und die Erweiterung für eigene XML-basierte Formate gegeben.

B.1 Quellcode/Testfälle/Javadoc

Informationen zum Bezug des aktuellen Quellcodes einschließlich der Testfälle und Javadoc sind unter [1] zu finden.

B.2 Wie implementiere ich ...

B.2.1 ...einen eigenen Interpreter

Interpreter implementieren das Interface IInterpreter[2], welches die Methode `interpret()` zur Verfügung stellt. Einem Interpreter wird ein Mapping zugeordnet, und anschließend können mit dieser Funktion unter Angabe einer Variante und des Zielordners die Zieldateien generiert werden.

Weitere Methoden können der Javadoc entnommen werden.

Die abstrakte Klasse AbstractInterpreter[3] stellt einige Methoden zur Auswertung der Variante zur Verfügung, die auch in anderen Interpretern Anwendung finden können.

Für die Integration eines Interpreters in die Benutzeroberfläche müssen neue Plugins den Erweiterungspunkt *de.mheinzerling.dfm.gui.interpreter* spezifizieren.

Um auf Änderungen an XML-Dateien während der Generierung zu lauschen, muss eine Klasse das Interface IXMLInterpreterListener[4] implementieren und anschließend beim Erweiterungspunkt *de.mheinzerling.dfm.interpreter.listener* registriert werden.

B.2.2 ...einen eigenen Editor

Ein neuer Editor, der als Quelle für die Abbildungsansicht funktionieren soll, muss als SelectionProvider innerhalb von Eclipse registriert werden.[5] Dabei muss eine IStructuredSelection geliefert werden, deren Elemente IIdentifyable[6] implementieren.[7]

Soll ein Editor die aktuelle Abbildung in einer beliebigen Form visualisieren, muss dieser das Interface IMappingStateListener[8] implementieren, eine Registrierung bei der Abbildungsansicht erfolgt automatisch mit der ersten Selek-

[1] http://www.mheinzerling.de/tud.php
[2] Javadoc: de.mheinzerling.dfm.core.interpreter.IInterpreter
[3] Javadoc: de.mheinzerling.dfm.interpreter.AbstractInterpreter
[4] Javadoc: de.mheinzerling.dfm.core.interpreter.IXMLInterpreterListener
[5] In einem normalen EditorPart mit `getSite().setSelectionProvider(...);`,wobei ... meist der enthaltene Viewer ist.
[6] Javadoc: de.mheinzerling.dfm.core.identifiable.IIdentifyable
[7] Das heißt der dem Editor zugrunde liegende Viewer hält Elemente, die `getURI()` und Informationen bzgl. des Entfernens bzw. Modifizierens enthalten.
[8] Javadoc: de.mheinzerling.dfm.core.mappingstate.IMappingStateListener

tion. Der so zu erhaltene MappingState[9] enthält dann die benötigten Farben und eine Angabe, ob eine Modifikation vorliegt.

Häufigste Fehler in Verbindung mit den Editoren sind das Fehlen eines geöffneten Mappings, oder die Datei im Editor ist nicht Teil der Quelldokumente!

B.2.3 ... einen eigenen XML- und ODF-Baumeditor

Grundsätzlich gibt es eigentlich nur ein Szenario, das es erforderlich macht, einen neuen XML-Editor zu erstellen, und zwar das Einführen eines neuen Archivs mit XML-Dateien, z. B. die Unterstützung von OpenXML. Hier wäre ein Editor von XMLTreeEditor[10] abzuleiten, der analog zum ODFTreeEditor[11] die Methode `createIndividualPages()` mit den entsprechenden Dateien im Archiv spezifiziert.

In allen anderen Fällen, insbesondere bei einfachen XML-Dateien und der Vervollständigung des ODF-Editors, genügt es, neue XML-Elemente bzw. Attribute in den Erweiterungspunkten *de.mheinzerling.dfm.editor.element* und *de.mheinzerling.dfm.editor.attribute* bekannt zu machen. Bei beiden Erweiterungspunkten kann eine Klasse (XMLEditorElement[12] und XMLEditorAttribute[13]) angegeben werden, die die Beschriftung, Identifikation und Bearbeitungsmöglichkeiten (ValueTypeDescriptor[14]) vom Standard abweichen lässt.

In diesem Zusammenhang können über den Erweiterungspunkt *de.mheinzerling.dfm.editor.filter* auch zusätzliche Filterkonstellationen hinzugefügt werden.[15]

B.2.4 ... einen Adapter für einen vorhandenen Editor

Adapter für einen vorhanden Editor wandeln den Inhalt einer IStructuredSelection, die vom Editor an die Abbildungsansicht geliefert wird, in Objekte, die IIdentifyable[16] implementieren, um. Hierzu instantiiert ein Adapter das Interface SelectionAdapter[17]. Für einen bidirektionalen Adapter, der die Zuordnung im Editor darstellen soll, ist die Klasse MappingStateListenerSelectionAdapter[18] zu erweitern.

[9] Javadoc: de.mheinzerling.dfm.core.mappingstate.MappingState
[10] Javadoc: de.mheinzerling.dfm.editor.XMLTreeEditor
[11] Javadoc: de.mheinzerling.dfm.editor.odf.ODFTreeEditor
[12] Javadoc: de.mheinzerling.dfm.editor.XMLEditorElement
[13] Javadoc: de.mheinzerling.dfm.editor.XMLEditorAttribute
[14] Javadoc: de.mheinzerling.dfm.core.identifiable.ValueTypeDescriptor
[15] Elemente dürfen auch in mehreren Filtern auftreten.
[16] Javadoc: de.mheinzerling.dfm.core.identifiable.IIdentifyable
[17] Javadoc: de.mheinzerling.dfm.gui.adapter.SelectionAdapter
[18] Javadoc: de.mheinzerling.dfm.gui.adapter.MappingStateListenerSelectionAdapter

Auf diese Weise erstellte Adapter müssen sich am Erweiterungspunkt *de.mheinzerling.dfm.gui.adapter* bekannt machen.

Für den leichteren Einstieg wurde bereits ein Adapter (EMFSelectionAdapter[19]) für generierte EMF-Editoren erstellt. Dieser wandelt zunächst die EObjects in IIdentifyable[20] um. Darüber hinaus wird in jedem StructuredViewer, der eine IStructuredSelection liefert, der LabelProvider gegen einen erweiterten ColorAdapterLabelProvider[21] ausgetauscht, um eine ähnliche Farbgebung wie in dem XML-Editor zu erzeugen.

B.2.5 ... ein eigenes Reparaturwerkzeug

Reparaturwerkzeuge implementieren das Interface IRepair[22] mit der Funktion `repair()`, welche ausschließlich von einem Mapping abhängt. Umfangreichere Reparaturmaßnahmen können darüber die Quelldateien ausfindig machen und verarbeiten. Die weiteren Methoden für die Integration in die Benutzeroberfläche können der Javadoc entnommen werden.

Für weitere einfache Reparaturwerkzeuge stellt die abstrakte Klasse SimpleRepair[23] die Integration in die Benutzeroberfläche zur Verfügung. Exemplarisch können hier StringRepair[24] oder FileNameRepair[25] als Einstiegspunkte dienen.

Da die meisten Reparaturmaßnahmen auf dem Ersetzen von Identifikatoren beruhen, wurde diese Funktionalität zur Wiederverwendung in der Klasse IdentifierReplacer[26] zusammengefasst.

Der Benutzeroberfläche können Reparaturwerkzeuge über den Erweiterungspunkt *de.mheinzerling.dfm.gui.repair* bekannt gemacht werden.

B.2.6 ... einen eigenen Nachbearbeitungsschritt

Jeder Nachbearbeitungsschritt implementiert das Interface IPostprocessor[27]. Detaillierte Informationen zu den einzelnen zur Verfügung zu stellenden Methoden sind der Javadoc zu entnehmen. Den Kern bildet dabei die Funktion `process()`, die basierend auf dem Zielverzeichnis die eigentliche Operation des Nachbearbeitungsschrittes ausführt.

[19] Javadoc: de.mheinzerling.dfm.gui.adapter.EMFSelectionAdapter
[20] Javadoc: de.mheinzerling.dfm.core.identifiable.IIdentifyable
[21] Javadoc: de.mheinzerling.dfm.gui.adapter.ColorAdapterLabelProvider
[22] Javadoc: de.mheinzerling.dfm.core.repair.IRepair
[23] Javadoc: de.mheinzerling.dfm.repair.SimpleRepair
[24] Javadoc: de.mheinzerling.dfm.repair.StringRepair
[25] Javadoc: de.mheinzerling.dfm.repair.FileNameRepair
[26] Javadoc: de.mheinzerling.dfm.repair.IdentifierReplacer
[27] Javadoc: de.mheinzerling.dfm.core.postprocessor.IPostprocessor

Hinzu kommt eine Methode, die im Rahmen der Automatisierung zum Laden der Konfiguration Verwendung findet, sowie fünf weitere, die für die Bereitstellung von Informationen für die Integration in die graphische Benutzeroberfläche zuständig sind.

Um einen Nachbearbeitungsschritt dem GUI bekannt zu machen, müssen neue Plugins den Erweiterungspunkt *de.mheinzerling.dfm.gui.postprocessor* spezifizieren.

Als allgemeiner Einstieg für die eigene Arbeit kann ConnectorRepairPostprocessor[28] dienen. Für die Schnittstellen zum GUI oder der Automatisierung ist FileRenamePostprocessor[29] zu empfehlen.

B.3 Hilfsklassen

Im Paket de.mheinzerling.dfm.core.util sind eine Vielzahl von Hilfsklassen enthalten um z. B. den Umgang mit XPaths, URIs, ZIPs, JDOM oder der Eclipse Extension Registry zu vereinfachen. Darüber hinaus bietet de.mheinzerling.dfm.core.testutil Unterstützung für Tests auf XML-Dokumenten und auch ein kleines Framework für SWTBOT[30]-Tests. Für die weitere Arbeit mir ODF sind in de.mheinzerling.dfm.core.odf weitere Hilfsklassen zu finden. Die konkreten Details können der Javadoc entnommen werden.

Es ist mit Nachdruck erwünscht diese Klassen zu verbessern und bei zukünftigen Anforderungen zu erweitern!

B.4 Testabdeckung

Tabelle B.1 zeigt die Testabdeckung über alle Plugins. Die Tabellen B.2, B.3 und B.4 führen die Angaben detaillierter für jedes Paket auf. Sämtliche Angaben beziehen sich auf den Zeitpunkt des Druckes dieser Arbeit und können auf der beigefügten CD schon höher sein.

Klassen	88% (374/425)	Methoden	78% (1772/2266)
Anweisungen	78% (33775/43387)	Zeilen	75% (7797,7/10413)
Pakete	77	Dateien	10413

Tabelle B.1 Zusammenfassung Testabeckung

[28] Javadoc: de.mheinzerling.dfm.postprocessor.connectorrepair.ConnectorRepairPostprocessor
[29] Javadoc: de.mheinzerling.dfm.postprocessor.filerename.FileRenamePostprocessor
[30] http://www.eclipse.org/swtbot

Klassen	Methoden	Anweisungen	Zeilen
de.mheinzerling.dfm.artefacts.common			
0% (0/1)	0% (0/1)	0% (0/4)	0% (0/2)
de.mheinzerling.dfm.artefacts.featuremapping			
100% (8/8)	92% (80/87)	86% (1018/1184)	87% (253/291)
de.mheinzerling.dfm.artefacts.featuremapping.term			
100% (9/9)	96% (95/99)	94% (1507/1595)	92% (348,6/377)
de.mheinzerling.dfm.artefacts.filelist			
100% (2/2)	69% (9/13)	58% (89/153)	52% (26/50)
de.mheinzerling.dfm.artefacts.test			
86% (6/7)	97% (28/29)	100% (1524/1528)	99% (334/336)
de.mheinzerling.dfm.artefacts.variant			
100% (2/2)	97% (28/29)	91% (525/579)	90% (144/160)
de.mheinzerling.dfm.automation			
80% (4/5)	65% (28/43)	46% (297/639)	51% (82/162)
de.mheinzerling.dfm.automation.test			
50% (1/2)	67% (2/3)	98% (138/141)	96% (27/28)
de.mheinzerling.dfm.core.artefact			
100% (4/4)	76% (22/29)	87% (482/555)	83% (100/120)
de.mheinzerling.dfm.core.artefact.impl			
100% (4/4)	83% (10/12)	69% (65/94)	58% (21/36)
de.mheinzerling.dfm.core.artefact.util			
100% (2/2)	83% (10/12)	78% (60/77)	74% (17,8/24)
de.mheinzerling.dfm.core.common			
100% (3/3)	100% (12/12)	97% (71/73)	96% (24/25)
de.mheinzerling.dfm.core.documentmediator			
33% (1/3)	56% (9/16)	57% (152/266)	57% (33,8/59)
de.mheinzerling.dfm.core.exception			
50% (2/4)	39% (7/18)	51% (40/79)	38% (11/29)
de.mheinzerling.dfm.core.identifiable			
50% (2/4)	33% (4/12)	38% (33/87)	41% (7/17)
de.mheinzerling.dfm.core.interaction			
60% (3/5)	39% (7/18)	57% (93/163)	49% (18/37)
de.mheinzerling.dfm.core.mappingstate			
100% (3/3)	64% (7/11)	36% (97/272)	46% (31/67)
de.mheinzerling.dfm.core.odf			
67% (2/3)	56% (5/9)	84% (74/88)	81% (26/32)
de.mheinzerling.dfm.core.postprocessor			
50% (1/2)	40% (4/10)	45% (18/40)	44% (8/18)
de.mheinzerling.dfm.core.state			
100% (3/3)	75% (9/12)	94% (245/260)	92% (8,3/9)
de.mheinzerling.dfm.core.test			
100% (2/2)	75% (6/8)	80% (280/351)	61% (72/119)
de.mheinzerling.dfm.core.testutil			
94% (17/18)	78% (66/85)	60% (1009/1679)	53% (191,7/360)
de.mheinzerling.dfm.core.util			

Tabelle B.2
Detailierte Testabdeckung 1

Klassen	Methoden	Anweisungen	Zeilen
100% (8/8)	86% (67/78)	76% (1889/2480)	75% (456,2/611)
de.mheinzerling.dfm.editor			
100% (11/11)	72% (59/82)	72% (620/863)	71% (172,6/244)
de.mheinzerling.dfm.editor.selection			
100% (4/4)	95% (18/19)	92% (213/231)	91% (61/67)
de.mheinzerling.dfm.editor.test			
50% (1/2)	88% (7/8)	77% (222/289)	76% (52/68)
de.mheinzerling.dfm.editor.xml			
100% (3/3)	94% (16/17)	93% (307/330)	94% (84/89)
de.mheinzerling.dfm.editor.xml.common			
100% (8/8)	74% (45/61)	81% (658/815)	77% (151,8/196)
de.mheinzerling.dfm.editor.xml.odf			
75% (3/4)	73% (16/22)	60% (149/249)	62% (47/76)
de.mheinzerling.dfm.editor.xml.odf.attributes			
22% (2/9)	22% (2/9)	22% (10/45)	22% (4/18)
de.mheinzerling.dfm.editor.xml.odf.elements			
77% (10/13)	58% (29/50)	75% (477/637)	68% (100/148)
de.mheinzerling.dfm.editor.xml.other			
0% (0/1)	0% (0/5)	0% (0/49)	0% (0/12)
de.mheinzerling.dfm.gui			
100% (2/2)	80% (8/10)	85% (28/33)	89% (16/18)
de.mheinzerling.dfm.gui.adapter			
67% (2/3)	45% (9/20)	26% (54/211)	27% (17/62)
de.mheinzerling.dfm.gui.common			
100% (7/7)	82% (28/34)	90% (384/429)	84% (109/129)
de.mheinzerling.dfm.gui.common.viewer			
100% (6/6)	83% (24/29)	74% (306/415)	78% (86,3/110)
de.mheinzerling.dfm.gui.test			
89% (8/9)	98% (61/62)	100% (2771/2778)	100% (570,6/572)
de.mheinzerling.dfm.gui.view			
100% (27/27)	95% (108/114)	91% (1782/1967)	88% (419,4/477)
de.mheinzerling.dfm.gui.view.featuremodel			
100% (16/16)	96% (54/56)	91% (1186/1306)	91% (275,8/304)
de.mheinzerling.dfm.gui.view.modification			
100% (13/13)	98% (45/46)	97% (780/802)	97% (187,4/193)
de.mheinzerling.dfm.gui.view.sourcedocument			
100% (7/7)	100% (23/23)	97% (347/358)	93% (86/92)
de.mheinzerling.dfm.gui.view.term			
94% (16/17)	92% (60/65)	94% (1011/1071)	92% (222,4/241)
de.mheinzerling.dfm.gui.view.term.checkbox			
100% (4/4)	95% (20/21)	94% (438/464)	91% (89,8/99)
de.mheinzerling.dfm.gui.view.term.termpart			
100% (12/12)	97% (56/58)	95% (948/995)	92% (227,6/247)
de.mheinzerling.dfm.gui.wizard			
100% (9/9)	100% (32/32)	79% (463/584)	78% (106,4/137)
de.mheinzerling.dfm.gui.wizard.generation			
100% (16/16)	97% (60/62)	93% (1188/1274)	90% (275,3/306)

Tabelle B.3
Detailierte Testabdeckung 2

Klassen	Methoden	Anweisungen	Zeilen
de.mheinzerling.dfm.gui.wizard.neww			
100% (7/7)	97% (31/32)	81% (372/461)	75% (99,4/132)
de.mheinzerling.dfm.interpreter			
100% (4/4)	100% (18/18)	91% (393/432)	92% (79,8/87)
de.mheinzerling.dfm.interpreter.test			
75% (3/4)	82% (14/17)	88% (297/336)	85% (72,9/86)
de.mheinzerling.dfm.interpreter.xml			
100% (2/2)	100% (21/21)	80% (602/753)	82% (140/170)
de.mheinzerling.dfm.ooview			
0% (0/1)	0% (0/5)	0% (0/24)	0% (0/12)
de.mheinzerling.dfm.ooview.views			
0% (0/6)	0% (0/28)	0% (0/464)	0% (0/129)
de.mheinzerling.dfm.perspective			
100% (1/1)	100% (2/2)	100% (33/33)	100% (11/11)
de.mheinzerling.dfm.postprocessor			
100% (3/3)	62% (5/8)	91% (64/70)	80% (16/20)
de.mheinzerling.dfm.postprocessor.connectorrepair			
100% (4/4)	74% (14/19)	81% (405/499)	76% (71/94)
de.mheinzerling.dfm.postprocessor.filecleanup			
100% (2/2)	62% (10/16)	57% (128/226)	59% (34/58)
de.mheinzerling.dfm.postprocessor.filerename			
100% (1/1)	92% (12/13)	87% (206/236)	89% (50/56)
de.mheinzerling.dfm.postprocessor.objectcleanup			
100% (4/4)	81% (17/21)	78% (261/335)	77% (66/86)
de.mheinzerling.dfm.postprocessor.presentationmerge			
100% (1/1)	50% (6/12)	63% (150/238)	59% (33/56)
de.mheinzerling.dfm.postprocessor.presentationmerge.mergedescriptor			
100% (5/5)	68% (23/34)	65% (240/369)	65% (68/105)
de.mheinzerling.dfm.postprocessor.presentationmerge.process			
100% (10/10)	87% (45/52)	96% (1143/1192)	93% (264/284)
de.mheinzerling.dfm.postprocessor.tablerepair			
100% (4/4)	76% (16/21)	85% (386/454)	84% (81/97)
de.mheinzerling.dfm.postprocessor.test			
88% (7/8)	93% (14/15)	99% (358/361)	99% (86/87)
de.mheinzerling.dfm.repair			
100% (5/5)	78% (21/27)	58% (209/362)	54% (53/99)
de.mheinzerling.dfm.test			
0% (0/4)	0% (0/10)	0% (0/217)	0% (0/46)
de.mheinzerling.dfm.test.common			
100% (1/1)	100% (8/8)	100% (597/597)	100% (101/101)
de.mheinzerling.dfm.test.interpreter			
100% (2/2)	100% (6/6)	98% (328/336)	94% (95/101)
de.mheinzerling.dfm.test.repair			
100% (4/4)	100% (13/13)	100% (722/722)	100% (156/156)
de.mheinzerling.dfm.test.samples			
100% (4/4)	100% (9/9)	100% (182/182)	100% (52/52)

Tabelle B.4
Detailierte Testabdeckung 3

B.5 Plugin- und Paketstruktur

Abbildung B.1 zeigt die Abhängigkeiten zwischen sämtlichen Plugins. In der Abbildung B.2 ist dann ein Einblick in das Plugin DE.MHEINZERLING.DFM.CORE gegeben.

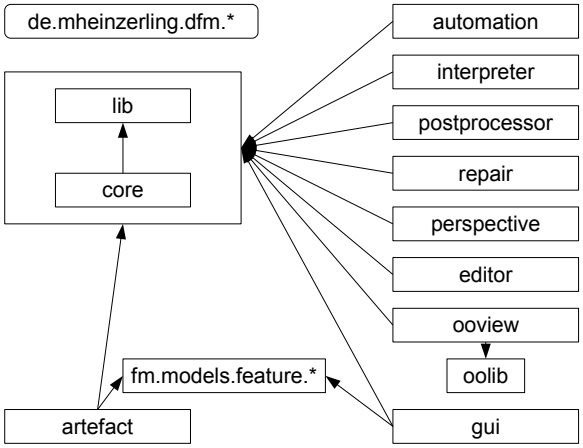


Abbildung B.1
Pluginabhängigkeiten

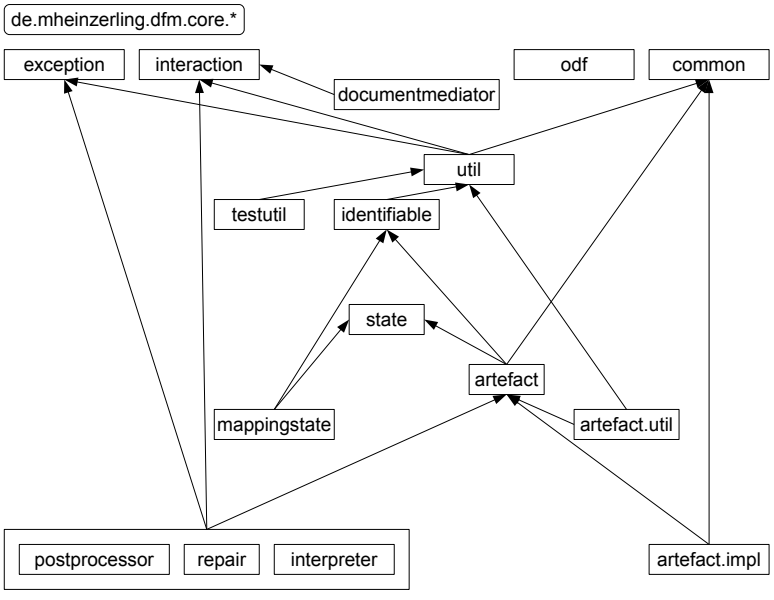


Abbildung B.2
Abhängigkeiten
DE.MHEINZERLING.DFM.CORE

Anhang C

Lizenzhinweis

Die hier vorgestellten und der Arbeit beiliegenden mehrteiligen Präsentationen des Arbeiter-Samariter-Bunds dürfen ohne besondere Genehmigung nur durch ASB-Lehrkräfte mit gültiger Lehrberechtigung im Auftrag des ASBs eingesetzt und nicht an Dritte weitergegeben werden. Im Rahmen dieser und darauf aufbauender Arbeiten dürfen die Präsentationen für rein wissenschaftliche Zwecke verwendet werden.