Johannes Kepler Universität Linz

Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

---

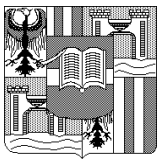
Vorgaben für Web-Anwendungen

Ein constraintbasierter Ansatz zur Definition und Durchsetzung von Vorgaben in verteilten Organisationen

Diplomarbeit zur Erlangung des akademischen Grades eines Magisters der Sozial- und Wirtschaftswissenschaften (Mag.rer.soc.oec)

Eingereicht bei o. Univ.-Prof. Dr. Michael Schrefl
Betreuung Mag. Christian Eichinger

Verfasst von
Katharina Grün



Linz, Dezember 2004

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Linz, im Dezember 2004

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die durch ihre fachliche und persönliche Unterstützung zur Entstehung dieser Diplomarbeit beigetragen haben.

Herrn o. Univ.-Prof. Dr. Michael Schrefl und Herrn Mag. Christian Eichinger möchte ich für die Unterstützung bei der Erstellung dieser Arbeit danken. Mag. Christian Eichinger trug durch seine engagierte Betreuung sowie seine fachlichen Ratschläge wesentlich zu dieser Arbeit bei.

Mein besonderer Dank richtet sich an meine Eltern, die mir das Studium ermöglichten und mir während der gesamten Studiendauer zur Seite standen.

Schließlich möchte ich meinen Geschwistern und Freunden für ihre Unterstützung während des Studiums danken.

Kurzfassung

Organisationen setzen Web-Anwendungen ein, um Inhalte und Dienste im Web anzubieten. Um den Besuchern der Web-Anwendung das Auffinden der gewünschten Information zu erleichtern, sollen die Web-Anwendungen einer Organisation ein einheitliches Erscheinungsbild aufweisen und gleiche Information einheitlich bereitstellen. Gerade für verteilte Organisationen ist es wünschenswert, dass diese Vereinheitlichung nicht durch eine zentrale Verwaltung der Web-Anwendungen erreicht wird, sondern dass die Organisationseinheiten Änderungen an ihren Web-Anwendungen selbst durchführen können. Um sowohl ein einheitliches Erscheinungsbild als auch Flexibilität garantieren zu können, beschäftigt sich diese Arbeit mit der Definition von Vorgaben für Web-Anwendungen, ihrer Erweiterung und Überprüfung. Da eine organisatorische Prüfung (z.B. durch den Web-Entwickler) fehleranfällig ist, sollen die Vorgaben automatisch gegen ein Modell der Web-Anwendung oder gegen Webseiten geprüft werden.

Die Definition der Vorgaben erfolgt mit Hilfe einer Web-Modellierungsmethode, wobei in dieser Arbeit die Web Modeling Language (WebML) zur Modellierung gewählt wird. Web-Modellierungsmethoden ermöglichen eine grafische Notation der Vorgaben und trennen die Modellierung nach Content, Hypertext und Präsentation. Sie können daher eingesetzt werden, um Vorgaben auf diesen drei Ebenen auszudrücken. Beispielsweise können sie festlegen, welche Information eine Webseite anzeigt oder wie der Besucher durch die Web-Anwendung navigieren kann.

Diese Arbeit betrachtet die Überprüfung der Vorgaben gegen ein Modell der Web-Anwendung und setzt daher voraus, dass die Web-Anwendung mit der verwendeten Modellierungsmethode modelliert wird. Die Überprüfung der Vorgaben basiert auf Constraints, welche mit Hilfe von Regeln automatisch aus den Vorgaben erzeugt werden. Um bestehende Inferenzmechanismen zur Überprüfung der Constraints gegen Modelle einsetzen zu können, werden die Modelle in einer Ontologie repräsentiert. Diese Arbeit verwendet das Resource Description Framework Schema (RDFS) als Ontologie- und die Protégé Axiom Language (PAL) als Constraint-sprache. Um die Mächtigkeit von Constraints voll auszuschöpfen, unterstützt diese Arbeit auch das Erweitern des Vorgabenmodells um Constraints, welche die Modellierungsmethode nicht ausdrücken kann.

Abstract

A wide range of organizations use web applications in order to deliver content and services to users and customers. The web applications of an organisation should provide the information in a standardised way to facilitate user navigation and content access. In order to achieve such a coherent appearance, decentralized organisations should not depend on a centralized administration of their web applications, but the organisational units should be able to manage their web applications independently. This thesis addresses the problem of ensuring a coherent appearance of web applications as well as guaranteeing flexibility. These ambitions are achieved by proposing mechanisms to define, extend and verify properties that are to be used for the web applications. As verifying the properties in an organisational manner (e.g. by the web designer) is error-prone, the organisational units should be able to automatically check whether their web application fulfils the required properties. The verification process can take place at design time against a web application model or at runtime against web pages.

In order to define the properties in a graphical format, this thesis describes web modelling methods and selects the Web Modeling Language (WebML) for this purpose. Web modelling methods separate the modelling of the content, the hypertext and the presentation of a web application and can therefore be used to express properties on these three levels. For example, properties can determine the actual web content or how users can navigate through the application.

The automatic verification of properties against a web application model is also described. It is assumed that the web application is modelled with the selected web modelling method. The verification process is based on constraints and utilizes existing inference engines that rely on ontology and constraint languages. This thesis uses the Resource Description Framework Schema (RDFS) as an ontology language and the Protégé Axiom Language (PAL) as a constraint language. Rules generate the constraints based on the modelled properties, and the web application model is automatically mapped onto an ontology. In order to fully exploit the power of the constraints, this thesis also supports extending the properties with further constraints, which cannot be captured by the employed modelling method alone.

Inhaltsverzeichnis

Abkürzungen

DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer
DAML-ONT	DARPA Agent Markup Language-Ontology
DL	Description Logics
DTD	Document Type Definition
F-Logic	Frame-Logic
HTML	Hypertext Markup Language
KIF	Knowledge Interchange Format
OCL	Object Constraint Language
OIL	Ontology Inference Layer
OO-H	Object-Oriented Hypermedia Method
OOHDM	Object-Oriented Hypermedia Design Model
OWL	Web Ontology Language
PAL	Protégé Axiom Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SGML	Standard Generalized Markup Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UWE	UML-based Web Engineering
W3C	Word Wide Web Consortium
WebML	Web Modeling Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Organisationen setzen Web-Anwendungen ein, um Inhalte und Dienste im Web bereitzustellen. Bei verteilten Organisationen, wie z.B. Universitätsinstituten oder virtuellen Unternehmen, werden die Web-Anwendungen entweder zentral von einer übergeordneten Stelle oder dezentral von den einzelnen Organisationseinheiten verwaltet. Im Falle einer zentralen Verwaltung sind die Web-Anwendungen für die gesamte Organisation einheitlich, und die Organisationseinheiten können Änderungen nur aufwendig über die Zentrale durchführen lassen. Erstellt hingegen jede Organisationseinheit ihre eigene Web-Anwendung, weisen die einzelnen Anwendungen oft große Unterschiede auf. Diese erschweren dem Benutzer das Auffinden von Information und erwecken den Eindruck, dass die Organisation keine Einheit bildet. Um die Bestrebung eines einheitlichen Erscheinungsbilds mit einer dezentralen Verwaltung in Einklang zu bringen, sollen verteilte Organisationen Vorgaben für Web-Anwendungen definieren, diese für Organisationseinheiten erweitern und ihre Einhaltung automatisch überprüfen können. Der Zweck solcher Vorgaben besteht darin, die Web-Anwendungen einer Organisation soweit zu vereinheitlichen, dass gleiche Information einheitlich dargestellt wird, ohne jedoch die Flexibilität einer dezentralen Verwaltung zu verlieren.

Die Einleitung ist folgendermaßen aufgebaut: Auf die Problemdefinition in Unterkapitel 1.1 folgt eine Beschreibung der bestehenden Ansätze in Unterkapitel 1.2. Die bestehenden Ansätze werden in Unterkapitel 1.3 mit dem Ansatz dieser Arbeit verglichen. Unterkapitel 1.4 erläutert die Vorgehensweise zur Definition und Überprüfung von Vorgaben, und Unterkapitel 1.5 stellt das Beispiel vor, welches diese Arbeit zur Veranschaulichung der Problemlösung verwendet. Abschließend wird in Unterkapitel 1.6 ein Überblick über den Aufbau der Diplomarbeit gegeben.

1.1 Problemdefinition

Aufgrund des Wachstums des World Wide Web steigen die Bedeutung und der Umfang von Web-Anwendungen. Web-Anwendungen erfüllen nicht nur den Zweck, Information zu präsentieren, sondern auch Dienste und Anwendungen bereitzustellen, auf welche der Benutzer mit einem Web-Browser zugreifen kann. Man spricht daher heute nicht mehr von Websites, sondern von Web-Anwendungen. Während eine Website aus einer Menge statischer Webseiten besteht, stellt eine Web-Anwendung ein komplexes Softwaresystem dar, welches die Webseiten dynamisch erzeugt und mit dem Benutzer interagieren kann. [Kapp04, S. 2] definiert eine Web-Anwendung folgendermaßen:

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

Web-Anwendungen verteilter Organisationen sollen bestimmte Vorgaben erfüllen. Entsprechend der Ebenen von Web-Anwendungen [Schw04] können sich solche Vorgaben auf den Content, den Hypertext oder die Präsentation beziehen. Der *Content* umfasst die Inhalte, welche die Web-Anwendung zur Verfügung stellt. Der Aufbau der Seiten sowie die Navigation zwischen den Seiten mittels Links sind Bestandteile des *Hypertexts*. Kennzeichnend für Web-Anwendungen und der wesentliche Unterschied zu traditionellen Softwaresystemen ist die Nicht-Linearität des Hypertexts. Diese ermöglicht die individuelle Navigation durch den Benutzer, d.h., dass sich der Benutzer beliebig in der Anwendung fortbewegen kann. Die *Präsentation* umfasst das Layout der Benutzerschnittstelle (z.B. die Platzierung von Elementen auf einer Seite), welches durch Ästhetik und Selbsterklärbarkeit gekennzeichnet sein soll.

Da Web-Anwendungen unterschiedliche Personen bzw. Organisationseinheiten involvieren, ist es schwierig, Vorgaben aufzustellen und ihre Einhaltung zu gewährleisten. Werden die Vorgaben zentral festgelegt, können die Organisationseinheiten die Vorgaben nicht verändern oder erweitern. Erstellt hingegen jede Organisationseinheit ihre eigene Web-Anwendung, sind die Vorgaben nicht explizit, sondern sie manifestieren sich in der Web-Anwendung. Dies hat zur Folge, dass die Vorgaben nicht einheitlich für die Organisation gelten. Für verteilte Organisationen ist daher ein Mittelweg zwischen einer zentralen und einer dezentralen Definition wünschenswert. Einerseits sollen Organisationen Vorgaben zentral festlegen können, andererseits sollen die Vorgaben für Organisationseinheiten erweiterbar sein, um eine gewisse Flexibilität aufrechtzuerhalten.

Die Durchsetzung der Vorgaben kann auf organisatorischer oder technischer Ebene erfolgen. Organisatorisch bedeutet, dass sich der Webentwickler an die Vorgaben halten muss. In diesem Fall ist es aufwendig festzustellen, ob die Web-Anwendung die Vorgaben tatsächlich erfüllt. Diese Arbeit beschäftigt sich mit einer Unterstützung auf technischer Ebene, d.h. mit der automatischen Überprüfung der Vorgaben. Die Überprüfung der Vorgaben kann hierbei statisch zum Zeitpunkt des Entwurfs oder dynamisch zur Laufzeit erfolgen. Bei einer Überprüfung zum Entwurfszeitpunkt werden die Vorgaben gegen ein Modell der Web-Anwendung überprüft. Dieser Ansatz geht davon aus, dass die Webseiten aus dem Modell generiert werden und daher die Vorgaben einhalten, sobald das Modell die Vorgaben erfüllt. Erfolgt die Überprüfung hingegen zur Laufzeit, werden die Vorgaben nicht gegen das Modell, sondern gegen die einzelnen Webseiten geprüft.

1.2 Bestehende Ansätze

Zur Definition und Überprüfung von Vorgaben für Web-Anwendungen sind bereits einige Ansätze vorhanden [Born97, Rous97, Harm99, Desp04, Alpu04, Fern99]. Diese Ansätze beschäftigen sich mit der Überprüfung von Vorgaben mittels Constraints, d.h., dass sie die Vorgaben in einer logikbasierten Sprache ausdrücken. Constraints, welche sich auf die Struktur bzw. den Hypertext der Web-Anwendung beziehen, werden vielfach als syntaktische Constraints bezeichnet. Constraints, welche den Content betreffen, werden semantische Constraints genannt. Die nachfolgend dargestellten Ansätze unterscheiden sich nicht nur in der Art der unterstützten Constraints, sondern auch dahingehend, ob die Constraints gegen ein Modell oder gegen konkrete Webseiten überprüft werden. Die Verifikation der Constraints basiert auf unterschiedlichen Techniken.

Die meisten Ansätze [Born97, Rous97, Harm99, Desp04, Alpu04] erlauben die Überprüfung von Constraints lediglich zur Laufzeit gegen konkrete Webseiten. Die Webseiten stehen entweder in einer Baumstruktur (z.B. HTML oder XML) zur Verfügung [Harm99, Desp04] oder werden als Fakten in eine logikbasierte Sprache übersetzt [Rous97, Alpu04].

[Born97] betrachtet Constraints hinsichtlich der Präsentation, um das Layout von Webseiten (z.B. die Platzierung von Elementen) und das Verhalten von Applets zu spezifizieren. Der Ansatz erlaubt das Bilden von Constraint-Hierarchien und das Gewichten der Constraints. Die Constraints können sowohl vom Designer der Webseiten als auch vom Betrachter der Webseite (z.B. Fenstergröße) stammen. Das endgültige Layout soll die Constraints bestmöglich erfüllen.

Der Ansatz von Rousset [Rous97] verwendet eine Sprache basierend auf logischen Regeln zur Definition semantischer Constraints. Rousset geht davon aus, dass die

Webseiten als Baumstruktur zur Verfügung stehen. Durch Übersetzung dieser Baumstruktur in Prädikate, welche in der verwendeten Sprache ausgedrückt werden, entsteht ein semantisches Modell. Dieses kann um zusätzliche Prädikate erweitert werden. Die Constraints werden ebenfalls in der verwendeten Sprache definiert. Ein Algorithmus prüft anschließend die Einhaltung der Constraints gegen das semantische Modell.

WebMaster [Harm99] definiert eine Sprache, um semantische Constraints auf XML Dokumenten zu repräsentieren. Die Constraints können beispielsweise das Vorhandensein von XML Tags oder den Inhalt von XML Elementen überprüfen. Eine Besonderheit dieses Ansatzes ist, dass die Constraints in einer grafischen Notation definiert werden können, welche automatisch in die verwendete Constraintsprache übersetzt wird. Zu diesem Ansatz ist ein Prototyp vorhanden.

[Desp04] überprüft die Constraints ebenfalls gegen XML Dokumente. Dieser Ansatz ermöglicht die Definition von syntaktischen und semantischen Constraints. Die Constraints werden in Form von Regeln spezifiziert, welche aus zwei Teilen bestehen: Der erste Teil überprüft die Existenz bzw. den Inhalt von XML Tags, der zweite Teil gibt Aktionen an, welche ausgeführt werden, wenn der erste Teil der Regel erfüllt ist. Aktionen können das Durchführen von Änderungen, das Ausgeben von Fehlermeldungen oder das Ableiten neuer Informationen auslösen. Zur Verifikation wird die natürliche Semantik, eine Technik aus dem Software Engineering, eingesetzt.

Der Ansatz von [Alpu04] verwendet eine Rewriting-Technik zur Überprüfung von Webseiten gegen Webspezifikationen. Die Webseiten sind HTML oder XML Dokumente, welche in eine Termalgebra übersetzt werden. Die Webspezifikation enthält die syntaktischen und semantischen Constraints und drückt die zu erfüllenden Bedingungen aus. VERDI ist ein Prototyp zu diesem Ansatz.

Im Gegensatz zu den bisher betrachteten Ansätzen geht [Fern99] nicht von konkreten Webseiten, sondern von einem Modell der Web-Anwendung aus. In diesem werden Struktur und Inhalt der Web-Anwendung, d.h. die Seiten, der Inhalt der Seiten und die Links zwischen den Seiten, deklarativ mit Fakten spezifiziert. Die Constraints werden mit Hilfe regulärer Pfadausdrücke definiert und drücken gewünschte Eigenschaften hinsichtlich der Struktur der Web-Anwendung aus. Der Algorithmus zur Verifikation der Constraints basiert auf Query Containment. Die Besonderheit dieses Ansatzes ist, dass die Constraints gegen das Modell der Web-Anwendung und nicht gegen Instanzen dieses Modells (Webseiten) geprüft werden. Die Webseiten erfüllen die Constraints, solange sie aus dem Modell generiert werden. Die Constraints müssen daher nicht jedes Mal überprüft werden, wenn sich der Inhalt der Webseiten ändert, sondern nur, wenn am Modell Änderungen durchgeführt werden.

1.3 Ziel

Das Ziel dieser Arbeit besteht darin, Vorgaben für Web-Anwendungen zu definieren und zu überprüfen. Wie bei den in Unterkapitel 1.2 vorgestellten Ansätzen basiert die Überprüfung der Vorgaben auf Constraints. Dies bedeutet, dass die Vorgaben in einer logikbasierten Sprache ausgedrückt werden, um sie überprüfen zu können.

Um eine einfache und verständliche Definition der Constraints zu ermöglichen, ist analog zu [Harm99] eine grafische Notation wünschenswert. Dieses Ziel wird durch den Einsatz einer Modellierungsmethode erreicht, mit welcher neben der Web-Anwendung auch die Vorgaben modelliert werden. Die Constraints werden anschließend automatisch aus dem Modell der Vorgaben generiert. Da die Modellierungsmethode meist nicht alle Vorgaben ausdrücken kann, sollen die Vorgabenmodelle erweiterbar sein.

Die in Unterkapitel 1.2 betrachteten Ansätze unterstützen Vorgaben auf unterschiedlichen Ebenen. Die meisten Ansätze behandeln Vorgaben bezüglich des Content [Rous97, Harm99, Desp04, Alpu04] und/oder des Hypertexts [Desp04, Alpu04, Fern99]. Lediglich [Born97] nimmt Bezug auf Präsentationsvorgaben. Der Ansatz dieser Arbeit unterstützt Vorgaben auf allen drei Ebenen. Eine konkrete Darstellung erfolgt für Vorgaben bezüglich Content und Hypertext. Die verwendeten Konzepte können jedoch auf Präsentationsvorgaben übertragen werden.

Die bestehenden Ansätze ermöglichen die Überprüfung der Vorgaben entweder gegen ein Modell der Web-Anwendung oder gegen konkrete Webseiten. Diese Arbeit unterstützt grundsätzlich beide Überprüfungszeitpunkte. Der Schwerpunkt liegt jedoch auf statischen Constraints, welche zum Zeitpunkt des Entwurfs gegen ein Modell der Web-Anwendung überprüft werden (vgl. [Fern99]). Eine Überprüfung zur Laufzeit würde zusätzliche Informationen (z.B. zum Zuordnen von Constraints zu Seiten) benötigen und eine spezielle Repräsentation der Seiten erfordern.

Die Überprüfung der Constraints erfolgt bei den bestehenden Ansätzen entweder gegen XML Dokumente oder gegen Fakten, in welche die Web-Anwendung übersetzt wird. In dieser Arbeit werden zur Repräsentation der Web-Anwendungsmodelle Ontologien eingesetzt (vgl. OntoWebber [Jin01]). Ontologien stellen explizite Konzeptualisierungen dar, welche semantische Konstrukte ausdrücken und daher das Ziehen von Schlussfolgerungen ermöglichen.

Die Verwendung von Ontologien hat den Vorteil, dass zur Überprüfung bestehende Inferenzmechanismen eingesetzt werden können und daher nicht eigene Algorithmen entwickelt werden müssen.

1.4 Vorgehensweise

Abbildung 1-1 zeigt den Ablauf der Definition und Überprüfung von Vorgaben. Die Vorgaben werden modelliert, in Constraints übersetzt, erweitert und gegen das Modell der Web-Anwendung überprüft.

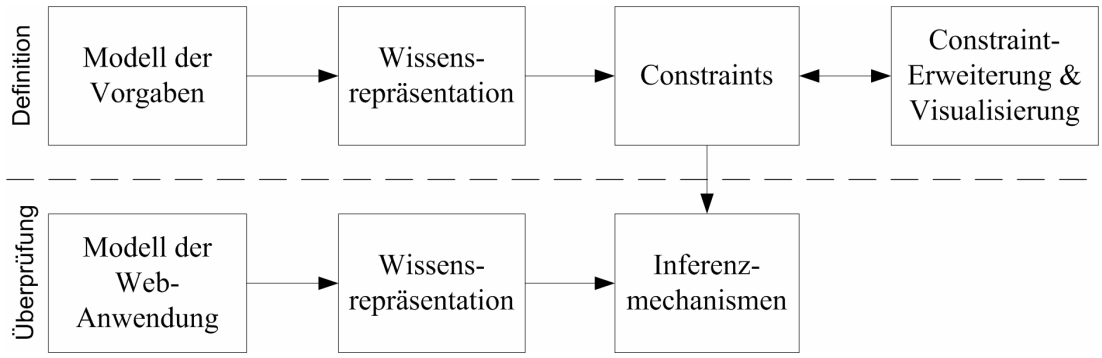


Abbildung 1-1 Vorgehensweise

Um die Vorgaben in einer grafischen Notation festlegen zu können, wird auf bestehende Modellierungsmethoden zurückgegriffen. Diese haben das Ziel, den Entwurf von Web-Anwendungen zu unterstützen, indem sie die relevanten Aspekte der zu erstellenden Anwendung in lesbarer Form darstellen. Beispiele für Modellierungsmethoden sind WebML [Ceri00], die objektorientierten Methoden OOHDM [Schw98], UWE [Koch02] und OO-H [Gome02] sowie die ontologiebasierten Methoden OntoWebber [Jin01] und OntoWeaver [Lei04]. Die Methoden trennen die Modellierung nach Content, Hypertext und Präsentation und können daher eingesetzt werden, um Vorgaben auf diesen drei Ebenen auszudrücken.

Um das Modell der Vorgaben gegen das Modell der Web-Anwendung überprüfen zu können, sind eine formale Repräsentation der Vorgaben und der Modelle sowie Inferenzmechanismen notwendig. Zur Repräsentation der Vorgaben werden Constraints eingesetzt. Inferenzmechanismen dienen einerseits der Generierung der Constraints aus dem Vorgabenmodell mit Hilfe von Regeln, andererseits der Überprüfung dieser Constraints gegen das Modell der Web-Anwendung. Damit Inferenzmechanismen Regeln bzw. Constraints gegen Modelle ausführen können, benötigen sie eine formale Repräsentation der Modelle, z.B. in Form einer Ontologie. Eine Ontologie ist eine zur Weiterverarbeitung durch den Computer geeignete Form der Wissensrepräsentation, welche die Semantik von Modellelementen und ihre Eigenschaften und Beziehungen festlegt. Auf diese Weise ermöglicht sie eine semantisch reichere Modellierung von Konzepten, Eigenschaften und Beziehungen [Will03].

Die Definition der Vorgaben erfolgt durch Modellierung der Vorgaben mit einer Modellierungsmethode. Die Modelle werden in einer Wissensrepräsentationssprache

ausgedrückt, damit Inferenzmechanismen Regeln auf die Modelle ausführen und die Constraints generieren können. Zusätzliche Vorgaben, welche die Modelle nicht festlegen, können in Form von Modellerweiterungen oder Constraints ergänzt werden. Außerdem wird die Visualisierung der Abhängigkeiten zwischen den Constraints unterstützt. Zur Überprüfung wird die Web-Anwendung ebenfalls mit der gewählten Modellierungsmethode modelliert und das Modell in einer Wissensrepräsentationssprache ausgedrückt. Inferenzmechanismen können anschließend die Constraints gegen das Modell überprüfen und auf diese Weise feststellen, welche Vorgaben die Web-Anwendung erfüllt bzw. nicht erfüllt.

1.5 Beispiel

In einer Universität besitzt jedes Universitätsinstitut eine eigene Web-Anwendung, um Informationen über Mitarbeiter, Lehrveranstaltungen, Projekte etc. zur Verfügung zu stellen. Aufgrund einer dezentralen Verwaltung sind die einzelnen Web-Anwendungen oft sehr unterschiedlich. Beispielsweise listet die Mitarbeiterseite des Instituts A neben den persönlichen Daten über den Mitarbeiter seine Lehrveranstaltungen auf. Institut B gibt zu jedem Mitarbeiter die Projekte an, welche der Mitarbeiter leitet bzw. in welchen er arbeitet. Auf den Mitarbeiterseiten des Instituts B sind jedoch keine Informationen über Lehrveranstaltungen verfügbar.

Um dem Besucher der Web-Anwendung das Auffinden der gewünschten Information zu erleichtern, setzt die Universität den Instituten Vorgaben für ihre Web-Anwendungen. Vorgaben über den Content beziehen sich hierbei auf die Daten, welche abrufbar sind:

- Daten über Mitarbeiter (Familienname, Vorname, Telefonnummer, Email-Adresse).
- Daten über Lehrveranstaltungen (Name, Wochenstunden).
- Lehrveranstaltungen eines Mitarbeiters.

Bezogen auf den Hypertext sollen die Web-Anwendungen folgende Vorgaben erfüllen:

- Jeder Mitarbeiter besitzt eine Mitarbeiterseite mit Informationen über den Mitarbeiter und über seine Lehrveranstaltungen.
- Die Mitarbeiterseite ist erreichbar, indem der gewünschte Mitarbeiter aus einer Liste der Mitarbeiter ausgewählt wird.

Um eine gewisse Flexibilität zu erhalten, sollen die Vorgaben erweiterbar sein. Das Rektorat setzt den Fakultäten Vorgaben, welche die Fakultäten erfüllen müssen, jedoch für ihre Institute erweitern können. Zum Beispiel kann das Rektorat festlegen, dass jedes Institut in seiner Web-Anwendung alle Mitarbeiter auflistet und diese

Liste zu den Details jedes Mitarbeiters verlinkt. Die Fakultät gibt den Inhalt der Mitarbeiterseite an. Zu jedem Mitarbeiter sollen Name, Telefonnummer und Lehrveranstaltungen angezeigt werden.

Die Vorgaben werden im Folgenden gegen eine Institutsanwendung geprüft, welche auf der Mitarbeiterseite auch die Projekte des Mitarbeiters auflistet. Da Web-Anwendungen Daten meist nicht nur anzeigen, sondern auch das Bearbeiten von Daten erlauben, wird die Änderung von Projektdaten miteinbezogen. Die Überprüfung kann zum Zeitpunkt der Modellierung oder zur Laufzeit erfolgen. Bei einer Überprüfung gegen ein Modell der Web-Anwendung wird beispielsweise geprüft, ob das Web-Anwendungsmodell des Instituts festlegt, dass auf jeder Mitarbeiterseite Lehrveranstaltungen angezeigt werden. Diese Vorgabe gilt für alle Mitarbeiterseiten und muss daher nicht für jede einzelne Seite definiert und überprüft werden. Ein Beispiel für eine dynamische Überprüfung zur Laufzeit ist die Vorgabe, dass die Mitarbeiterseite Huber zusätzlich einen Link zur persönlichen Homepage des Mitarbeiters enthält. Diese Vorgabe gilt nur für eine Mitarbeiterseite und kann daher erst zur Laufzeit, wenn die Webseite generiert wurde, überprüft werden.

Das Beispiel der Mitarbeiterseiten dient als durchgängiges Beispiel der Diplomarbeit und wird im Anhang A.5 im Gesamtzusammenhang dargestellt.

1.6 Aufbau der Diplomarbeit

Kapitel 2 vergleicht Modellierungsmethoden und wählt eine dieser Methoden zur Modellierung der Vorgaben und der Web-Anwendung aus. Die Überprüfung der Vorgaben gegen Web-Anwendungen mit Hilfe von Inferenzmechanismen wird in Kapitel 3 behandelt. Außerdem erläutert dieses Kapitel die Notwendigkeit von Ontologiesprachen zur Repräsentation der Modelle und vergleicht Ontologiesprachen sowie Werkzeuge zur Erstellung von Ontologien und zur Durchführung von Inferenzen. Kapitel 4 geht auf die praktische Umsetzung der Aufgabenstellung mit den entsprechenden Technologien und Werkzeugen ein und behandelt das Übersetzen der Modelle in die gewählte Ontologiesprache, das Erzeugen der Constraints aus Vorgabenmodellen mit Hilfe von Regeln und das Überprüfen der Constraints. Außerdem wird erläutert, wie Vorgaben erweitert und Abhängigkeiten zwischen Constraints visualisiert werden können. Kapitel 5 fasst die Ergebnisse der Diplomarbeit zusammen und gibt einen Ausblick auf mögliche zukünftige Entwicklungen. Details der Implementierung sowie ein zusammenfassendes Beispiel werden im Anhang dargestellt.

Kapitel 2

Modellierung von Web-Anwendungen

Aufgrund der steigenden Komplexität von Web-Anwendungen spielen Modelle bei der Entwicklung dieser Anwendungen eine bedeutende Rolle. Traditionelle Methoden des Software Engineering wie Entity-Relationship Modelle [Chen76] oder die Unified Modeling Language (UML) [Booc99] sind für den Entwurf von Web-Anwendungen nicht ausreichend, da diese lediglich die Modellierung des Content berücksichtigen. Sie gehen jedoch nicht auf die speziellen Anforderungen von Web-Anwendungen wie Aspekte des Hypertexts und der Präsentation ein. Die Modellierung von Web-Anwendungen erfordert daher die Erweiterung traditioneller Methoden um Modelle zur Abbildung des Hypertexts und der Präsentation.

Unterkapitel 2.1 beschreibt die Modelle für den Entwurf von Web-Anwendungen. In Unterkapitel 2.2 wird ein Überblick über Modellierungsmethoden für Web-Anwendungen gegeben. Das Content- und Hypertextmodell einiger dieser Methoden werden näher betrachtet. Auf das Präsentationsmodell wird nicht im Detail eingegangen, da bestehende Ansätze wie XSLT [W3C01b] keine ausreichenden Modellierungskonstrukte bereitstellen. Abschließend folgt eine Gegenüberstellung der gewählten Methoden.

2.1 Modelle

Die Modellierung einer Web-Anwendung basiert auf der Trennung von Content, Hypertext und Präsentation, um den Entwurf von Web-Anwendungen zu vereinfachen. Die Modelle, welche durch diese Trennung entstehen, werden nachfolgend gemäß [Schw04] beschrieben.

2.1.1 Content-Modell

Das Content-Modell beschreibt die Informationsstruktur des Problembereichs, d.h. die Inhalte, welche eine Web-Anwendung zur Verfügung stellt. Die Festlegung der Struktur hat den Vorteil, dass die Struktur unabhängig von den Inhalten ist und auch dann unverändert bleibt, wenn sich der Inhalt häufig ändert. Statt Content-Modell werden auch die Bezeichnungen Strukturmodell [Ceri00], konzeptuelles Modell [Schw98, Koch02], Klassendiagramm [Gome02] oder Domain-Modell [Jin01, Lei04] verwendet.

Zur Modellierung des Content greifen einige Web-Modellierungsmethoden [Ceri00, Schw98, Koch02, Gome02] auf traditionelle Methoden wie Entity-Relationship Modelle [Chen76] oder Klassendiagramme (z.B. Unified Modeling Language (UML) [Booc99]) zurück. Um zusätzlich zur Struktur auch die Semantik der Daten zu spezifizieren, verwenden andere Modellierungsmethoden Ontologien (vgl. [Jin01, Lei04]). Zur Modellierung können außerdem Frames [Lass01] eingesetzt werden, welche Schlussfolgerungen unterstützen. Gemeinsam ist diesen Ansätzen die Zusammenfassung gleichartiger Objekte zu Klassen bzw. Konzepten, welche dieselben Attribute und Beziehungen zu anderen Klassen teilen.

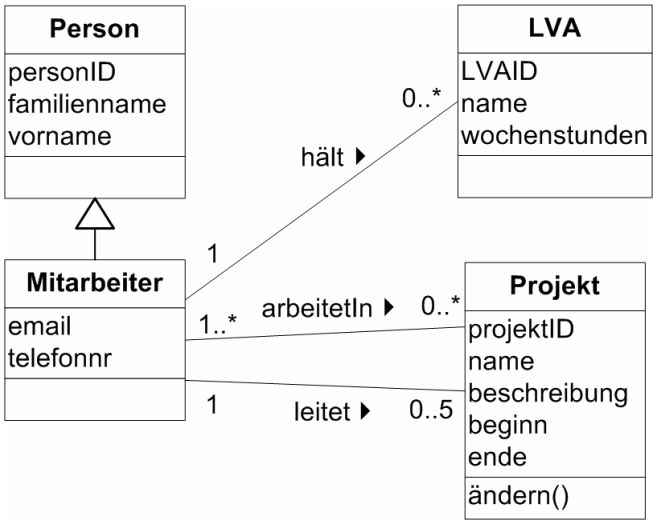


Abbildung 2-1 Content-Modell

Beispiel: *Abbildung 2-1 zeigt das Content-Modell der Institutsanwendung in UML Notation. Das Modell legt fest, dass Daten über Mitarbeiter (Familienname, Vorname, Email-Adresse, Telefonnummer), über Lehrveranstaltungen (Name, Wochenstunden) und über Projekte (Name, Beschreibung, Beginn, Ende) angezeigt werden. Zwischen Mitarbeiter und Lehrveranstaltung besteht die Beziehung „hält", welche angibt, dass Mitarbeiter Lehrveranstaltungen abhalten. Die Beziehungen zwischen Mitarbeiter und Projekt drücken aus, dass Mitarbeiter in Projekten arbeiten bzw. dass jedes Projekt einen Projektleiter besitzt. Mitarbeiter sind spezielle Personen und erben daher die Attribute der Person.*

2.1.2 Hypertext-Modell

Das Hypertext-Modell spezifiziert, wie Benutzer durch den Inhalt einer Web-Anwendung navigieren und auf die Informationen zugreifen können. Die Trennung von Struktur- und Hypertextmodell ermöglicht die Definition unterschiedlicher Sichten (auch „Siteviews“ genannt) auf dieselben Daten, welche für verschiedene Benutzergruppen oder Endgeräte angepasst sind. Auf diese Weise können z.B. Benutzerrechte oder Datensichten realisiert werden. [Schw98, Koch02, Gome02] nennen dieses Modell Navigationsmodell, [Jin01, Lei04] verwenden die Bezeichnung Siteview-Modell.

Ausgehend vom Content-Modell wird zuerst die Struktur des Hypertexts mittels Knoten und Links definiert. Die Information aus dem Content-Modell kann hierbei redundant auf mehrere Knoten aufgeteilt werden, um diese Information über mehrere Pfade erreichbar zu machen. Anschließend wird das Strukturmodell verfeinert, damit die Information für den Benutzer leichter auffindbar wird. Das Hypertext-Modell wird in einer auf UML basierenden (vgl. [Schw98, Koch02, Gome02]) oder in einer methodenspezifischen Notation (vgl. [Ceri00, Jin01]) definiert.

Beispiel: *Die Institutsanwendung besteht aus den Knoten Mitarbeiter, Lehrveranstaltungen und Projekte, zwischen denen Links definiert sind. Um beispielsweise auf Mitarbeiterdaten zugreifen zu können, wird die Struktur mit einem Index über die Mitarbeiter erweitert. Dieser legt fest, dass der Benutzer den gewünschten Mitarbeiter aus einer Liste auswählen kann, um auf dessen Detailseite zu navigieren. Die Konstrukte zur Änderung der Projektdaten werden ebenfalls im Hypertext-Modell definiert. Darf nur der Projektleiter die Projektdaten ändern, ist hierfür eine eigene Sicht notwendig. Zur Vereinfachung wird jedoch nur eine Sicht definiert.*

Die Notation des Hypertextmodells unterscheidet sich bei den betrachteten Modellierungsmethoden, weshalb die jeweilige Abbildung dieses Modells erst bei der Beschreibung der Methoden betrachtet wird (vgl. Unterkapitel 2.2).

2.1.3 Präsentationsmodell

Das Präsentationsmodell legt die Struktur und das Verhalten der Benutzerschnittstelle fest. Es können mehrere Benutzerschnittstellen für dasselbe Hypertext-Modell definiert werden.

Zur Definition der Präsentation greifen die Methoden auf unterschiedliche Konzepte zurück. In [Koch02] werden UML Klassendiagramme und Stereotype eingesetzt.

Beispiel: *Abbildung 2-2 legt das Layout der Mitarbeiterseiten fest.*

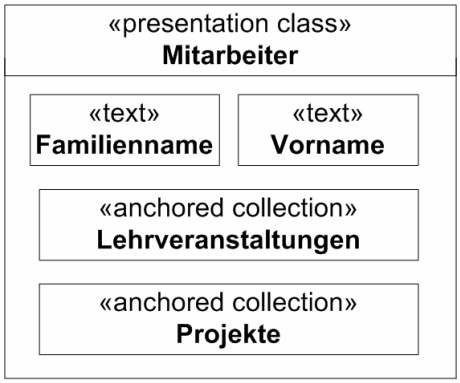


Abbildung 2-2 Präsentationsmodell

Mangels hinreichend mächtiger Modelle sind die Ansätze zur Modellierung der Präsentation sehr technologiezentriert und basieren z.B. auf XSL [W3C01b] (vgl. [Ceri00]). Da aus diesem Grund eine Modellierung der Präsentation nur eingeschränkt möglich ist, wird dieses Modell nicht näher betrachtet.

2.2 Modellierungsmethoden

Es existiert eine große Anzahl unterschiedlicher Methoden für die Modellierung von Web-Anwendungen. Aufgrund der besonderen Anforderungen an Web-Anwendungen müssen traditionelle Software Engineering Methoden erweitert werden. Neben einer Modellierungssprache definieren die meisten Methoden auch ein Vorgehensmodell und werden von einem Werkzeug unterstützt.

[Schw04, S. 70f] unterscheidet folgende Paradigmen:

- *Datenorientierte Methoden:* Erweitern das Entity-Relationship Modell um Konzepte zur Modellierung des Hypertexts. Sie eignen sich vor allem für die Modellierung von datenintensiven Web-Anwendungen. Ein Beispiel dieser Methoden ist die Web Modeling Language (WebML) [Ceri00].
- *Hypertext-orientierte Methoden:* Konzentrieren sich auf die Modellierung des Hypertexts. Beispiele sind das Hypertext Design Model (HDM) [Garz95] sowie Weiterentwicklungen von HDM wie W2000 [Bare00] und HDM-lite [Frat98].
- *Objektorientierte Methoden:* Erweitern objektorientierte Modellierungssprachen wie UML. Beispiele sind das Object-Oriented Hypermedia Design Model (OOHDM) [Schw98], UML-based Web Engineering (UWE) [Koch02] und die Object-Oriented Hypermedia Method (OO-H) [Gome02].
- *Softwareorientierte Methoden:* Verwenden Techniken aus dem klassischen Software Engineering. Hierzu zählt z.B. Web Application Extension (WAE) [Cona03].

Durch die Entwicklungen im Bereich des Semantic Web [Bern01] kann diese Aufzählung um ontologiebasierte Methoden erweitert werden, welche neben der Struktur auch die Semantik von Modellelementen spezifizieren und Schlussfolgerungen über Modelle ermöglichen [Will03]. Beispiele für diese Methoden sind OntoWebber [Jin01] und OntoWeaver [Lei04].

Im Folgenden werden einige Methoden näher betrachtet, wobei der Schwerpunkt auf neueren Ansätzen liegt. Ausgewählt werden WebML, die objektorientierten Methoden UWE und OO-H sowie die ontologiebasierten Methoden OntoWebber und OntoWeaver. UWE und OO-H sind Weiterentwicklungen von OOHDM, weshalb diese Methode ebenfalls kurz beschrieben wird.

2.2.1 WebML

WebML [Ceri00, Ceri02, WebR02] steht für Web Modeling Language und ist eine konzeptuelle Modellierungssprache für den Entwurf datenintensiver Web-Anwendungen. Hinsichtlich des Strukturmodells unterstützt WebML das Entity-Relationship Modell und UML. Für die Hypertext-Modellierung wird eine eigene Notation verwendet. WebML definiert kein explizites Präsentationsmodell.

2.2.1.1 Entwurfsprozess

Der Entwurfsprozess mit WebML gliedert sich in mehrere Schritte, welche unterschiedliche Modelle verwenden. Aufbauend auf die Anforderungen wird die Informationsstruktur in einem Strukturmodell festgelegt und der Hypertext entworfen. Das Hypertextmodell besteht aus dem Kompositionsmodell, welches den Seitenaufbau beschreibt, und dem Navigationsmodell, welches die Navigation mit Hilfe von Links definiert. Schließlich wird mit Hilfe von XSL-Templates die Präsentation der Applikation festgelegt.

2.2.1.2 Strukturmodell

Die Modellierung der Informationsstruktur basiert auf dem Entity-Relationship Modell und ist mit UML Klassendiagrammen kompatibel. Die elementaren Bausteine des Strukturmodells sind Entitäten und Beziehungen zwischen Entitäten. Beziehungen können Kardinalitätseinschränkungen sowie Rollennamen aufweisen. Entitäten besitzen benannte Eigenschaften, so genannte Attribute. Instanzen einer Entität werden über eine OID (Object ID) eindeutig identifiziert und angesprochen. Im Gegensatz zu UML unterstützt WebML lediglich binäre Beziehungen zwischen Entitäten, und es können keine Attribute zu Beziehungen definiert werden.

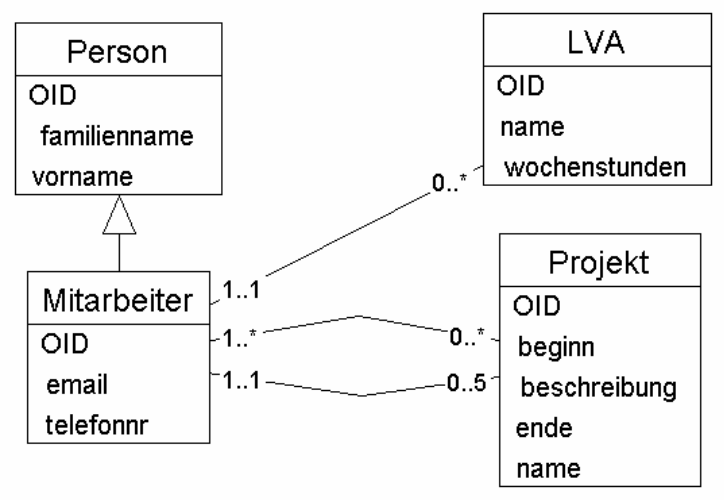


Abbildung 2-3 WebML Strukturmodell

In WebML ist es außerdem möglich, redundante Information aus bestehender Information abzuleiten. Diese so genannte Derivation wird über Abfragen ausgedrückt und ermöglicht das Hinzufügen von Konzepten, welche nicht explizit gespeichert, sondern aus anderen Komponenten berechnet werden. Ein Beispiel ist die Berechnung des Alters aus dem Geburtsdatum.

2.2.1.3 Hypertextmodell

Das Hypertextmodell beschreibt einen oder mehrere Hypertexts einer Web-Anwendung. Jeder Hypertext definiert eine so genannte „Siteview“, welche eine Sicht auf die Web-Anwendung darstellt. Die Beschreibung einer Siteview besteht aus zwei Submodellen, dem Kompositions- und dem Navigationsmodell.

Mit Hilfe des *Kompositionsmodells* wird die eigentliche Seitenstruktur festgelegt. Dieses Modell spezifiziert die Seiten, aus welchen der Hypertext besteht, und die so genannten „Content Units“, aus welchen sich eine Seite zusammensetzt. Eine Seite kann mehrere Seiten zusammenfassen.

Die sieben Arten vordefinierter Content Units sind:

- *Data Unit:* Repräsentiert eine Instanz einer Entität (z.B. einen Mitarbeiter).
- *Multidata Unit:* Präsentiert mehrere Instanzen einer Entität (z.B. Lehrveranstaltungen eines Mitarbeiters).
- *Index Unit:* Zeigt mehrere Instanzen einer Entität als Liste an, von welchen eine ausgewählt werden kann (z.B. Index über Mitarbeiter).
- *Multichoice Index Unit:* Wie Index Unit, jedoch können mittels Checkboxen mehrere Instanzen ausgewählt werden.
- *Hierarchical Index Unit:* Organisiert die Indexeinträge als mehrstufigen Baum.

- *Scroller Unit:* Stellt Kommandos zum Durchlaufen einer Menge von Instanzen einer Entität zur Verfügung.
- *Entry Unit:* Repräsentiert ein Eingabeformular.

Mit Ausnahme der Entry Unit zeigen Content Units eine oder mehrere Instanzen von Entitäten aus dem Strukturmodell an. Sie werden mit einer Entität verbunden und optional mit einem Selektor verknüpft. Selektoren geben Bedingungen an, um jene Instanzen einer Entität auszuwählen, welche angezeigt werden sollen.

Das *Navigationsmodell* spezifiziert Links, welche Content Units innerhalb einer Seite, Content Units verschiedener Seiten oder Seiten verknüpfen. Sie drücken die Navigation innerhalb der Web-Anwendung sowie den Transfer von Information von einer Unit zu einer anderen aus.

WebML unterstützt nicht nur die Anzeige von Information, sondern auch deren Manipulation. So genannte Operation Units rufen externe Operationen auf, um den Inhalt einer Web-Anwendung zu verwalten. Operationen des Datenmanagements dienen der Verwaltung von Entitäten (Erzeugen, Ändern, Löschen) und von Beziehungen (Hinzufügen, Löschen). Vordefinierte Operationen zur Zugriffskontrolle (Login, Logout, Ändern der Benutzergruppe) erleichtern die Personalisierung einer Web-Anwendung.

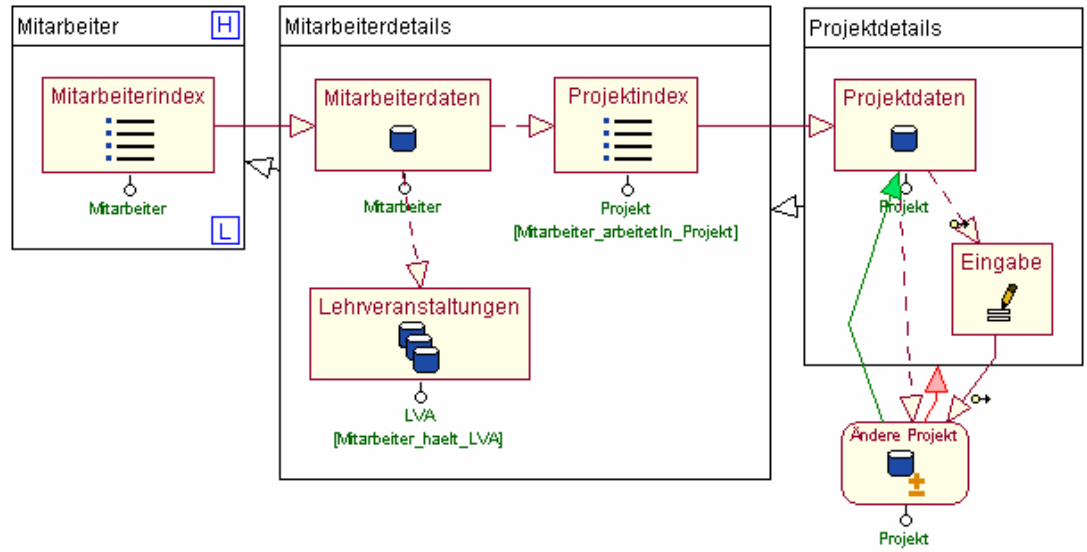


Abbildung 2-4 WebML Hypertextmodell

Beispiel: *Abbildung 2-4 definiert die Seite „Mitarbeiter“, welche alle Mitarbeiter über eine Index Unit auflistet. Durch Auswahl eines Mitarbeiters gelangt der Benutzer auf die Seite „Mitarbeiterdetails“ mit Mitarbeiterdaten (Data Unit), seinen Lehrveranstaltungen (Multidata Unit) und Projekten (Index Unit). Details eines Projekts werden auf der Seite „Projektdetails“ angezeigt. Über eine Entry Unit und eine Modify Unit können auf dieser Seite die Projektdaten geändert werden.*

2.2.1.4 Werkzeug

WebRatio (http://www.webratio.com) ist ein Modellierungswerkzeug, welches auf WebML aufbaut. Die Modelle können mit Hilfe vordefinierter Regeln auf ihre Gültigkeit überprüft werden. Das Tool EasyStyle generiert XSL-Stylesheets zur Festlegung der Präsentation und zur Transformation der Seiten. WebRatio unterstützt die automatische Generierung der Webseiten aus der XML Repräsentation der Modelle. Die generierte Web-Anwendung läuft in Frameworks wie J2EE [Bodo04] oder .NET [Beer02].

2.2.2 OOHDM

OOHDM (Object-Oriented Hypermedia Design Method) [Schw98] ist eine objekt-orientierte Methode für den Entwurf von Web-Anwendungen, welche auf objekt-orientierten Modellierungssprachen traditioneller Methoden basiert.

2.2.2.1 Entwurfsprozess

Der Entwurfsprozess besteht aus vier Aktivitäten, welche iterativ und inkrementell durchgeführt werden. Diese Aktivitäten umfassen die konzeptuelle Modellierung, die Navigationsmodellierung, die Modellierung der Benutzerschnittstelle und die Implementierung.

2.2.2.2 Konzeptuelles Modell

Das konzeptuelle Modell beschreibt die Informationsstruktur des Problembereichs mit Hilfe von UML Klassendiagrammen (vgl. Abbildung 2-1).

2.2.2.3 Navigationsmodell

Ein Navigationsmodell ist eine Sicht über das konzeptuelle Modell und besteht aus zwei Schemata, dem Klassen- und dem Kontextschema. Entsprechend der Benutzerbedürfnisse können mehrere Navigationsmodelle erstellt werden.

Im *Klassenschema* werden navigierbare Objekte (Navigationsklassen) definiert. Diese müssen nicht den Objekten des konzeptuellen Schemas entsprechen, sondern können aus einer Kombination von Attributen verschiedener Klassen des konzeptuellen Schemas bestehen. Links drücken Navigationspfade zwischen den Klassen aus.

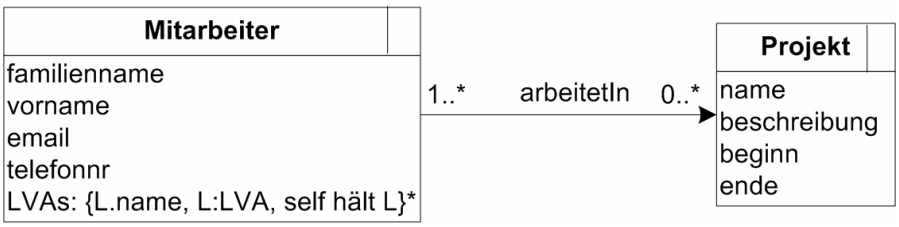


Abbildung 2-5 OOHDM Navigation Class Schema

Beispiel: *Das Klassenschema in Abbildung 2-5 definiert, dass zu einem Mitarbeiter zusätzlich zu Name, Email-Adresse und Telefonnummer auch die Namen der Lehrveranstaltungen angezeigt werden. Ausgehend von einem Mitarbeiter sind die Projekte erreichbar, in welchen der Mitarbeiter arbeitet.*

Das *Kontextschema* fasst Navigationsobjekte zu Navigationskontexten zusammen. Zugriffsstrukturen (z.B. Indices) geben an, wie auf Knoten zugegriffen werden kann.

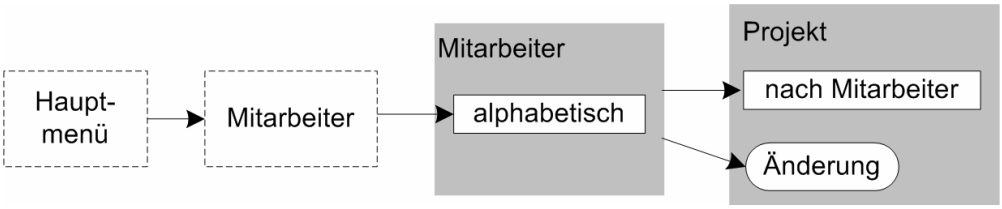


Abbildung 2-6 OOHDM Navigation Context Schema

Beispiel: *Abbildung 2-6 zeigt das Kontextschema zur Auswahl und Ansicht eines Mitarbeiters. Der Index „Mitarbeiter“ listet alle Institutsmitarbeiter alphabetisch sortiert auf. Wählt der Benutzer einen Mitarbeiter aus, gelangt er zum Knoten „Mitarbeiter“, welcher die Daten über den Mitarbeiter und die Namen seiner Lehrveranstaltungen anzeigt. Dieser Knoten besitzt außerdem einen Link auf den Knoten „Projekt“, welcher die Projektdaten beinhaltet und die Änderung dieser Daten erlaubt.*

2.2.2.4 Werkzeug

Die Modellierung mit OOHDM wird nicht durch Werkzeuge unterstützt.

2.2.3 UWE

UML-based Web Engineering (UWE) [Koch01, Koch02] ist ein objektorientierter Ansatz zur Modellierung von Web-Anwendungen, welcher vollständig auf UML basiert. UWE profitiert hierbei von der weiten Verbreitung und Akzeptanz von UML. Die Modellierung der Web-Anwendung erfolgt anhand mehrerer UML Modelle. Da reines UML nicht alle Anforderungen von Web-Anwendungen erfüllen kann, werden Stereotype zur Modellierung der Navigation und der Präsentation verwendet.

2.2.3.1 Entwurfsprozess

UWE ist ein objektorientierter, iterativer und inkrementeller Ansatz, welcher auf dem Unified Software Development Process [Booc99] basiert.

Mit Hilfe von UML Anwendungsfalldiagrammen werden die Anforderungen an die Anwendung beschrieben. UML Klassendiagramme definieren aufbauend auf die Anforderungen die statischen Aspekte der Web-Anwendung. Hierzu zählt der

Entwurf des konzeptuellen Modells sowie des Navigations- und des Präsentationsmodells. Anschließend werden die dynamischen Aspekte wie Aufgaben und Szenarien mit Hilfe der dynamischen Diagramme von UML modelliert. Zu diesen Diagrammen gehören Zustands- und Interaktionsdiagramme zur Modellierung von Webszenarien und Aktivitätsdiagramme zur Modellierung von Aufgaben.

Auf das konzeptuelle Modell und das Navigationsmodell wird im Folgenden näher eingegangen.

2.2.3.2 Konzeptuelles Modell

Das konzeptuelle Modell beschreibt die Objekte des Problembereichs mit Hilfe von UML Klassendiagrammen (vgl. Abbildung 2-1).

2.2.3.3 Navigationsmodell

Das Navigationsmodell besteht aus zwei Submodellen, zu deren Modellierung UML Klassendiagramme und Stereotype eingesetzt werden.

Das *Navigationsraummodell* („navigation space model“) definiert, welche Objekte bei der Navigation durch die Anwendung besucht bzw. erreicht werden können. Die zwei Elemente dieses Modells sind Klassen und Assoziationen. Es werden nur jene Klassen aus dem konzeptuellen Modell aufgenommen, welche für die Navigation relevant sind. Diese Navigationsklassen entsprechen Seiten oder Knoten. Assoziationen drücken Links und daher die Navigabilität aus.

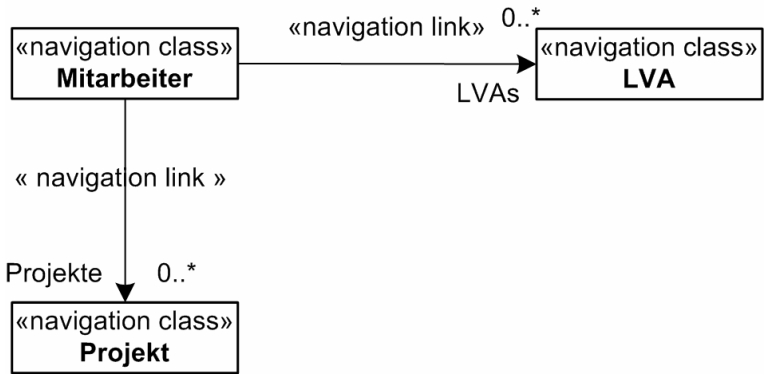


Abbildung 2-7 UWE Navigation Space Model

Beispiel: *Abbildung 2-7 definiert, dass Daten über Mitarbeiter, Projekte und Lehrveranstaltungen angezeigt werden. Projekte und Lehrveranstaltungen sind über den Mitarbeiter erreichbar.*

Das *Navigationsstrukturmodell* („navigation structure model“) verfeinert das Navigationsraummodell mit Hilfe von Zugriffsstrukturen, welche mit Stereotypen repräsentiert werden.

Die vier vordefinierten Zugriffsstrukturen sind:

- *Index:* Erlaubt die Auswahl einer Instanz einer Klasse.
- *Guided-Tour:* Dient dem sequentiellen Durchlaufen einer Menge von Knoten.
- *Query:* Zur Suche nach Knoten.
- *Menü:* Ist ein Index über eine Menge unterschiedlicher Elemente.

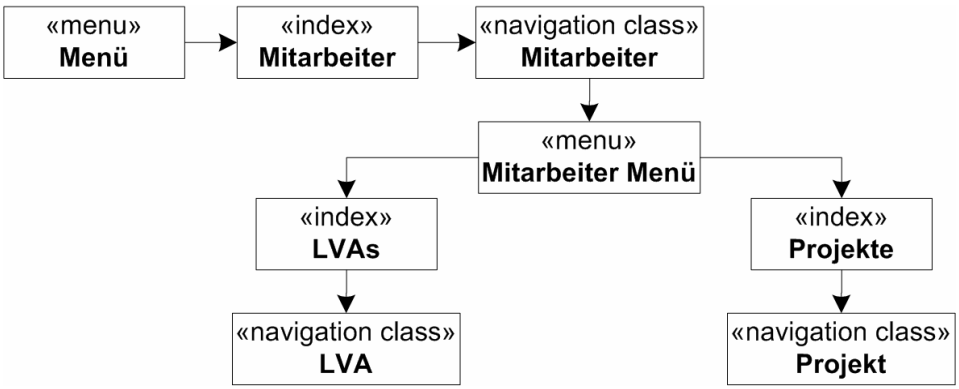


Abbildung 2-8 UWE Navigation Structure Model

Beispiel: *Abbildung 2-8 zeigt das Navigationsstrukturmodell mit den Zugriffsstrukturen „menu" und „index" zur Auswahl von Mitarbeitern, Lehrveranstaltungen und Projekten. Die Änderung der Projektdaten kann in einem UML Aktivitätsdiagramm modelliert werden.*

2.2.3.4 Werkzeug

OpenUWE (http://www.pst.informatik.uni-muenchen.de/projekte/openuwe) ist eine Entwicklungsumgebung für den Entwurf und die Generierung von Web-Anwendungen basierend auf UWE. Die Entwicklungsumgebung besteht aus den Werkzeugen ArgoUWE und UWEXML.

ArgoUWE ist ein Werkzeug für den Entwurf von Web-Anwendungen basierend auf UWE. Neben dem Entwurf der Modelle wird auch die Überprüfung der Modelle mit Hilfe von OCL Constraints unterstützt. Bestimmte Modelle können semi-automatisch generiert werden, z.B. kann das Navigationsmodell aus dem konzeptuellen Modell und das Präsentationsmodell aus dem Navigationsmodell generiert werden.

UWEXML unterstützt die Generierung der Web-Anwendung aus den Entwurfsmodellen, indem es die Entwurfsmodelle in XML Dokumente übersetzt. Ein Codegenerator erzeugt die Implementierung der Web-Anwendung. Die physischen Eigenschaften der Benutzerschnittstelle (z.B. Farbe und Schriftart) werden mit Hilfe eines Layouteditors definiert.

2.2.4 OO-H

OO-H (Object-Oriented Hypermedia Method) [Gome01, Gome02] ist eine objektorientierte Methode zur Modellierung und Implementierung von Web-Anwendungen. Sie definiert eine Menge von Diagrammen, Techniken und Werkzeugen für die Modellierung von Web-Anwendungen und erweitert UML, um den speziellen Anforderungen von Web-Anwendungen gerecht zu werden. Die Verwendung von UML hat den Vorteil, dass auf bekannte und bewährte Notationen und Techniken zurückgegriffen wird. OO-H definiert als Erweiterung zu UML das Navigations- und das Präsentationsmodell.

2.2.4.1 Entwurfsprozess

Kennzeichnend für den Entwurfsprozess ist eine iterative und inkrementelle Vorgehensweise. Der Entwurf der Anwendung wird in mehreren Iterationen verfeinert. Zuerst werden die funktionalen Anforderungen mit Anwendungsfall-diagrammen beschrieben und das Klassendiagramm erstellt. Anschließend werden die Anwendungsfälle gruppiert. Aufbauend auf diese Gruppen erfolgt die Spezifikation der Navigation und der Präsentation.

2.2.4.2 Klassendiagramm

Das Klassendiagramm erfasst die Informationsstruktur des Problembereichs mit Klassen und Beziehungen zwischen den Klassen und entspricht dem UML Klassendiagramm (vgl.Abbildung 2-1).

2.2.4.3 Navigationsmodell

Das Navigationsmodell baut auf dem Anwendungsfall- und dem Klassendiagramm auf und besteht aus ein oder mehreren Navigation Access Diagrammen (NAD). Jedes NAD stellt eine Sicht auf das System dar und ist spezifisch für eine bestimmte Benutzergruppe. Zur Konstruktion eines NADs werden zuerst die Anwendungsfälle, welche dasselbe Ziel („navigation target“) verfolgen, zu Gruppen zusammengefasst.

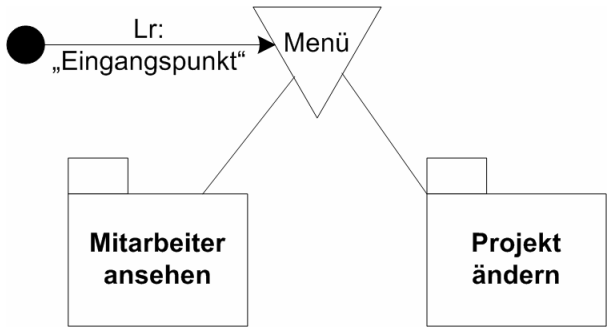


Abbildung 2-9 OO-H NAD Level 0

Beispiel: *Abbildung 2-9 definiert die zwei Navigationsziele „Mitarbeiter ansehen“ und „Projekt ändern“.*

Anschließend wird die interne Navigation jeder Gruppe detailliert beschrieben, indem das Klassendiagramm um Navigationsaspekte erweitert wird (vgl. Abbildung 2-10 und Abbildung 2-11). Das NAD wird hierbei in mehreren Iterationen verfeinert.

Ein NAD verwendet folgende vier Konstrukte:

- *Navigation classes:* Klassen aus dem Klassendiagramm, in welchen die Sichtbarkeit der Attribute und Methoden entsprechend der Zugriffsberechtigungen und Navigationsanforderungen von Benutzern eingeschränkt ist.
- *Navigation targets:* Eine Gruppierung von Elementen eines Modells.
- *Navigation links:* Definieren Navigationspfade, welchen ein Benutzer folgen kann. Ihnen können Navigationspattern und Navigationsfilter, welche mit OCL ausgedrückt werden, zugeordnet sein.
- *Collections:* Entsprechen Zugriffsstrukturen und geben an, wie ein Benutzer auf Information zugreifen kann.

OO-H definiert verschiedene Arten von Links:

- *Interne Links (Li)* definieren die Pfade innerhalb eines Navigationsziels.
- *Requirement-Links (Lr)* stellen den Ausgangspunkt der Navigation dar.
- *Exit-Links (ExP)* verweisen auf externe Knoten.
- *Service-Links (Ls)* verweisen auf Dienste.
- *Response-Links (Lres)* geben an, zu welchem Knoten nach Ausführung eines Dienstes zurückgekehrt werden soll.

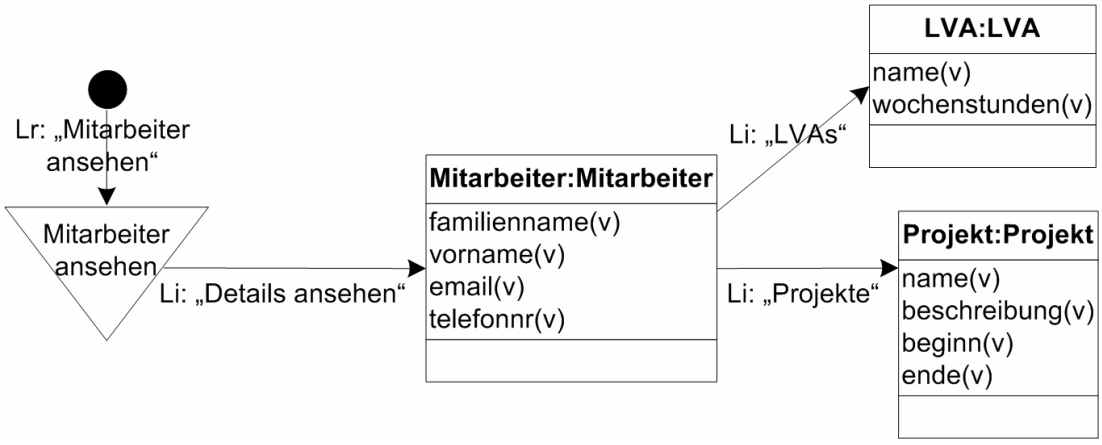


Abbildung 2-10 OO-H NAD Mitarbeiter ansehen

Beispiel: *Abbildung 2-10 zeigt die Navigation zur Auswahl und Ansicht von Mitarbeiterdaten. Mitarbeiter, LVA und Projekt sind Navigationsklassen, deren Attribute als sichtbar (v) deklariert sind. Das Menü „Mitarbeiter ansehen“ gibt an, wie der Benutzer auf die Daten zugreifen kann. Der Link „Details ansehen“ stellt einen internen Link dar, „Mitarbeiter ansehen“ ist ein Requirement-Link. Abbildung*

2-11 modelliert das Navigationsziel der Änderung von Projektdaten mit Hilfe des Service-Links „ändereProjekt“ und des Response-Links „ProjektGeändert“.

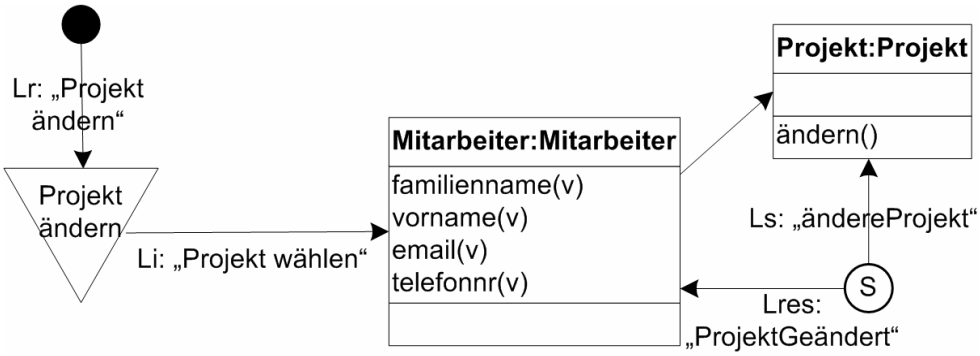


Abbildung 2-11 OO-H NAD Projekt ändern

2.2.4.4 Werkzeug

Das Werkzeug VisualWade (http://www.visualwade.com) basiert auf der OO-H Methode und unterstützt den Entwurf und die Generierung von Web-Anwendungen. Alle Modelle können mit Hilfe einer grafischen Notation entworfen werden. Die Modelle werden anschließend in XML Dokumente transformiert, welche die gesamte Information der Diagramme beinhalten. Aus ihnen kann automatisch die Web-Anwendung in Form von PHP Seiten erzeugt werden.

2.2.5 OntoWebber

OntoWebber [Jin01, Jin02] ist ein Website Managementsystem, welches zur Repräsentation der Modellelemente Ontologien verwendet. Diese legen das Vokabular zur Spezifikation von Websites fest und erlauben die Überprüfung der Modelle auf Integritätsbedingungen. Die OntoWebber-Ontologien sind in DAML+OIL definiert.

Im Allgemeinen ist eine Website eine Menge dynamischer oder statischer Webseiten. In OntoWebber wird eine solche Website eine instantiierte Website genannt. Zu einer Website kann es mehrere Instanzen geben, welche „Siteviews“ genannt werden. Eine Siteview wird über eine Menge von Modellen definiert, welche in Ontologien beschrieben sind. Die Instantiierung dieser Modelle erzeugt eine fertige Website.

OntoWebber verwendet sechs Modelle zur Spezifikation von Websites, welche zusammenfassend als „site model“ bezeichnet werden. Die Konzepte dieser Modelle werden in einer so genannten „Upper Ontology” festgelegt, welche das Metamodell definiert und mit dem UML-Metamodell vergleichbar ist. Sie legt die Konstrukte der einzelnen Modelle sowie Eigenschaften und Beziehungen dieser Konstrukte fest.

Modellbeschreibungen müssen das in dieser Ontologie definierte Vokabular verwenden.

Die OntoWebber Modelle können in zwei Kategorien eingeteilt werden:

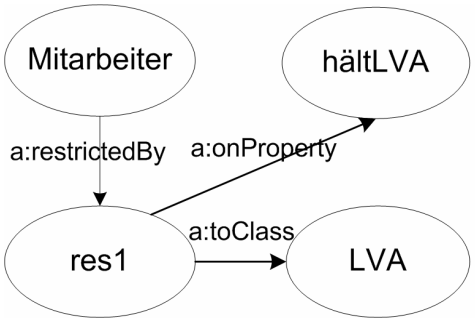
- *Site-spezifische Modelle:* Hierzu zählen das Domain-Modell, das Personalisierungsmodell und das Wartungsmodell. Diese Modelle existieren einmal pro Website.
- *Siteview-spezifische Modelle* sind das Navigationsmodell, das Inhaltsmodell und das Präsentationsmodell. Sie existieren pro Siteview und sind an eine bestimmte Benutzergruppe angepasst.

2.2.5.1 Entwurfsprozess

Der Entwurf einer Website ist ein iterativer Prozess, dessen Ausgangspunkt das Erfassen der Anforderungen darstellt. Anschließend wird das Domain-Modell entworfen, welches das Wissen über das Anwendungsgebiet definiert und Basis für den Inhalt ist, welcher auf den Webseiten angezeigt wird. Aufbauend auf das Domain-Modell folgt der Entwurf der Siteviews. Mit Hilfe einer grafischen Notation werden die drei Aspekte einer Siteview – Navigation, Content und Präsentation – in einem Siteview-Graphen modelliert. OntoWebber unterstützt auch die Modellierung der Personalisierung, um Webseiten an die unterschiedlichen Benutzerbedürfnisse anzupassen.

2.2.5.2 Domain-Modell

Das Domain-Modell wird als eine Ontologie über das Anwendungsgebiet definiert und enthält dessen Konzepte, Eigenschaften und Beziehungen. Das Schema dieser Ontologie wird in der Ontologiesprache DAML+OIL definiert.




```
<s:Class rdf:about="&c;Mitarbeiter">
   <a:restrictedBy rdf:resource="&c;res1"/>
</s:Class>
<s:Class rdf:about=”&c;LVA”/>
<rdf:Property rdf:about="&c;hältLVA"/>
<a:Restriction rdf:about="&c;res1">
   <a:toClass rdf:resource="&c;LVA"/>
   <a:onProperty rdf:resource="&c;hältLVA"/>
</a:Restriction>
```

Abbildung 2-12 OntoWebber Domain-Modell

Beispiel: *Abbildung 2-12 definiert die Klasse Mitarbeiter und die Beziehung „hältLVA“ zwischen der Klasse Mitarbeiter und der Klasse LVA.*

2.2.5.3 Siteview-Modell

Siteviews unterscheiden sich in ihrer Navigationsstruktur, den selektierten Daten und der Präsentation. Es können mehrere Siteviews für unterschiedliche Benutzer und Benutzergruppen erzeugt werden. Die Modellierung einer Siteview erfolgt mit Hilfe einer grafischen Notation, dem Siteview-Graphen. Dieser ist ein vereinfachtes konzeptuelles Modell zur Beschreibung des Hypertexts. Seine Basiselemente sind Karten, Seiten und Links. Eine Karte ist die kleinste Einheit eines Siteview-Graphen. Eine Seite beinhaltet eine oder mehrere Karten und entspricht einer physischen Webseite. Links werden verwendet, um Karten zu verknüpfen und auf diese Weise die Navigationsstruktur abzubilden.

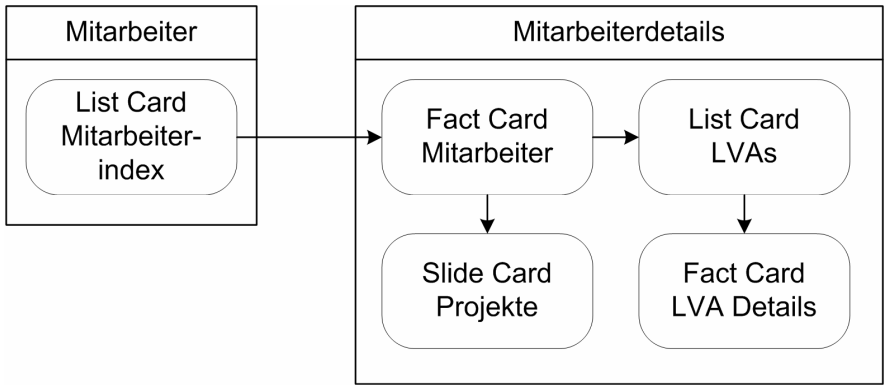


Abbildung 2-13 OntoWebber Siteview-Modell

Beispiel: *Abbildung 2-13 zeigt den Siteview-Graphen für die Institutsanwendung. Wird ein Mitarbeiter aus der Liste der Mitarbeiter selektiert, werden seine Details auf der Seite „Mitarbeiterdetails" angezeigt. Die Details umfassen Daten über den Mitarbeiter, über seine Lehrveranstaltungen und über seine Projekte. Um Details zu einer Lehrveranstaltung zu erreichen, muss diese aus der Liste der Lehrveranstaltungen gewählt werden. Projektdaten werden in diesem Beispiel angezeigt, indem die Projekte sequentiell durchlaufen werden können.*

OntoWebber stellt keine Konstrukte zur Verfügung, um die Änderung von Projektdaten zu modellieren.

Der Siteview-Graph umfasst drei Modelle zur Spezifikation einer Siteview, das Navigations-, Content- und Präsentationsmodell.

Das *Navigationsmodell* verknüpft Karten und Seiten mit Hilfe von Links. OntoWebber definiert dynamische und statische Karten. Statische Karten sind zum Beispiel statischer Text oder Bilder, welche unabhängig von den Datenquellen sind. Dynamische Karten hingegen beziehen sich auf Daten der Datenquellen.

Die vier Arten dynamischer Karten sind:

- *Fact card:* Zeigt eine Instanz einer Entität mit dessen Attributen.
- *List card:* Umfasst eine Liste von Instanzen einer Entität.
- *Slide card:* Ermöglicht das Blättern durch eine Liste von Instanzen einer Entität, wobei jeweils eine Instanz angezeigt wird.
- *Query card:* Dient dem Suchen von Entitäten.

Bei der *Modellierung des Content* wird angegeben, welche Daten eine Karte anzeigt. In statischen Karten kann dies beispielsweise Text oder ein Bild sein. Für dynamische Karten wird angegeben, auf welche Klasse des Domain-Modells sie sich beziehen. In den Karten werden dann die Instanzen dieser Klasse mit den gewählten Attributen angezeigt. Auf diese Weise wird das Domain-Modell mit dem Navigationsmodell verbunden. Abfragen zu Links geben an, wie der Inhalt einer Karte generiert wird.

Beispiel: *Der folgende Code zeigt am Beispiel der Karte „Mitarbeiterindex“, wie die Modelle in der definierten Ontologie repräsentiert werden. Die Karte bezieht sich auf die Entität Mitarbeiter und zeigt die Familiennamen der Mitarbeiter an.*

```
<ListCard rdf:about="&bm;CARD_1">
   <title>Mitarbeiterindex</title>
   <entity rdf:resource="&swrc;Mitarbeiter"/>
   <outputProp rdf:resource="&swrc;familienname"/>
   <indexProp rdf:resource="&swrc;familienname"/>
</ListCard>
```

2.2.5.4 *Werkzeug*

OntoWebber (http://www-db.stanford.edu/OntoAgents/OntoWebber) unterstützt den Entwurf einer Web-Anwendung auf Basis der genannten Ontologien. Die Architektur von OntoWebber gliedert sich in drei Ebenen [Jin01], der Integrations-, Kompositions- und Generierungsebene.

Die *Integrationsebene* („integration level“) integriert Daten aus verschiedenen Datenquellen. Zur Auflösung der syntaktischen und semantischen Unterschiede zwischen den verschiedenen Datenquellen werden die Datenquellen in ein einheitliches Datenformat (RDF) übersetzt und eine Referenzontologie definiert. Diese legt für alle Datenquellen ein einheitliches Vokabular fest.

Auf der *Kompositionsebene* („composition level“) werden Domain-Modelle erstellt und Siteviews erzeugt.

Auf *Ebene der Generierung* („generation level“) wird die Spezifikation der Siteview mit Hilfe von Integritätsbedingungen überprüft. Beispielsweise kann ein Constraint definiert werden, welcher feststellt, ob jede Karte von der Homepage erreichbar ist. Zur Erzeugung der Websites werden die Datenquellen abgefragt und statische oder

dynamische Webseiten entsprechend der Modelle der Siteview-Spezifikation generiert.

OntoWebber unterstützt nicht die Definition des Domain-Modells. Diese muss mit einem Ontologieeditor erstellt werden (vgl. Unterkapitel 3.4).

2.2.6 OntoWeaver

OntoWeaver [Lei02, Lei04] ist wie OntoWebber ein ontologiebasierter Ansatz zur Modellierung von Web-Anwendungen. Die Web-Anwendung wird in mehreren Schritten unter Verwendung verschiedener Modelle entworfen. Ontologien definieren das Vokabular dieser Modellbeschreibungen. Anschließend an die Modellierung können automatisch Webseiten generiert werden. Die Spezifikation einer Website besteht aus dem Domain-Modell, dem Siteview-Modell und einem expliziten Präsentationsmodell. OntoWeaver verwendet RDF zur Beschreibung der Sitespezifikationen und zur Annotation der Websites. Die Ontologien werden in RDFS definiert. Da dieser Ansatz sehr ähnlich zu OntoWebber ist und bis jetzt lediglich ein Prototyp zu OntoWeaver vorhanden ist, wird auf diesen Ansatz nicht weiter eingegangen.

2.2.7 Gegenüberstellung

Die Gegenüberstellung der Modellierungsmethoden (vgl. Tabelle 2-1) erfolgt anhand einiger Kriterien von [Schw04, S. 72]. Zusätzlich werden OntoWebber und OntoWeaver in den Vergleich aufgenommen.

Die betrachteten Modellierungsmethoden trennen die Modellierung einer Web-Anwendung nach Content, Hypertext und Präsentation. Zur Modellierung des Content greifen WebML sowie die objektorientierten Methoden auf UML Klassendiagramme bzw. Entity-Relationship Modelle zurück. OntoWebber und OntoWeaver hingegen repräsentieren das Content-Modell in Form einer Ontologie. Das Hypertext-Modell wird bei den objektorientierten Methoden in einer an UML angelehnten Notation, bei den anderen Methoden in einer eigenen Notation definiert. Zur Modellierung der Präsentation besitzen lediglich OOHDM, UWE und OntoWeaver eine Sprache auf abstrakter Ebene, während WebML, OO-H und OntoWebber technologieabhängige Konzepte einsetzen.

Modellierungsmethode	Modellierungsparadigma	Content-Modellierung	Hypertextmodellierung	Präsentationsmodellierung	Werkzeugunterstützung	Generierung	Stärken
WebML	DO	✓	✓	✗	✓	✓	verständliche Notation, ausgereiftes Werkzeug, Vielzahl an Konstrukten, gut dokumentiert
OOHDM	OO	✓	✓	✓	✗	✗	umfasst alle Ebenen
UWE	OO	✓	✓	✓	✓	✓	UML-basiert, Präsentationsmodell
OO-H	OO	✓	✓	✗	✓	✓	UML-basiert
OntoWebber	OB	✓	✓	✗	✓	✓	Integration von Datenquellen, Ontologien
OntoWeaver	OB	✓	✓	✓	PT	✓	Ontologien, Präsentationsmodell

DO datenorientiert OB ontologiebasiert
OO objektorientiert PT Prototyp

Tabelle 2-1 Gegenüberstellung Modellierungsmethoden

WebML besitzt kein explizites Präsentationsmodell, bietet jedoch eine verständliche Notation und ein gutes Modellierungswerkzeug. Sowohl die Notation als auch das Werkzeug sind umfassend dokumentiert. WebML stellt eine Vielzahl an Konstrukten zur Modellierung zur Verfügung und ist das einzige kommerzielle System innerhalb der betrachteten Methoden.

Die Methode OOHDM unterstützt die Modellierung einer Web-Anwendung auf allen drei betrachteten Ebenen. Jedoch besitzt OOHDM kein Werkzeug, um die Modelle zu entwerfen und aus den Modellen Webseiten zu generieren.

Die auf OOHDM aufbauenden Methoden UWE und OO-H basieren auf UML. Dies hat einerseits den Vorteil, dass UML ein bekannter Standard für die Modellierung traditioneller Anwendungen ist, andererseits ist Hintergrundwissen in UML eine Voraussetzung für das Verständnis der Modelle. Die Werkzeuge zu UWE und OO-H

befinden sich noch in Entwicklung und sind daher weniger ausgereift und dokumentiert als WebML.

Für den Einsatz von OntoWebber bzw. OntoWeaver spricht die Verwendung von Ontologien, welche in Unterkapitel 3.2 zur Repräsentation der Modelle betrachtet werden. Allerdings ist das Werkzeug zu OntoWebber noch nicht sehr ausgereift, zu OntoWeaver wurde lediglich ein Prototyp entwickelt. Weder OntoWebber noch OntoWeaver können Webseiten als Instanzen der Modelle abbilden. Eine solche Abbildung ist jedoch zur Überprüfung von Vorgaben zur Laufzeit notwendig (vgl. Unterkapitel 4.1). Da der Ansatz dieser Arbeit die Überprüfung zur Laufzeit unterstützen soll, ist die in OntoWebber bzw. OntoWeaver verwendete Repräsentation nicht ausreichend.

WebML besitzt im Vergleich zu den anderen Modellierungsmethoden das am weitesten entwickelte Werkzeug, welches die verwendete Notation gut unterstützt. Für diese Arbeit wird daher WebML als Modellierungsmethode gewählt.

Kapitel 3

Constraint-Prüfung

Aufbauend auf die Modellierung der Vorgaben und der Web-Anwendung wird überprüft, ob das Modell der Web-Anwendung die Vorgaben erfüllt. Die Vorgehensweise zur Überprüfung der Modelle entspricht dem Model Checking [Merz01]. Um die Vorgaben automatisch gegen das Modell der Web-Anwendung prüfen zu können, sind eine Repräsentation des Modells sowie der Vorgaben notwendig. Hierfür werden in dieser Arbeit Ontologien für die Modelle und Constraints für die Vorgaben eingesetzt. Inferenzmechanismen prüfen anschließend die Einhaltung der Vorgaben gegen das Modell.

Unterkapitel 3.1 geht auf Voraussetzungen zur Überprüfung der Modelle ein. Wissensrepräsentationen und Ontologien zur Repräsentation der Modelle werden in Unterkapitel 3.2 näher betrachtet. Unterkapitel 3.3 behandelt Ontologiesprachen und Unterkapitel 3.4 vergleicht Werkzeuge für das Erstellen von Ontologien, für das Definieren von Constraints und für die Überprüfung von Constraints mit Inferenzmechanismen.

3.1 Voraussetzungen

Die Überprüfung der Modelle basiert in dieser Arbeit auf Constraints. Diese repräsentieren die Vorgaben, welche das Modell erfüllen muss. [Born97] definiert einen Constraint als „a statement of a relation (in the mathematical sense) that we would like to have hold“. Constraints werden beispielsweise bei UML Diagrammen eingesetzt, um die Diagramme auf Korrektheit und Konsistenz zu überprüfen und den Entwurfsprozess zu unterstützen [Ha03, Cali02]. [Ha03] verwendet ein UML Metamodell, welches die Bedeutung der Komponenten und Beziehungen zwischen den Komponenten ausdrückt. Die Eigenschaften, welche die Diagramme erfüllen

müssen, werden in OCL definiert und können mit Hilfe eines Werkzeugs automatisch überprüft werden. [Cali02] überprüft UML Diagramme mit Hilfe der Beschreibungslogik, in welcher sowohl die Diagramme als auch die Constraints ausgedrückt werden. Dies ermöglicht die automatische Überprüfung der Constraints mit Hilfe von Inferenzmechanismen.

Beispiel: *Eine Vorgabe lautet, dass Daten über Mitarbeiter abrufbar sind. Das Content-Modell der Web-Anwendung muss daher eine Entität Mitarbeiter enthalten. Diese Vorgabe kann mit dem Constraint „es existiert eine Entität mit dem Namen Mitarbeiter" überprüft werden.*

Zur Repräsentation der Modelle eignen sich Ontologien, welche eine spezielle Form von Wissensrepräsentationen darstellen. Der Vorteil von Ontologien ist, dass sie Beziehungen zwischen Modellelementen explizit repräsentieren und daher Schlussfolgerungen ermöglichen [Will03].

Inferenzmechanismen unterstützen das Ziehen von Schlussfolgerungen über Ontologien und können sowohl Regeln gegen Modelle ausführen, um Constraints zu generieren, als auch die Constraints gegen Modelle überprüfen.

Zur Überprüfung der Modelle werden folglich Ontologien zur Repräsentation der Modelle, Constraintsprachen zur Repräsentation der Vorgaben und Inferenzmechanismen eingesetzt. Die Übersetzung der Modelle in eine Ontologiesprache sowie der Vorgaben in Constraints wird in Kapitel 4 behandelt.

3.2 Wissensrepräsentation

Wissensrepräsentation ist ein Bereich der künstlichen Intelligenz, dessen Zweck die Repräsentation eines Teils der Wirklichkeit zur Durchführung von Abfragen und Schlussfolgerungen ist. [Davi93] definiert fünf Rollen der Wissensrepräsentation. Eine davon bezeichnet Wissensrepräsentation als „a set of ontological commitments". Um eine Repräsentation der realen Welt zu finden, ist es notwendig zu abstrahieren und den Fokus auf die Aspekte zu richten, welche relevant sind. Diese Vorgehensweise ist mit Modellierungsmethoden vergleichbar. Während jedoch bei Modellierungsmethoden die Modellierung des Problembereichs im Vordergrund steht, verfolgen Ontologien den Zweck, Schlussfolgerungen über den Problembereich ziehen zu können.

Eine Wissensrepräsentation hat eine Syntax und eine Semantik. Die Syntax, also die Sprache, in welcher die Repräsentation niedergeschrieben wird, spielt eine untergeordnete Rolle. Wichtig ist vor allem die Semantik, welche durch Auswahl von Konzepten und Beziehungen zwischen diesen Konzepten festgelegt wird.

Eine Wissensrepräsentationssprache kann formal oder informal (z.B. natürliche Sprache) sein. Eine formale und daher zur Weiterverarbeitung durch Computer geeignete Form der Wissensrepräsentation sind *Ontologien*.

Der Ontologiebegriff in der Informatik unterscheidet sich von jenem der Philosophie, in welcher Ontologie als die Lehre vom Sein definiert ist. Im Bereich der Informatik spielen Ontologien in unterschiedlichen Forschungsgebieten eine wichtige Rolle, und es gibt eine Vielzahl von Definitionen, welche in [Guar95] verglichen werden. Die bekannteste Definition im Bereich der Wissensrepräsentation stammt von [Grub93] und lautet: „An ontology is an explicit specification of a conceptualization". Unter Konzeptualisierung (conceptualization) wird hierbei ein abstraktes, vereinfachtes Modell eines Teils der Wirklichkeit verstanden. Explizit bedeutet, dass die Konzepte und ihre Beziehungen explizit angegeben werden, um Austauschbarkeit zu ermöglichen.

Eine Ontologie ist ein formales Modell, welches das gemeinsame Vokabular zur Repräsentation von Wissen eines Teils der Wirklichkeit definiert. Dieses Wissen kann zwischen Personen und Applikationen ausgetauscht werden. Ontologien bestehen aus Konzepten, welche in einer Vererbungshierarchie angeordnet sind, Eigenschaften der Konzepte, Beziehungen zwischen Konzepten und Axiomen [Corc03].

Beispiel: *Eine Ontologie über eine Web-Anwendung beinhaltet die Konzepte Entität, Seite und Link. Diese Konzepte haben verschiedene Eigenschaften, z.B. besitzt eine Entität einen Namen. Beziehungen zwischen Konzepten drücken aus, dass auf Seiten Entitäten angezeigt werden und Seiten über Links verknüpft sind. Instanzen dieser Ontologie sind Web-Anwendungsmodelle, welche mit dem in der Ontologie definierten Vokabular beschrieben werden, und zum Beispiel einen Mitarbeiter als eine Instanz einer Entität definieren.*

Ontologien repräsentieren einen Ausschnitt der Wirklichkeit, innerhalb dessen Grenzen Aussagen überprüfbar sind und Schlussfolgerungen gezogen werden können. [Fens03, S. 17] unterscheidet Schlussfolgerungen über Konzepte und über Instanzen einer Ontologie. Erstere ermöglichen beispielsweise das Klassifizieren von Konzepten, d.h. die automatische Einordnung von neuen Konzepten in die gegebene Konzepthierarchie. Mit Hilfe von Schlussfolgerungen über Instanzen werden das Ableiten von Wissen, das Überprüfen von Constraints und das Ausführen von Abfragen und Regeln unterstützt.

Beispiel: *Fasst man das Content-Modell aus Abbildung 2-1 als Ontologie auf, sind mögliche Schlussfolgerungen:*

- *Klassifizieren: Eine Person, welche eine Telefonnummer besitzt, ist ein Mitarbeiter.*
- *Ableiten von Wissen: Ein Mitarbeiter, welcher keine Telefonnummer besitzt, arbeitet nicht am Institut.*
- *Mit Hilfe eines Constraints kann überprüft werden, ob jede Telefonnummer mit der Nummer der Universität beginnt.*
- *Eine Regel kann eingesetzt werden, um zu garantieren, dass Mitarbeiter, welche ein Projekt leiten, auch in diesem Projekt arbeiten.*

Zur Durchführung solcher Schlussfolgerungen sind Inferenzmechanismen notwendig. Diese können einerseits Regeln gegen Modelle ausführen, welche aufbauend auf bestimmte Bedingungen Aktionen auslösen. Andererseits können sie Constraints gegen Modelle überprüfen, um festzustellen, ob das Modell gewisse Eigenschaften erfüllt oder nicht erfüllt.

Zur Repräsentation der Ontologie und der Modelle ist eine Ontologiesprache notwendig. Erste Sprachen sind aus dem Bereich der logikbasierten Sprachen entstanden, welche als Wurzeln heutiger Ontologiesprachen angesehen werden können. [Corc03] unterscheidet:

- *Logik erster Ordnung:* Die Logik erster Ordnung (auch genannt Prädikatenlogik) ist eine Erweiterung der Aussagenlogik, welche zusätzlich zur Verknüpfung von Aussagen Quantoren erlaubt. Beispiel für eine Sprache dieser Kategorie ist KIF (Knowledge Interchange Format) [Gene92].
- *Frames und Logik erster Ordnung:* Frames repräsentieren sowohl Konzepte als auch Instanzen zu diesen Konzepten. Sie können in einer Vererbungshierarchie angeordnet werden und Eigenschaften besitzen. Ein Beispiel für eine Sprache dieser Art ist F-Logic (Frame-Logic) [Kife95].
- *Beschreibungslogik:* Die Beschreibungslogik beschreibt das Wissen mit Konzepten und Beziehungen zwischen den Konzepten und ermöglicht das Schlussfolgern über Konzeptbeschreibungen sowie das automatische Klassifizieren. Loom [Loom04] ist ein Beispiel dieser Sprachen.

Im Zusammenhang mit dem Semantic Web [Bern01] sind neue Ontologiesprachen entstanden, welche auf XML basieren. Beispiele hierfür sind RDFS, DAML+OIL und OWL. Zu diesen Sprachen sind Inferenzmechanismen sowie Werkzeuge vorhanden, um Schlussfolgerungen zu ermöglichen. Aufgrund des Trends zu XML werden diese Sprachen im folgenden Unterkapitel näher betrachtet.

3.3 Ontologiesprachen

Abbildung 3-1 zeigt einen Überblick über Ontologiesprachen, welche auf XML basieren. HTML [W3C99a] ist als Ausgangspunkt ungeeignet, da HTML lediglich der Publikation von Daten im Internet dient und mit Hilfe von Layoutinformationen angibt, wie der Browser Text anzeigen soll. XML [W3C04a] legt die Struktur von Dokumenten fest, drückt jedoch keine explizite Semantik aus. Zur Definition eines semantisch reicheren Modells als XML eignet sich RDF [W3C04c]. RDF beschreibt Ressourcen und definiert ein einfaches Datenmodell. Zu RDF gibt es eine XML Repräsentation.

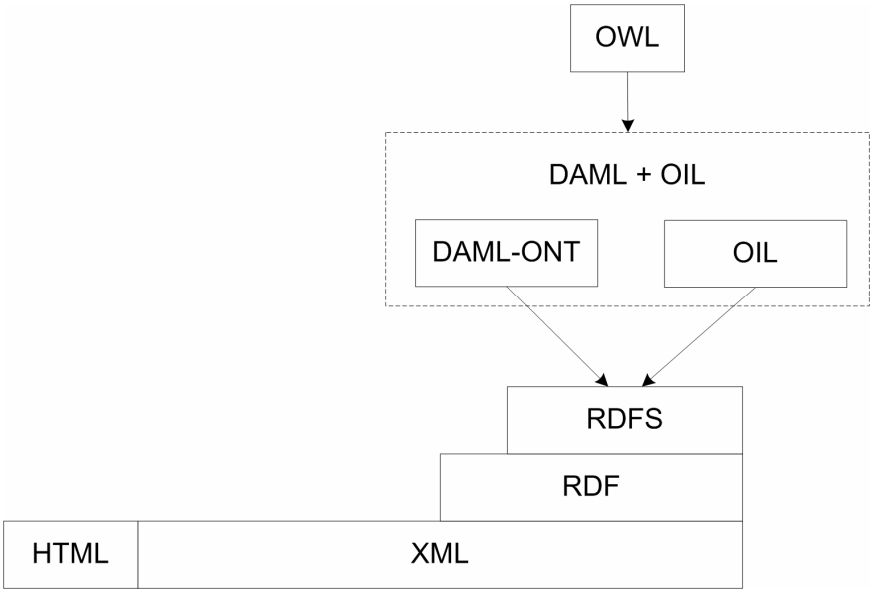


Abbildung 3-1 Ontologiesprachen (nach [Fens03, S. 9])

Die Ontologiesprachen bauen auf XML und RDF auf und legen das Vokabular fest, welches in den Ressourcen-Beschreibungen verwendet wird. RDFS [W3C04e] definiert grundlegende ontologische Konstrukte wie Klassen, Eigenschaften und Vererbungshierarchien. Erweiterungen zu RDFS sind DAML-ONT [Hend00] und OIL [Horr00]. Diese wurden zu DAML+OIL [W3C01c] zusammengefasst. OWL [W3C04f] ist die neueste Entwicklung und basiert auf DAML+OIL. Zweck der Erweiterungen sind detailliertere Beschreibungen der Klassen und Eigenschaften. Außerdem unterstützen sie die Definition von Beziehungen zwischen Ontologien, die Versionierung von Ontologien und das Ziehen von Schlussfolgerungen.

Im Folgenden wird genauer auf XML und RDF zur Datenrepräsentation sowie auf die Ontologiesprachen RDFS, DAML+OIL und OWL eingegangen, welche hinsichtlich ihrer Ausdrucksstärke zur Repräsentation der Modelle betrachtet werden.

3.3.1 XML

XML (Extensible Markup Language) [W3C04a] ist eine Teilmenge von SGML und ermöglicht die Strukturierung von Daten mit Hilfe von Tags. Die Struktur kann in einer DTD oder einem XML Schema [W3C01a] festgelegt werden. Hinsichtlich der Angabe der Semantik besitzt XML jedoch mehrere Nachteile [Deck00]. Es ist zwar möglich, bestimmte semantische Eigenschaften von der Struktur aufgrund der Verschachtelung und Reihenfolge von Elementen abzuleiten. Jedoch definiert XML nicht explizit die Bedeutung von Tags. Die Tagnamen sind meist implizit für Menschen verständlich (z.B. <Mitarbeiter>), aber nicht maschinell interpretierbar. XML garantiert keine allgemein gültige Interpretation der Daten, weshalb sich die Parteien vor der Kommunikation auf die Bedeutung der Tags einigen müssen. Ein weiterer Nachteil von XML ist, dass gleiche Information auf verschiedene Arten repräsentiert werden kann, was die Verarbeitung erschwert.

Da XML keine explizite Semantik besitzt, gibt es keinen standardisierten Weg, die Konstrukte zu interpretieren und Inferenzmechanismen zu entwickeln. Dies führte zur Entwicklung von RDF, welches eine klare Objektstruktur und die dafür notwendigen Tags definiert.

3.3.2 RDF

Das Resource Description Framework (RDF) [W3C04b, W3C04c] ist ein Modell zur Repräsentation von Metadaten, welches vom World Wide Web Consortium (W3C) entwickelt wurde.

In RDF werden Aussagen (statements) über Ressourcen getroffen. Ressourcen können Webseiten oder beliebige andere Objekte wie Personen, Bücher etc. sein und werden über URIs (Uniform Resource Identifier) identifiziert.

Beispiel: *Die folgenden zwei Aussagen definieren, dass der Mitarbeiter Huber die Lehrveranstaltung mit der Nummer 243145 abhält, welche den Namen Datenbanken besitzt.*

```
<Huber> <hältLVA> <243145>
<243145> <name> “Datenbanken“
```

Aussagen haben die Form eines Tripels aus Subjekt, Prädikat und Objekt. Das Subjekt identifiziert die Ressource, über welche eine Aussage gemacht wird. Das Prädikat ist eine Eigenschaft des Subjekts und ist ebenfalls eine Ressource. Der Wert der Eigenschaft ist das Objekt, welches entweder eine Ressource oder eine Zeichenkette (Literal) ist. Im Falle eines Literals ist der Wert der Eigenschaft konstant und besitzt keinen URI.

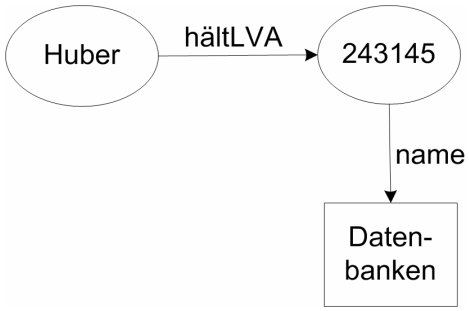


Abbildung 3-2 RDF Graph

Die Aussagen können als Graph mit Knoten und Kanten dargestellt werden. Subjekte und Objekte sind die Knoten, Prädikate die Kanten des Graphen. Ressourcen werden als Ellipsen, Literale als Rechtecke dargestellt. Durch Verknüpfung von Aussagen entsteht ein gerichteter Graph.

Beispiel: *Abbildung 3-2 zeigt einen solchen Graphen für die Aussagen, dass die Ressource Huber die LVA 243145 mit dem Namen Datenbanken abhält. Die Ressource 243145 ist sowohl ein Objekt als auch ein Subjekt. Der Name dieser Ressource (Datenbanken) ist eine Zeichenkette und daher konstant.*

RDF Aussagen bilden selber Ressourcen, auf welche mit weiteren Aussagen verwiesen werden kann. Diese Technik der Aussagen über Aussagen wird Reifikation genannt und ermöglicht es beispielsweise anzugeben, von wem eine Aussage gemacht wurde oder Zweifel an einer Aussage auszudrücken.

Um Gruppen von Ressourcen zusammenzufassen, enthält RDF vordefinierte Datentypen für Mengen (rdf:Bag, rdf:Seq, rdf:Alt) und Listen (rdf:List).

Zu RDF gibt es eine XML Repräsentation [W3C04d], um die Information über Ressourcen in maschinenausführbarer Art zu repräsentieren.

Beispiel: *Folgender Programmcode definiert in RDF/XML Syntax jene Aussagen, welche Abbildung 3-2 grafisch darstellt.*

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://www.example.com/terms#"
  xml:base="http://www.example.com">
<rdf:Description rdf:about="Huber">
    <ex:hältLVA rdf:resource="#243145"/>
</rdf:description>
<rdf:Description rdf:about="243145>
    <ex:name>Datenbanken</ex:name>
</rdf:description>
```

3.3.3 RDF Schema

RDF Schema (RDFS) [W3C04e] ist ein Vokabular zur Beschreibung von Klassen und Eigenschaften von RDF Ressourcen. RDF Schema wird in der RDF/XML Syntax formuliert und definiert weitere Konstrukte wie Klassen und Eigenschaften zu Klassen.

Klassen repräsentieren Kategorien von Dingen (z.B. Personen) und werden in einer Hierarchie angeordnet. Eigenschaften charakterisieren Klassen und können wie Klassen spezialisiert werden. Im Gegensatz zu objektorientierten Programmiersprachen sind Eigenschaften in RDFS global und unabhängig von Klassendefinitionen. Sie können jedoch Klassen zugeordnet werden.

Beispiel: *Abbildung 3-3 definiert die Klassen Person, Mitarbeiter und LVA sowie die Eigenschaften „hältLVA“ und „name“. Die Klasse Mitarbeiter ist eine Subklasse von Person. Die Eigenschaft „hältLVA“ ist der Klasse Mitarbeiter zugeordnet (rdfs:domain). Das Element rdfs:range gibt den Wertebereich von Eigenschaften an. Im Falle der Eigenschaft „hältLVA“ müssen die Werte Instanzen der Klasse LVA sein. Die Eigenschaft „name“, welche der Klasse LVA zugeordnet ist, besitzt als Wert keine Ressource, sondern eine Zeichenkette.*

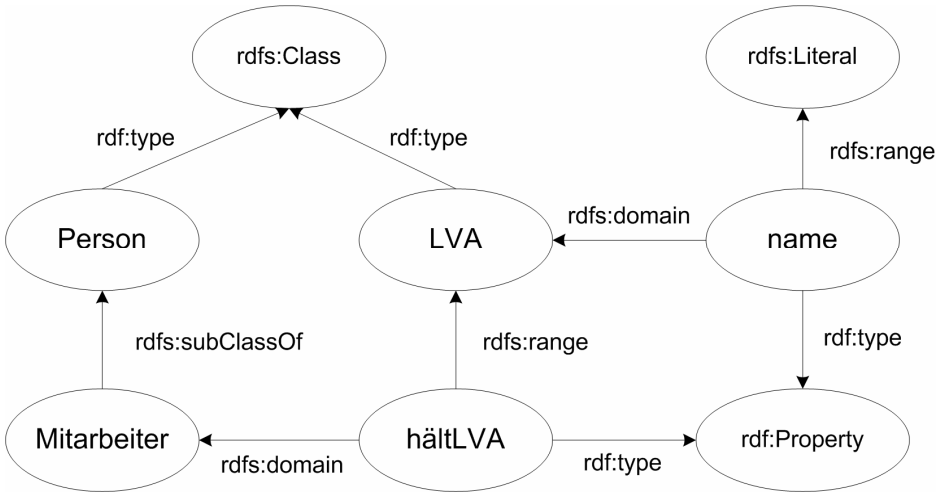


Abbildung 3-3 RDF Schema

Beispiel: *Die RDF/XML Syntax zu Abbildung 3-3 lautet:*

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.com/terms">
<rdfs:Class rdf:about=“Person“/>
<rdfs:Class rdf:about=“Mitarbeiter“>
   <rdfs:subClassOf rdf:resource=“#Person“/>
</rdfs:Class>
<rdfs:Class rdf:about=“LVA“/>
<rdf:Property rdf:about=“hältLVA“>
   <rdfs:domain rdf:resource=”#Mitarbeiter”/>
```

```
    <rdfs:range rdf:resource=”#LVA”/>
</rdf:Property>
<rdf:Property rdf:about="name">
    <rdfs:domain rdf:resource="#LVA"/>
    <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-
      19990303#Literal"/>
</rdf:Property>
```

Beispiel: *Folgender Code zeigt eine Instanz in RDF, welche das im RDF Schema definierte Vokabular verwendet.*

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex=“http://www.example.com/terms#“
  xml:base="http://www.example.com">
<ex:Mitarbeiter rdf:about=”Huber”>
    <ex:hältLVA rdf:resource=“#243145“/>
</ex:Mitarbeiter>
<ex:LVA rdf:about=“243145“>
    <ex:name>Datenbanken</ex:name>
</ex:LVA>
```

3.3.4 DAML+OIL

DAML+OIL [W3C01c] ist eine Zusammenfassung der Sprachen DAML-ONT und OIL. DAML-ONT (DARPA Agent Markup Language-Ontology) [Hend00] wurde von der US amerikanischen Forschungsorganisation DARPA entwickelt. OIL (Ontology Inference Layer) [Horr00] entstand im Rahmen eines europäischen Forschungsprogramms. Beide Sprachen basieren auf der Beschreibungslogik und erweitern RDFS, um eine detailliertere Modellierung von Klassen und Eigenschaften zu ermöglichen.

Zusätzlich zur Definition von Klassenhierarchien erlaubt DAML+OIL, Klassen voneinander abzugrenzen, die Äquivalenz zwischen Klassen anzugeben und Kombinationen von Klassen (Vereinigung, Durchschnitt, Negation) zu bilden. Außerdem können die Instanzen einer Klasse aufgezählt werden.

DAML+OIL ermöglicht im Gegensatz zu RDFS die Definition von Kardinalitätseinschränkungen zu Eigenschaften und die Definition inverser Eigenschaften. Außerdem kann festgelegt werden, dass eine Eigenschaft einen eindeutigen Wert (Schlüssel) besitzt. DAML+OIL unterstützt auch die Definition transitiver Eigenschaften und die Angabe der Äquivalenz von Eigenschaften.

3.3.5 OWL

Die „Web Ontology Language“ (OWL) [W3C04f] basiert auf DAML+OIL und baut auf der RDF/XML Syntax auf.

OWL besteht aus drei Untersprachen, deren Komplexität zunimmt:

- *OWL Lite:* Ziel von OWL Lite ist es, eine einfache Sprache zur Verfügung zu stellen, welche effizient zu realisieren ist. Zusätzlich zu RDFS können in OWL Lite Gleichheit und Ungleichheit angegeben und transitive, symmetrische und inverse Eigenschaften definiert werden. Kardinalitäts-einschränkungen bei Eigenschaften sind nur über 0 oder 1 möglich.
- *OWL DL* (DL für Beschreibungslogik) umfasst alle OWL Sprachkonstrukte. Um Vollständigkeit und Entscheidbarkeit zu garantieren, erlaubt OWL DL jedoch nicht die Definition von Metaklassen.
- *OWL Full* beinhaltet wie OWL DL alle Sprachkonstrukte, jedoch ohne auf die syntaktischen Freiheiten von RDF verzichten zu müssen. Die Unterstützung von Metaklassen erlaubt zwar eine maximale Ausdrucksstärke, jedoch gibt OWL Full keine Garantie bezüglich Entscheidbarkeit und Vollständigkeit.

OWL DL und OWL Full ermöglichen zusätzlich zu OWL Lite die Kombination von Klassen und Eigenschaften (Vereinigung, Durchschnitt, Negation) und das Aufzählen der Instanzen einer Klasse oder einer Eigenschaft. Klassen können voneinander abgegrenzt und zu Eigenschaften können beliebige Kardinalitätsein-schränkungen definiert werden.

3.4 Werkzeuge

Für die Überprüfung der Modelle ist ein Werkzeug notwendig, welches Ontologiesprachen zur Repräsentation der Modelle unterstützt und Inferenzmechanismen zur Verfügung stellt. Hierfür können Ontologieeditoren eingesetzt werden, welche das Erstellen von Ontologien mit Hilfe von grafischen Oberflächen erleichtern. Die hier betrachteten Werkzeuge beinhalten auch Sprachen zur Definition von Constraints und Regeln sowie Inferenzmechanismen zur Überprüfung der Constraints und zum Ausführen der Regeln.

In Folge werden Ontologieeditoren gegenübergestellt, welche mindestens eine der in Unterkapitel 3.3 genannten Ontologiesprachen unterstützen und Inferenz-mechanismen zur Verfügung stellen. Tabelle 3-1 zeigt die Bewertung der Werkzeuge anhand für die Auswahl eines Werkzeugs relevanter Kriterien. Ein ausführlicherer Vergleich von Ontologieeditoren findet sich in [Corc03].

	OILEd	**OntoEdit**	**Protégé**	**WebODE**
Anbieter	Universität Manchester	Ontoprise GmbH	Universität Stanford	Universität Madrid
Version	3.5	2.7	2.1.2	2.0.9
Frei verfügbar	✓	✗	✓	✓
Art der Applikation	Standalone	Standalone	Standalone	Web Applikation
Erweiterbar	✗	✓	✓	✗
Unterstützte Ontologie-sprachen	DAML+OIL	RDFS, DAML+OIL etc.	RDFS, DAML+OIL, OWL etc.	RDFS, DAML+OIL, OWL etc.
Inferenz-maschinen	FaCT	Ontobroker	PAL, Jess, Algernon etc.	Prolog, Jess
Visualisierung	✗	✓	✓	✓
Metaklassen	✗	✗	✓	✗

Tabelle 3-1 Gegenüberstellung Ontologieeditoren

3.4.1 OILEd

OILEd (http://oiled.man.ac.uk/) ist ein einfacher Ontologieeditor der Universität Manchester, welcher als Open-Source Produkt verfügbar ist und das Erstellen und Bearbeiten von DAML+OIL Ontologien ermöglicht. Als Inferenzmaschine wird FaCT verwendet. OILEd ist nicht erweiterbar und kann die Ontologien nicht visuell darstellen.

3.4.2 OntoEdit

OntoEdit (http://www.ontoprise.de/products/ontoedit) ist eine Entwicklungs-umgebung für Ontologien, welche an der Universität Karlsruhe entstand und nun von der Ontoprise GmbH weiterentwickelt wird. Das Werkzeug ist in den drei Versionen OntoEdit Free, OntoEdit und OntoEdit Professional verfügbar. OntoEdit Free ist nicht kostenpflichtig, jedoch in der Nutzung eingeschränkt. OntoEdit Professional unterstützt zusätzlich zu OntoEdit den Einsatz von Datenbanken und Servern und beinhaltet einen Regeleditor und ein Abfragewerkzeug. Die von OntoEdit

unterstützten Ontologiesprachen sind RDFS, DAML+OIL und F-Logic, für Inferenzen wird Ontobroker verwendet. OntoEdit ist über Plugins erweiterbar und ermöglicht die Visualisierung von Ontologien.

3.4.3 Protégé

Protégé (http://protege.stanford.edu/) wurde von der Universität Stanford entwickelt und ist als Open-Source Produkt verfügbar. Zu diesem Werkzeug steht eine Reihe von Plugins zur Verfügung, welche Ontologiesprachen, Inferenzmaschinen, die Visualisierung von Ontologien etc. unterstützen. Protégé ermöglicht das Importieren und Exportieren von Ontologien in XML, RDFS, DAML+OIL, OWL etc. Die in Protégé standardmäßig integrierte Inferenzmaschine ist PAL (Protégé Axiom Language). Über Plugins können jedoch auch andere Inferenzmaschinen verwendet werden, darunter Jess, Algernon und für frühere Protégé-Versionen (bis Version Protégé-2000) auch F-Logic und Prolog. Protégé ist mit Hilfe von eigenen Plugins erweiterbar. Eine Besonderheit von Protégé ist die Unterstützung von Metaklassen, deren Instanzen wiederum Klassen sind.

3.4.4 WebODE

Der Ontologieeditor WebODE (http://delicias.dia.fi.upm.es/webODE/) wurde von der Technischen Universität von Madrid entwickelt und läuft auf einem Web Server. Für den temporären Zugriff über ein Web Interface können freie Lizenzen beantragt werden. WebODE unterstützt das Importieren und Exportieren von Ontologien in XML, RDFS, DAML+OIL, OWL, UML etc. Ein Editor erleichtert das Erstellen von Axiomen, als Inferenzmaschinen stehen Prolog und Jess zur Verfügung. Das Werkzeug unterstützt die Visualisierung von Ontologien, kann jedoch nicht über eigene Plugins erweitert werden.

3.5 Auswahl von Technologien und Werkzeugen

Zur Überprüfung der Modelle mit Inferenzmechanismen ist die XML Repräsentation der WebML Modelle nicht ausreichend. Die Modelle müssen daher in eine Ontologiesprache und die Vorgaben in eine Constraintsprache übersetzt werden, damit Inferenzmechanismen die Vorgaben gegen die Web-Anwendung überprüfen können.

Als Ontologieeditor wird Protégé gewählt, da dieses Werkzeug frei verfügbar und erweiterbar ist und eine Vielzahl von Plugins zur Verfügung stellt. Protégé unterstützt alle betrachteten Ontologiesprachen sowie die Visualisierung von Ontologien. Die Verwendung von Metaklassen ist ebenfalls ein entscheidendes Kriterium für die Auswahl dieses Werkzeugs, um die Abbildung einer Web-

Anwendung auf drei Ebenen unterstützen zu können (vgl. Unterkapitel 4.1). Protégé ist weit verbreitet, ausreichend dokumentiert und wird aktiv weiterentwickelt.

Zur Repräsentation der Modelle wird in dieser Arbeit die Ontologiesprache RDFS eingesetzt, da Protégé RDFS unterstützt und auch Protégé spezifische Eigenschaften wie Kardinalitätseinschränkungen für RDFS zur Verfügung stellt. Als Constraint- bzw. Regelsprache und Inferenzmaschine wird PAL verwendet.

Kapitel 4

Prüfung von Web-Anwendungsmodellen

Zur Prüfung von Web-Anwendungsmodellen werden die Vorgaben in Constraints und die Modelle in eine Ontologie übersetzt. Dieses Kapitel betrachtet die Realisierung der Aufgabenstellung mit den entsprechenden Technologien und Werkzeugen. Die Repräsentation von Web-Anwendungen gliedert sich in unterschiedliche Ebenen, der Meta-, Modell- und Instanzebene, welche in Unterkapitel 4.1 beschrieben werden. In Unterkapitel 4.2 folgt ein Überblick über den Prozess zur Prüfung der Modelle. Unterkapitel 4.3 betrachtet die Architektur von WebML und Protégé und stellt die Repräsentation der Modelle in WebML mit jener in Protégé gegenüber. Unterkapitel 4.4 behandelt die Übersetzung der einzelnen Ebenen von WebML in Protégé. Wie Constraints aus den Modellen generiert und gegen Modelle überprüft werden, wird in Unterkapitel 4.5 erläutert. Dieses Unterkapitel geht außerdem auf die Erweiterung der Vorgaben und auf die Visualisierung von Abhängigkeiten zwischen Constraints ein. Unterkapitel 4.6 gibt einen Überblick über die Implementierung eines Plugins für Protégé, welches die Prüfung von Modellen unterstützt.

4.1 Abbildungsebenen

Die Repräsentation einer Web-Anwendung gliedert sich in drei Ebenen, der Meta-, Modell- und Instanzebene. Diese Ebenen werden nachfolgend anhand eines Ausschnitts aus dem Universitätsbeispiels erläutert (vgl. Abbildung 4-1), welcher festlegt, dass jede Mitarbeiterseite die Lehrveranstaltungen des jeweiligen Mitarbeiters anzeigt.

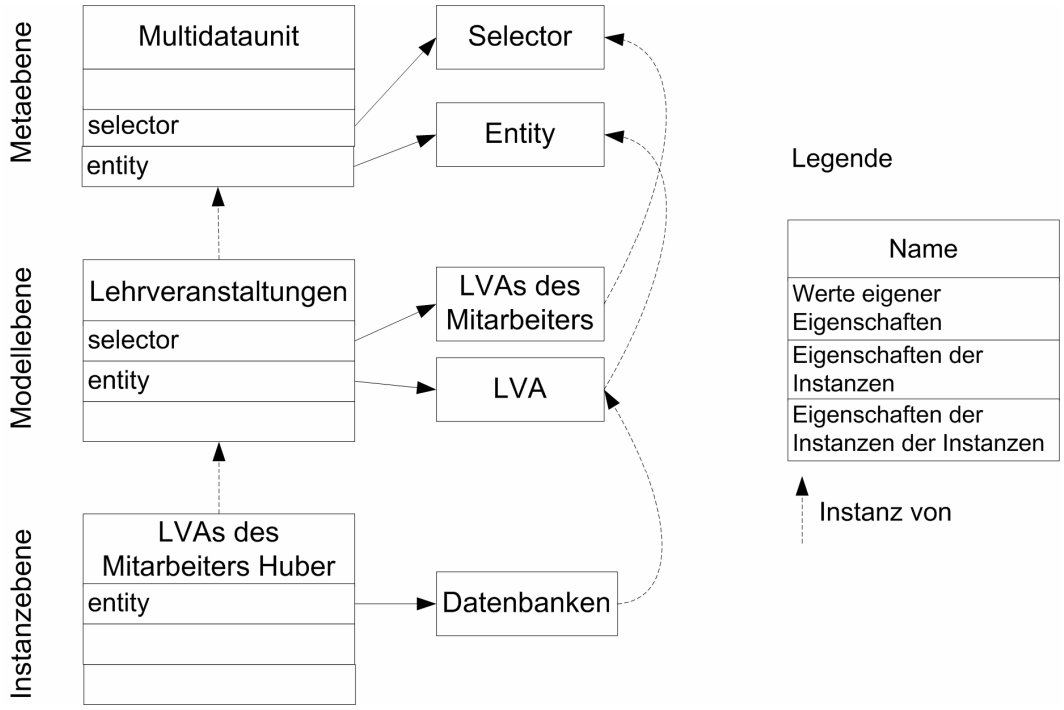


Abbildung 4-1 Ebenen am Beispiel Multidataunit

4.1.1 Metaebene

Die Metaebene legt die Metakonstrukte fest, welche in den Modellbeschreibungen auftreten. Beispiele für solche Konstrukte sind Entity, Page, Multidataunit und Link. Diese Konstrukte haben bestimmte Eigenschaften (z.B. einen Namen oder Beziehungen zu anderen Konstrukten), welche das Aussehen der Instanzen dieser Konstrukte festlegen.

Die Definition der Metakonstrukte erfolgt mit Hilfe von Metaklassen. Eine Metaklasse ist eine Klasse, deren Instanzen wiederum Klassen sind. Im Allgemeinen definiert eine Metaklasse Eigenschaften, welche für Klassen gelten. Das Aussehen von Instanzen dieser Klassen wird in den Klassen festgelegt. Der Ansatz von [Klas89] hingegen erlaubt, mit Hilfe von Metaklassen sowohl das Aussehen von Klassen als auch von Instanzen dieser Klassen zu bestimmen. Hierbei wird bei der Zuordnung von Eigenschaften zu Metaklassen unterschieden, ob die Eigenschaft für Klassen oder für Instanzen der Klassen gilt. Aufgrund der Teilung in drei Ebenen ist es wünschenswert, auf Metaebene ebenfalls das Aussehen sowohl der Modell- als auch der Instanzebene festlegen zu können. Diese Unterscheidung ist für die Abbildung von Instanzen notwendig.

Beispiel: *Abbildung 4-1 definiert auf Metaebene das Konstrukt „Multidataunit“, um Instanzen einer Entität anzuzeigen. Die Beziehung zum Konstrukt „Entity“ gibt die Entität an, deren Instanzen aufgelistet werden. Das „Selector“-Konstrukt wählt die anzuzeigenden Instanzen aus. Wie in der Abbildung ersichtlich ist, wird auf Metaebene zwischen Eigenschaften unterschieden, welche für Modelle oder für*

Instanzen der Modelle gelten. Die Beziehung zu einem Selector ist eine Eigenschaft von Modellelementen und ist auf Instanzebene nicht änderbar. Für die Beziehung zwischen einer Multidataunit und einer Entity hingegen gilt, dass der Wert der Beziehung erst auf Instanzebene bekannt ist.

4.1.2 Modellebene

Die Modellebene repräsentiert die modellierte Web-Anwendung mit Hilfe von Instanzen der auf Metaebene definierten Metakonstrukte. Auf dieser Ebene müssen Werte für jene Eigenschaften angegeben werden, welche auf Metaebene als Eigenschaften von Modellelementen deklariert wurden.

Beispiel: *Abbildung 4-1 zeigt das Modellelement „Lehrveranstaltungen“, welches Instanz einer Multidataunit ist. Das Modellelement enthält einen Wert für die Selector-Eigenschaft. Dieser Wert ist eine Instanz des Metakonstrukts „Selector“ und gibt an, dass nur Lehrveranstaltungen des jeweiligen Mitarbeiters aufgelistet werden. Die Beziehung zur Entität LVA legt fest, dass die Multidataunit Instanzen der Entität LVA anzeigt. Auf Modellebene kann jedoch noch kein konkreter Wert für diese Beziehung angegeben werden.*

4.1.3 Instanzebene

Die Instanzebene umfasst Instanzen der Modellebene, welche konkrete Webseiten darstellen und zur Laufzeit erzeugt werden. Die Instanzen enthalten Werte für jene Eigenschaften, welche auf Metaebene als Eigenschaften der Instanzen von Modellelementen definiert wurden.

Beispiel: *Für jede Mitarbeiterseite wird eine Instanz der Multidataunit Lehrveranstaltungen erzeugt, welche die Lehrveranstaltungen des jeweiligen Mitarbeiters auflistet. Abbildung 4-1 zeigt eine solche Instanz mit der LVA „Datenbanken“ für den Mitarbeiter „Huber“.*

4.2 Prozess

Abbildung 4-2 zeigt die einzelnen Schritte zur Umsetzung der Aufgabenstellung. Die erste Aufgabe besteht darin, die WebML Modelle in die Ontologiesprache RDFS zu übersetzen, um auf bestehende Inferenzmechanismen zur Überprüfung der Constraints zurückgreifen zu können. Um die WebML Modelle in eine RDFS konforme Darstellung zu bringen, wird zuerst die WebML DTD in einem RDF-Schema unter Verwendung von Metaklassen abgebildet (Schritt 1). Die in XML repräsentierten WebML Modelle werden anschließend mit Hilfe von XSLT in RDFS-Dokumente übersetzt (Schritt 2).

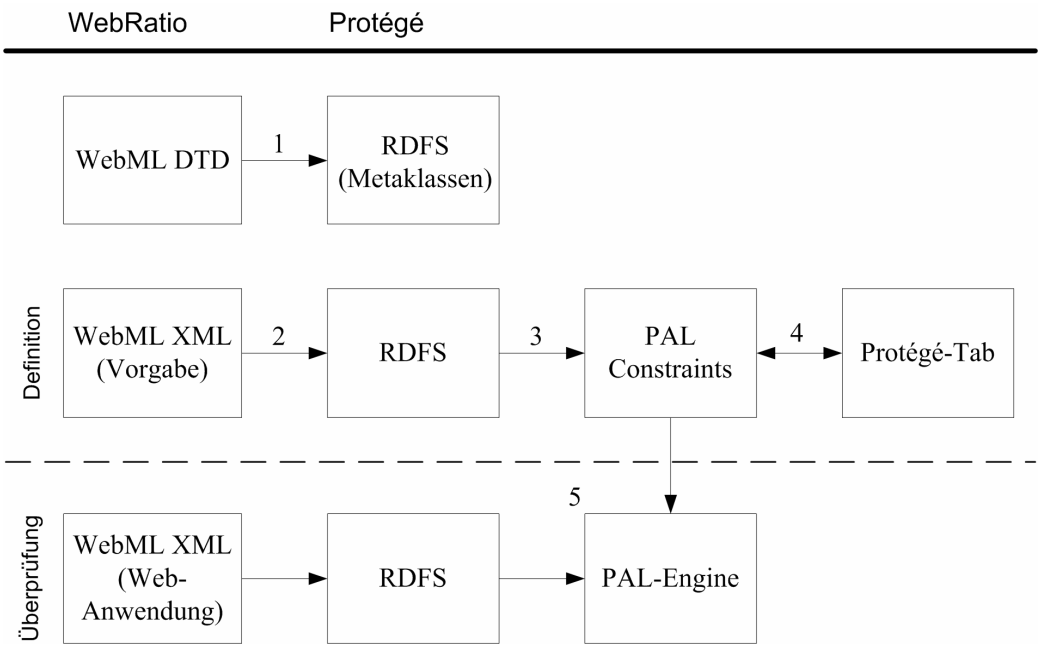


Abbildung 4-2 Prozess der Prüfung von Web-Anwendungsmodellen

Der dritte Schritt umfasst die Übersetzung der RDFS Dokumente, welche die Vorgaben abbilden, in Constraints mit Hilfe von frei definierbaren Mappingregeln. Die Mappingregeln können jederzeit geändert, hinzugefügt oder gelöscht werden. Als Constraint- und Regelsprache wird PAL (Protégé Axiom Language) [Crub02] verwendet.

Beispiel: *Eine Entität Mitarbeiter in WebML drückt die Vorgabe aus, dass die Web-Anwendung Daten über Mitarbeiter enthält. Aus dieser Vorgabe soll eine Mapping-regel den Constraint „es existiert eine Entität namens Mitarbeiter" generieren. Die Mappingregel gibt hierbei an, dass für alle Entitäten ein Constraint erzeugt wird, welcher überprüft, ob eine Entität mit dem jeweiligen Namen der Entität existiert.*

Da die gewählte Modellierungsmethode meist nicht alle Vorgaben ausdrücken kann, können im vierten Schritt die WebML Modelle in Protégé erweitert werden. Einerseits können Modellelemente hinzugefügt werden, andererseits ist es auch möglich, eigene Constraints in Form von PAL-Constraints zu ergänzen.

Beispiel: *Als Modellerweiterung wird die Klasse Professorenseite als Subklasse der Mitarbeiterseite definiert, welche zusätzlich einen Link auf das Sekretariat besitzt. WebML kann solche Subklassen nicht ausdrücken. Mit Hilfe eines eigenen Constraints können transitive Links überprüft werden. Diese geben an, dass ein Element von einem anderen Element entweder direkt oder indirekt (d.h. über eine oder mehrere Links) erreichbar ist. Ein transitiver Link kann beispielsweise festlegen, dass ausgehend von den Mitarbeiterdetails Informationen über Projekte entweder direkt oder über mehrere Links erreichbar sein müssen.*

Um eine Web-Anwendung auf Einhaltung der Vorgaben zu überprüfen, wird die Web-Anwendung in WebML modelliert. Das WebML-Modell wird mittels XSLT in ein RDFS-Dokument übersetzt und in Protégé zusammen mit den Metaklassen und den Constraints geladen. Die Constraint-Überprüfung erfolgt mit Hilfe der PAL-Engine (Schritt 5). Diese zeigt an, welche Constraints erfüllt bzw. nicht erfüllt sind.

4.3 Architektur

Im Folgenden wird auf die Architektur von WebML und Protégé eingegangen. WebML besitzt eine DTD, welche die Struktur der XML Dokumente festlegt. Protégé hat ein eigenes Wissensrepräsentationsmodell, unterstützt jedoch auch die Ontologiesprache RDFS.

4.3.1 WebML

WebML Modelle werden in XML Dokumenten abgespeichert. Die WebML DTD regelt die Struktur dieser Dokumente.

4.3.1.1 WebML DTD

Die WebML DTD [WebM03] definiert die Struktur der XML Dokumente, welche die WebML Modelle repräsentieren. Die Spezifikation für DTDs ist in der XML-Spezifikation [W3C04a] enthalten. Die WebML DTD besteht aus mehreren Dokumenten, diese Arbeit verwendet folgende Dokumente:

- *WebML.dtd:* Wurzel-DTD, bindet andere DTDs ein.
- *Structure.dtd:* Elemente und Attribute des Strukturmodells.
- *Navigation.dtd:* Elemente und Attribute des Hypertextmodells.

Die WebML DTD legt fest, welche Elemente in den WebML Modellen auftreten, wie die Elemente strukturiert sind und welche Attribute sie besitzen. Eine detaillierte Beschreibung der Elemente und Attribute findet sich in [WebR02].

Beispielsweise gibt es in der WebML DTD das Element MULTIDATAUNIT. Das Universitätsbeispiel verwendet dieses Element, um die Lehrveranstaltungen des Mitarbeiters (Instanzen der Entität LVA) auf der Mitarbeiterseite anzuzeigen. Die DTD deklariert dieses Element folgendermaßen:

```
<!ELEMENT MULTIDATAUNIT (SELECTOR?, DISPLAYATTRIBUTE*,
                         SORTATTRIBUTE*, LINK*)>
```

Einer MULTIDATAUNIT sind weitere Elemente untergeordnet (z.B. SELECTOR und LINK). Mit Hilfe solcher Verschachtelungen der Elemente drückt eine DTD Beziehungen zwischen den Elementen aus. Einer MULTIDATAUNIT kann kein oder ein SELECTOR untergeordnet sein, welcher die anzuzeigenden Instanzen

auswählt. Außerdem können von einer MULTIDATAUNIT beliebig viele Links ausgehen.

Die Attribute eines Elements werden ebenfalls in der DTD definiert.

```
<!ATTLIST MULTIDATAUNIT
   id ID #REQUIRED
   name CDATA #IMPLIED
   entity IDREF #IMPLIED
   distinct (yes | no) “no”
>
```

Eine MULTIDATAUNIT besitzt die Attribute id, name, entity und distinct. Das Attribut id ist verpflichtend und erlaubt die eindeutige Identifizierung eines Elements. Das Attribut entity gibt mit der Referenz auf das ID-Attribut einer ENTITY die Entität an, deren Instanzen die MULTIDATAUNIT anzeigt. Das Attribut distinct besitzt zwei mögliche Werte (yes und no), wobei der Wert „no“ den Default-Wert darstellt. Mit Hilfe dieses Attributs kann man festlegen, dass nur Instanzen mit unterschiedlichen Attributwerten angezeigt werden sollen.

4.3.1.2 WebML XML

Abbildung 4-3 zeigt die grafische Notation der Multidataunit Lehrveranstaltungen sowie die Definition ihrer Eigenschaften in WebRatio.

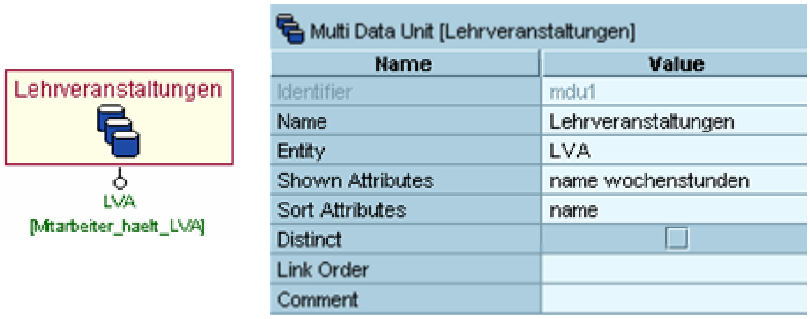


Abbildung 4-3 Multidataunit Lehrveranstaltungen in WebRatio

Diese Multidataunit wird in XML folgendermaßen repräsentiert:

```
<MULTIDATAUNIT distinct="no" entity="ent3" id="mdu1"
     name="Lehrveranstaltungen">
   <DISPLAYATTRIBUTE attribute="att8"/>
   <DISPLAYATTRIBUTE attribute="att9"/>
   <SORTATTRIBUTE attribute="att8" order="ascending"/>
   <SELECTOR defaultPolicy="fill">
      <SELECTORCONDITION id="cond1" name="LVAs des Mitarbeiters"
        predicate="in" relationship="rel2" type="required"/>
   </SELECTOR>
</MULTIDATAUNIT>
```

4.3.2 Protégé

Das Werkzeug Protégé unterstützt die Ontologiesprache RDFS, besitzt jedoch intern ein eigenes Wissensrepräsentationsmodell, welches RDFS sehr ähnlich ist und auf Frames basiert.

4.3.2.1 Wissensrepräsentationsmodell von Protégé

Das Protégé-Modell beruht auf folgenden Frames [Noy01b]:

- *Klassen:* Konzepte des Problembereichs, welche in einer Vererbungshierarchie angeordnet werden.
- *Slots* sind Eigenschaften von Klassen und Instanzen. Sie können unabhängig von einer Klasse definiert oder zu einer Klasse hinzugefügt werden.
- *Facets:* Zusatzinformationen zu Slots, welche Eigenschaften von Slots (z.B. Kardinalität, erlaubte Datentypen etc.) beschreiben.
- *Constraints* für das Ziehen von Schlussfolgerungen über die Ontologie oder Inhalte der Ontologie. Zur Definition und Überprüfung von Constraints ist ein Plugin notwendig.
- *Instanzen* von Klassen mit Werten für die Slots.

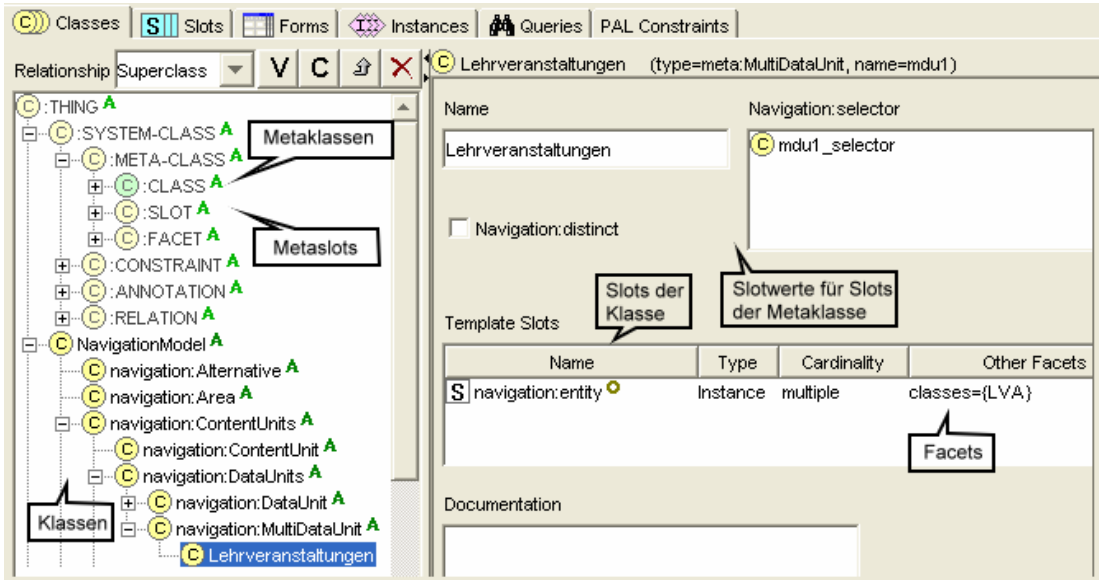


Abbildung 4-4 Protégé

4.3.2.2 Metaklassen und Metaslots

Protégé ermöglicht das Erstellen von Metaklassen und Metaslots [Noy01b], welche Vorlagen für Klassen bzw. Slots darstellen. Jede Klasse bzw. jeder Slot ist eine Instanz einer Metaklasse bzw. eines Metaslots. Es gibt eine vordefinierte Metaklasse und einen vordefinierten Metaslot. Der Benutzer kann jedoch weitere Metaklassen und Metaslots erstellen.

In Protégé definiert eine Metaklasse Eigenschaften einer Klasse. Hierfür werden Slots, welche diese Eigenschaften festlegen, zu der Metaklasse hinzugefügt. Protégé unterstützt jedoch nicht das Festlegen von Eigenschaften von Instanzen der Klasse in den Metaklassen, wie es in Unterkapitel 4.1.1 gefordert wurde. Solche Eigenschaften müssen in Protégé Klassen zugeordnet werden. Die Anforderung, auf Metaebene Eigenschaften von Modellinstanzen definieren zu können, wird daher folgendermaßen umgesetzt: Zu jeder Klasse gibt es zusätzlich zu einer Metaklasse auch eine Superklasse, welche diese Eigenschaften enthält. Die Superklasse sowie ihre Eigenschaften werden bei der Abbildung der Metaebene definiert.

4.3.2.3 RDFS Unterstützung

[Noy01a] beschreibt die Anpassung des Wissensrepräsentationsmodells von Protégé an andere Ontologiesprachen, um Ontologien in anderen Sprachen importieren und exportieren zu können. Diese Funktionalität wird über Plugins zur Verfügung gestellt. Eine Vielzahl der Konstrukte wie Klassen, Vererbungshierarchien, Eigenschaften und Instanzen sind in jeder Ontologiesprache vorhanden. Zur Beseitigung der Unterschiede wird so weit wie möglich auf Merkmale von Protégé zurückgegriffen, um die Austauschbarkeit von Modellen unterschiedlicher Sprachen zu ermöglichen. Im Folgenden wird auf die Handhabung der Unterschiede zwischen dem Protégé-Modell und RDFS eingegangen, wobei die Unterschiede in vier Kategorien eingeteilt werden:

- *Konzepte, welche gleich sind, aber unterschiedlich bezeichnet werden:* In Protégé werden Eigenschaften als Slots, in RDFS als Properties bezeichnet. Für solche Konzepte ist eine direkte Übersetzung möglich. In dieser Arbeit wird die Bezeichnung Slot verwendet.
- *Konzepte, welche gleich sind, aber unterschiedlich ausgedrückt werden:* Protégé fügt Slots einer Klasse hinzu, in RDFS wird dies über die Domain einer Property ausgedrückt.
- *Konzepte, welche nur in RDFS existieren:* Für diese Konzepte werden in Protégé Metaklassen und Metaslots definiert.
- *Konzepte, welche nur in Protégé existieren:* Hierzu zählen Kardinalitäten für Slots, inverse Slots und Default-Werte für Slots. Außerdem stehen in Protégé mehr Datentypen als in RDFS zur Verfügung. Um diese Konzepte auch in RDFS verfügbar zu machen, werden in RDFS neue, nicht standardisierte Eigenschaften definiert.

Das RDF Plugin führt die Anpassung des Wissensrepräsentationsmodells von Protégé an RDFS durch. Es definiert Protégé spezifische Konzepte in einem eigenen Namensraum (xmlns:a=http://protege.stanford.edu/system#), um diese Konzepte in

RDFS zur Verfügung zu stellen. Namensräume (namespaces) dienen der Vermeidung bzw. Auflösung von Namenskonflikten gleich benannter Elemente.

Zu diesen Konzepten zählen u.a. Kardinalitätseinschränkungen für Slots, welche in RDFS nicht definierbar sind. Kardinalitäten in Protégé werden daher in RDFS mit a:minCardinality und a:maxCardinality ausgedrückt. Protégé besitzt zusätzlich zu Literal die Datentypen Integer, Float, Boolean und Symbol (zur Aufzählung der möglichen Werte eines Slots). Diese werden in RDFS mit a:range übersetzt. Für die Definition von Standardwerten verwendet das RDFS Plugin die Eigenschaft a:defaultValues. Ob eine Klasse abstrakt oder konkret ist, wird in RDFS mit a:role angegeben. In Protégé besteht die Möglichkeit, Slots zu überschreiben, um die Eigenschaften eines Slots (z.B. Kardinalität oder erlaubte Werte) einzuschränken. Dies wird in RDFS mit a:OverridingProperty ausgedrückt.

Im Folgenden bezeichnet RDFS das um Protégé spezifische Konstrukte erweiterte RDFS, welches in dieser Arbeit verwendet wird.

4.3.3 Gegenüberstellung

In WebML gibt es Elemente, welche Attribute sowie verschachtelte Elemente besitzen. Das Protégé-Modell basiert auf Frames, welche Klassen und Slots sowie Instanzen mit Werten für die Slots darstellen. Sowohl die Repräsentation in WebML als auch jene mit Frames ist objektorientiert, weshalb Protégé geeignet ist, WebML Modelle zu repräsentieren.

Durch die Verschachtelung der Elemente entsteht bei XML Dokumenten eine Baumstruktur, während RDFS Dokumente Graphen darstellen. Die Verschachtelung drückt in XML implizit Beziehungen aus. In RDFS können Elemente ebenfalls verschachtelt sein, jedoch haben die Beziehungen eine explizite Bedeutung. Beispielsweise drücken sie Vererbungshierarchien oder Beziehungen zu Klassen bzw. Slots aus.

Ein weiterer Unterschied betrifft die Gliederung in Ebenen. Abbildung 4-5 zeigt die Meta-, Modell- und Instanzebene in WebML und in Protégé. In WebML regelt die DTD die Struktur der XML Dokumente, welche die Modelle repräsentieren. Instanzen können in WebML nicht direkt abgebildet werden, sondern die generierten Webseiten (z.B. in Form von JSP-Seiten) stellen die Instanzen dar. Protégé definiert mit Hilfe von Metaklassen das Aussehen der RDFS Dokumente auf Modellebene. Instanzen der Modelle können in RDF repräsentiert werden.

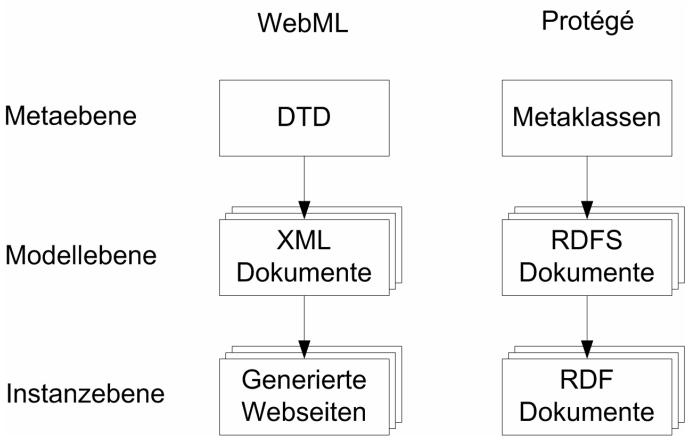


Abbildung 4-5 Ebenen in WebML und Protégé

4.3.3.1 Metaebene

Die WebML DTD legt die Metaebene mit Hilfe von Element- und Attributdeklarationen fest. Beziehungen werden ausgedrückt, indem Attribute auf Elemente verweisen, oder implizit durch die Verschachtelung von Elementen. WebML trifft hierbei keine Unterscheidung, ob sich eine Beziehung auf die Modell- oder Instanzebene bezieht (vgl. Unterkapitel 4.1.1), da Instanzen nicht direkt abbildbar sind.

In Protégé gibt es Klassen und Slots sowie Instanzen der Klassen mit Werten für Slots. Klassen, welche Instanzen anderer Klassen sind, werden als Metaklassen bezeichnet. Die Unterscheidung zwischen Eigenschaften von Modellen und Instanzen kann in Protégé im Gegensatz zur DTD berücksichtigt werden, indem Slots entweder der Metaklasse oder der Klasse zugeordnet werden (vgl. Unterkapitel 4.4.1).

Klassen in Protégé entsprechen Elementen in WebML. Slots drücken in RDFS sowohl Eigenschaften von Klassen als auch Beziehungen zwischen Klassen aus. Sie sind daher sowohl mit Attributen in WebML vergleichbar als auch mit jenen Beziehungen, welche in WebML durch das Verschachteln von Elementen dargestellt werden. RDFS drückt Beziehungen jedoch nicht implizit durch die Anordnung der Elemente aus, sondern benennt alle Beziehungen. Tabelle 4-1 vergleicht die Abbildung der Metaebene in der WebML DTD mit jener in Protégé.

DTD	**Protégé**
Element	Metaklasse
Beziehung zu untergeordnetem Element	Slot (rdfs:domain, rdfs:range)
verpflichtende Beziehung (keine	a:minCardinality 1,

Angabe der Kardinalität)	a:maxCardinality 1
? (optional)	a:minCardinality 0, a:maxCardinality 1
* (beliebig)	a:minCardinality 0
+ (mindestens einmal)	a:minCardinality 1
Attribut	Slot (rdfs:domain)
ID	URI
IDREF	rdfs:range (Angabe der Klasse)
CDATA	rdfs:range (Literal)
#REQUIRED (verpflichtend)	a:minCardinality 1, a:maxCardinality1
#IMPLIED (optional)	a:minCardinality 0, a:maxCardinality 1
Aufzählung der möglichen Werte	a:range (Symbol)
Default-Wert	a:defaultValues

Tabelle 4-1 Gegenüberstellung DTD und Protégé

4.3.3.2 Modellebene

WebML repräsentiert Modelle in XML Dokumenten mit Hilfe von Elementen und Attributen. Elemente entsprechen Klassen in Protégé, welche Instanzen der Metaklassen sind. Attribute sowie Beziehungen zu verschachtelten Elementen werden in Protégé mit Slots ausgedrückt. Protégé unterstützt im Gegensatz zu WebML die Definition von Vererbungshierarchien mit Hilfe von rdfs:subClassOf.

4.3.3.3 Instanzebene

WebML besitzt keine richtige Instanzebene, sondern WebRatio generiert Webseiten z.B. in Form von JSP-Seiten. Protégé hingegen ermöglicht die Abbildung von Modellinstanzen in RDF, indem Instanzen der auf Modellebene erzeugten Klassen angelegt werden.

4.4 Abbildung WebML in Protégé

Um die WebML Dokumente in RDFS zu übersetzen, wird zuerst die WebML DTD in ein RDFS Schema unter Verwendung von Metaklassen und Metaslots abgebildet. Ein XSLT Stylesheet legt die Übersetzung der WebML Modelle (XML) in RDFS Dokumente fest. Somit kann jedes Modell von WebRatio in Protégé geladen werden (vgl. Abbildung 4-2).

Dieses Unterkapitel beschreibt den Transformationsprozess der Meta-, Modell- und Instanzebene von WebML in Protégé.

4.4.1 Abbildung der Metaebene

Die Metaebene bildet die DTD unter Verwendung von Metaklassen und Metaslots in einem RDFS Dokument ab. Die Metaebene ist für alle Modelle identisch und muss daher nur einmal festgelegt werden. Folgende Namensräume werden bei der Abbildung definiert:

- *meta:* Beinhaltet Metaklassen und Metaslots.
- *webml:* Für modellübergreifende Konstrukte.
- *structure:* Strukturmodell.
- *navigation:* Hypertextmodell.

Im Allgemeinen werden Elemente in Metaklassen und Attribute sowie Beziehungen zu untergeordneten Elementen in Slots übersetzt. Dieser Vorgang wird im Folgenden anhand des Beispiels der Multidataunit betrachtet.

Das Element MULTIDATAUNIT wird mit einer Metaklasse, welche Subklasse der Standard-Metaklasse von Protégé ist, abgebildet.

DTD	<!ELEMENT MULTIDATAUNIT>
RDFS	<rdfs:Class rdf:about="&meta;MultiDataUnit"> <rdfs:subClassOf rdf:resource="&rdfs;Class"/> </rdfs:Class>

Beziehungen zu untergeordneten Elementen werden in RDFS mit Slots definiert. Einer MULTIDATAUNIT ist das Element SELECTOR untergeordnet. Für diese Beziehung wird in RDFS der Slot selector erstellt. Die Domain des Slots (rdfs:domain) legt fest, dass der Slot der Metaklasse Multidataunit zugeordnet ist. Erlaubte Werte des Slots sind Instanzen der Metaklasse Selector. Dies wird mit rdfs:range angegeben.

DTD	<!ELEMENT MULTIDATAUNIT (SELECTOR?)>
RDFS	<rdf:Property rdf:about="&navigation;selector" a:maxCardinality="1"> <rdfs:domain rdf:resource="&meta;MultiDataUnit"/> <rdfs:range rdf:resource="&meta;Selector"/> </rdf:Property>

Attribute werden analog zu untergeordneten Elementen mit Slots abgebildet. Die Domain des Slots entspricht jenem Element, zu welchem das Attribut deklariert wurde. Der Attributtyp CDATA stimmt mit dem RDF Datentyp Literal für Zeichenketten überein.

DTD	<!ATTLIST MULTIDATAUNIT name CDATA #IMPLIED >
RDFS	<rdf:Property rdf:about="&webml;name" a:maxCardinality="1"> <rdfs:domain rdf:resource="&meta;MultiDataUnit"/> <rdfs:range rdf:resource="&rdfs;Literal"/> </rdf:Property>

Der Attributtyp IDREF drückt in einer DTD eine Referenz eines Attributs auf ein anderes Element aus. In RDFS wird mit rdfs:range jene Klasse angegeben, deren Instanzen der Slot referenzieren kann.

DTD	<!ATTLIST MULTIDATAUNIT entity IDREF #IMPLIED >
RDFS	<rdf:Property rdf:about="&navigation;entity"> <rdfs:domain rdf:resource="&navigation;MultiDataUnit"/> <rdfs:range rdf:resource="&structure;Entity"/> </rdf:Property>

Zur Aufzählung erlaubter Attributwerte wird in RDFS der Protégé-Datentyp Symbol verwendet. Die Angabe von Default-Werten ist ebenfalls möglich.

DTD	<!ATTLIST LINK type (normal | automatic | transport) “normal” >
RDFS	<rdf:Property rdf:about="&navigation;linkType" a:defaultValues="normal" a:maxCardinality="1" a:range="symbol"> <rdfs:domain rdf:resource="&meta;Link"/> <rdfs:range rdf:resource="&rdfs;Literal"/> <a:allowedValues>normal</a:allowedValues> <a:allowedValues>automatic</a:allowedValues> <a:allowedValues>transport</a:allowedValues> </rdf:Property>

Der Datentyp Boolean in Protégé entspricht den Attributwerten (yes | no).

DTD	<!ATTLIST MULTIDATAUNIT distinct (yes | no) “no” >
RDFS	<rdf:Property rdf:about="&navigation;distinct" a:defaultValues="false" a:maxCardinality="1" a:range="boolean"> <rdfs:domain rdf:resource="&meta;MultiDataUnit"/> <rdfs:range rdf:resource="&rdfs;Literal"/> </rdf:Property>

Wichtig bei der Abbildung ist die Unterscheidung, ob der Wert einer Beziehung auf Modell- oder Instanzebene festgelegt wird (vgl. Unterkapitel 4.1.1 und Abbildung 4-1). Diese Unterscheidung wird in WebML nicht getroffen.

Beispiel: *Eine Multidataunit besitzt sowohl eine Beziehung zu einem Selector als auch zu einer Entity. Der Wert der Beziehung zu einem Selector steht bereits zum Zeitpunkt des Entwurfs fest und ändert sich auf Instanzebene nicht. Daher wird der Slot selector der Metaklasse Multidataunit zugeordnet (vgl. Abbildung 4-6). Der Slot drückt eine Eigenschaft einer Instanz der Metaklasse aus. Eine Instanz dieser Metaklasse ist z.B. die Klasse Multidataunit Lehrveranstaltungen, welche bereits einen konkreten Wert für diesen Slot aufweist (vgl. Abbildung 4-10).*

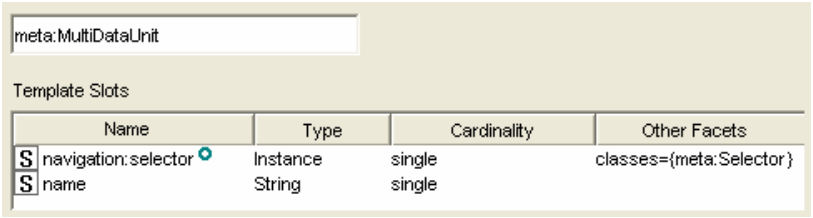


Abbildung 4-6 Metaklasse Multidataunit in Protégé

Beispiel: *Für die Beziehung zwischen einer Multidataunit und einer Entity gilt, dass der Wert der Beziehung erst zur Laufzeit bekannt ist. Zum Zeitpunkt des Entwurfs wird die Entität ausgewählt, deren Instanzen angezeigt werden. Welche Instanzen dies konkret sind, steht erst zur Laufzeit fest, wenn die Webseite erzeugt wird. Beispielsweise kann eine solche Instanz für die Mitarbeiterseite Huber die LVA „Datenbanken“ sein. Die Werte der Beziehung unterscheiden sich für jede Mitarbeiterseite.*

Metaklassen in Protégé können keine Eigenschaften definieren, welche für Instanzen der Klassen gelten. Daher werden solche Eigenschaften mit Hilfe von abstrakten Superklassen umgesetzt (vgl. Unterkapitel 4.3.2.2). Zu jeder Metaklasse wird eine gleichnamige abstrakte Superklasse, zum Beispiel die Superklasse Multidataunit, definiert. Wird eine Instanz der Metaklasse erzeugt, muss diese auch Subklasse

dieser abstrakten Klasse sein. Slots, welche Eigenschaften von Instanzen einer Klasse darstellen, werden auf Metaebene in der abstrakten Superklasse definiert und auf Modellebene überschrieben.

Beispiel: *Die abstrakte Klasse Multidataunit legt fest, dass der Wert des Slots entity eine Instanz einer Entity referenziert. Abbildung 4-7 zeigt die Definition der Superklasse Multidataunit in Protégé.*

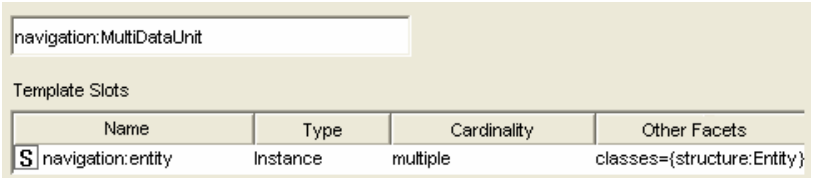


Abbildung 4-7 Superklasse Multidataunit in Protégé

Sowohl die Metaklassen als auch die Klassen werden in einer Vererbungshierarchie angeordnet, welche die WebML DTD nicht ausdrücken kann. Die Vererbungshierarchie hat den Vorteil, dass bestimmte (Meta-)klassen dieselben Slots besitzen und es daher ausreichend ist, diese Slots lediglich zu den Superklassen hinzuzufügen. Eigenschaften der Slots können dann für eine bestimmte Subklasse überschrieben werden.

Beispiel: *Abbildung 4-8 zeigt die Einordnung der Multidataunit in die Vererbungshierarchie. Der Slot selector wurde der Metaklasse Dataunits zugeordnet, da sowohl die Metaklasse Dataunit als auch die Metaklasse Multidataunit diesen Slot besitzen.*

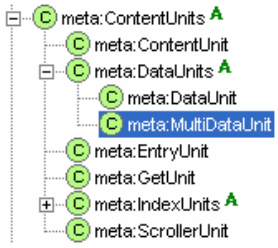


Abbildung 4-8 Vererbungshierarchie auf Metaebene

4.4.2 Übersetzung auf Modellebene

Um beliebige WebML Modelle in Protégé laden zu können, werden die XML Dokumente, welche die WebML Modelle repräsentieren, in RDFS übersetzt (vgl. Abbildung 4-9).

Die Übersetzung erfolgt mit Hilfe von XSLT (Extensible Stylesheet Language Transformations) [W3C01b], einer Transformationssprache zur Umwandlung von XML Dokumenten in ein anderes Format. XSLT basiert auf dem Prinzip des „Pattern Matching“. Das XSLT Stylesheet definiert eine Menge von Template-Regeln zur Umwandlung eines Quellbaumes in einen Ergebnisbaum. Eine solche Regel besteht

aus einem Muster und einem Template. Das Muster gibt mit Hilfe eines XPath-Ausdrucks an, für welche Knoten des Quellbaums das Template gilt. Im Template ist definiert, wie der entsprechende Teil des Ergebnisbaums erzeugt wird. Bei der Transformation wird das Muster gegen die Elemente des Quellbaums getestet. Wird das Muster im Quellbaum gefunden, führt dies zur Ausführung des Templates und somit zur Erzeugung eines Teils des Ergebnisbaums.

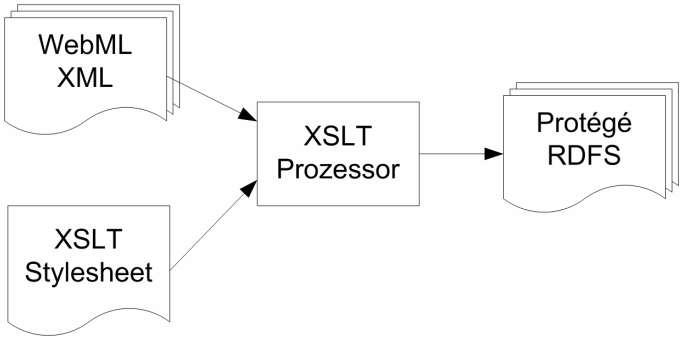


Abbildung 4-9 Übersetzung der Modelle mit XSLT

Im Allgemeinen wird jedes Element des XML Dokuments in eine Klasse übersetzt, welche Instanz der entsprechenden Metaklasse ist. Der URI dieser Instanz ist die ID des WebML Elements. Da die Klassen in RDFS analog zu den Metaklassen in einer Vererbungshierarchie angeordnet sind, ist auch die entsprechende Superklasse anzugeben.

Beispiel: *Für das Universitätsbeispiel wird die Multidataunit Lehrveranstaltungen in eine Klasse übersetzt, welche Instanz der Metaklasse Multidataunit und Subklasse der abstrakten Klasse Multidataunit ist.*

Attributwerte in XML entsprechen Slotwerten in RDFS. Bei der Übersetzung der Beziehungen zu untergeordneten Elementen ist darauf zu achten, ob die Beziehung auf Meta- oder auf Modellebene definiert wurde (vgl. Unterkapitel 4.1.2 und Abbildung 4-1). Referenziert der Slot, welcher die Beziehung repräsentiert, Elemente der Metaebene, ist der Wert dieses Slots auf Modellebene festzulegen.

Beispiel: *Der Wert des Slots selector der Multidataunit Lehrveranstaltungen ist die Klasse, in welche das Selector-Element übersetzt wurde.*

Referenziert der Slot hingegen Elemente der Modellebene, wurde bei der Abbildung der Metaebene der entsprechende Slot nicht in der Metaklasse, sondern in der abstrakten Superklasse festgelegt. In diesem Fall wird für die Klasse nicht der Slotwert angegeben, sondern der Slot wird überschrieben, um den Wertebereich des Slots einzuschränken.

Beispiel: *Die Multidataunit Lehrveranstaltungen überschreibt den Slot entity, um festzulegen, dass gültige Slotwerte nur Instanzen der Entität LVA sein können.*

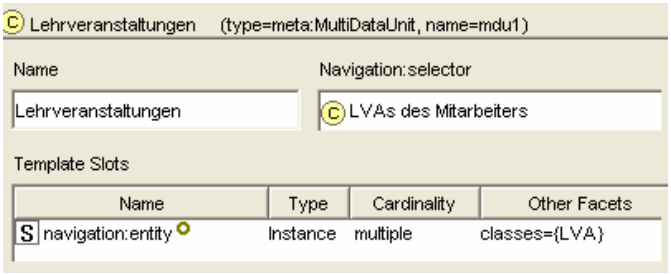


Abbildung 4-10 Klasse Multidataunit Lehrveranstaltungen in Protégé

Beispiel: *Abbildung 4-10 zeigt das Ergebnis der Übersetzung der Multidataunit Lehrveranstaltungen. Die Klasse besitzt einen Wert für den Slot selector und überschreibt den Slot entity.*

Zur Transformation der Dokumente beinhaltet das XSLT Stylesheet Template-Regeln für die Elemente in XML. Diese übersetzen die Elemente in Klassen, legen die Slotwerte der Klassen fest und überschreiben Slots der Superklasse.

Ein Template besteht aus folgenden Anweisungen:

1. Erzeugen einer Instanz der Metaklasse.
2. Festlegen der Slotwerte, welche Attributwerten in XML entsprechen.
3. Definieren der Superklasse.
4. Festlegen der Werte jener Slots, welche in der Metaklasse definiert wurden.
5. Überschreiben der Slots der Superklasse, welche Beziehungen zu Modellinstanzen ausdrücken. Durch Überschreiben können Eigenschaften eines Slots (z.B. der Wertebereich) eingeschränkt werden.
6. Aufrufen der Templates der untergeordneten Elemente, um für diese Elemente Klassen zu erzeugen und Slotwerte zu bestimmen.

Im Folgenden wird die Übersetzung der Modellelemente anhand des Beispiels der Multidataunit Lehrveranstaltungen näher betrachtet. Das Template zur Transformation einer Multidataunit besteht aus folgenden Anweisungen:

```
   <xsl:template match="MULTIDATAUNIT">
1  <xsl:element name="meta:MultiDataUnit"
                  use-attribute-sets="generalAttributes"> a

     <!--Attribute distinct-->
     <xsl:attribute name="navigation:distinct">
2       <xsl:choose>
           <xsl:when test="@distinct='yes'">true</xsl:when>
           <xsl:otherwise>false</xsl:otherwise>
        </xsl:choose>
     </xsl:attribute>

3    <rdfs:subClassOf rdf:resource="&navigation;MultiDataUnit"/>

     <!-- value of slot selector-->
4    <xsl:apply-templates select="SELECTOR" mode="slot"/> b
     …
   </xsl:element>
```

```
    <!-- override slot entity-->
    <xsl:if test="@entity">
      <xsl:call-template name="OverrideEntity"/>  c
5   </xsl:if>

    <!-- override slot link-->
    <xsl:apply-templates select="LINK" mode="OverridingProperty"/>

    <!-- apply templates of child elements-->
6   <xsl:apply-templates/>
  </xsl:template>
```

Dieses Template ruft ein Attribute-Set (a) sowie die Templates Selector (b) und OverrideEntity (c) auf.

Das Attribute-Set „generalAttributes“ (a) gruppiert wiederkehrende Attributdefinitionen. Diese umfassen das Festlegen des URIs sowie der Werte für die Slots id und name und das Übersetzen von Kommentaren.

```
<xsl:attribute-set name="generalAttributes">
  <xsl:attribute name="rdf:about">&webml;<xsl:value-of select="@id"/>
  </xsl:attribute>
  <xsl:attribute name="webml:id"><xsl:value-of select="@id"/></xsl:attribute>
  <xsl:attribute name="webml:name"><xsl:value-of select="@name"/>
  </xsl:attribute>
  <xsl:attribute name="rdfs:label"><xsl:value-of select="@name"/></xsl:attribute>
  <xsl:attribute name="rdfs:comment"><xsl:value-of select="COMMENT"/>
  </xsl:attribute>
</xsl:attribute-set>
```

Das Übersetzen der Beziehungen ist in eigenen Template-Regeln definiert, um die Regeln zur Transformation nur einmal festlegen zu müssen. Um eine Beziehung zu einem verschachtelten Element zu übersetzen, wird das Template des Elements mit Angabe des „mode“-Attributs aufgerufen. Ein Template mit mode=“slot“ erzeugt einen Slotwert (z.B. Wert für Slot selector), ein Template mit mode=“OverridingProperty“ überschreibt einen Slot (z.B. Überschreiben des Slots link). Folgendes Template legt den Wert der Beziehung zu einem Selector fest (b).

```
<xsl:template match="SELECTOR" mode="slot">
  <xsl:element name="navigation:selector">
    <xsl:attribute name="rdf:resource">&webml;<xsl:value-of
      select="../@id"/>_selector</xsl:attribute>
  </xsl:element>
</xsl:template>
```

Das Template „OverrideEntity“ (c) überschreibt den Slot entity, um z.B. für die Multidataunit Lehrveranstaltungen anzugeben, dass erlaubte Werte Instanzen der Klasse LVA sind. Im endgültigen Stylesheet definiert ein parametrisiertes Template das Überschreiben von Slots, um die Vorgehensweise zum Überschreiben nicht für jeden Slot neu angeben zu müssen.

```
<xsl:template name="OverrideEntity">
  <xsl:element name="a:OverridingProperty">
    <xsl:attribute name="rdf:about">&a;<xsl:value-of
      select="@id"/>_entity</xsl:attribute>
    <xsl:element name="a:domain">
```

```
            <xsl:attribute name="rdf:resource">&webml;<xsl:value-of
               select="@id"/></xsl:attribute>
         </xsl:element>
         <xsl:element name="rdfs:range">
            <xsl:attribute name="rdf:resource">&webml;<xsl:value-of
               select="@entity"/></xsl:attribute>
         </xsl:element>
         <a:overriddenProperty rdf:resource="&navigation;entity"/>
      </xsl:element>
   </xsl:template>
```

Die beschriebenen Template-Regeln übersetzen Multidataunits von XML in RDFS.

```
<MULTIDATAUNIT distinct="no" entity="ent3" id="mdu1"
      name="Lehrveranstaltungen">
   <SELECTOR defaultPolicy="fill">
      <SELECTORCONDITION id="cond1" name="LVAs des Mitarbeiters"
         predicate="in" relationship="rel2" type="required"/>
   </SELECTOR>
</MULTIDATAUNIT>
```

Beispiel: *Wendet man die Template-Regeln auf die Multidataunit Lehrveranstaltungen an, wird eine Klasse mit Werten für die Slots erzeugt und der Slot entity für diese Klasse überschrieben. Abbildung 4-10 zeigt das Ergebnis der Transformation in Protégé. In RDFS wird die Klasse folgendermaßen repräsentiert:*

```
1  <meta:MultiDataUnit rdf:about="&webml;mdu1"
       webml:id="mdu1"
       webml:name="Lehrveranstaltungen"
2      navigation:distinct="false"
       rdfs:label="Lehrveranstaltungen“
       rdfs:comment="">
3     <rdfs:subClassOf rdf:resource="&navigation;MultiDataUnit"/>
4     <navigation:selector rdf:resource="&webml;mdu1_selector"/>
   </meta:MultiDataUnit>

   <a:OverridingProperty rdf:about="&a;mdu1_entity">
      <rdfs:range rdf:resource="&webml;ent4"/>
5     <a:domain rdf:resource="&webml;mdu1"/>
      <a:overriddenProperty rdf:resource="&navigation;entity"/>
   </a:OverridingProperty>
```

4.4.3 Instanzebene

Im Gegensatz zu WebML ermöglicht Protégé die Abbildung von Instanzen der Modelle. Solche Instanzen der auf Modellebene erzeugten Klassen entsprechen konkreten Webseiten und werden in RDF repräsentiert.

Beispiel: *Die LVA „Datenbanken“ ist eine Instanz der Klasse LVA, welche Werte für die Attribute und eine Beziehung zum Mitarbeiter Huber besitzt. Die Klassennamen entsprechen den URIs, welche auf Modellebene bei der Erzeugung der Klassen angegeben wurden (z.B. ent4 für Klasse LVA).*

```
<webml:ent4 rdf:about="&webml;WebML_Instance_1"
    webml:OID="1"
    webml:att15="Datenbanken"
    webml:att16="2">
   <webml:rel10 rdf:resource="&webml;WebML_Instance_2"/>
</webml:ent4>
```

Beispiel: *Die Mitarbeiterseite Huber ist eine Instanz der Klasse Mitarbeiterseite und zeigt u.a. die Lehrveranstaltungen des Mitarbeiters an. Hierfür wird eine Instanz der Multidataunit Lehrveranstaltungen erzeugt, welche die LVA „Datenbanken" referenziert (vgl. Abbildung 4-11).*

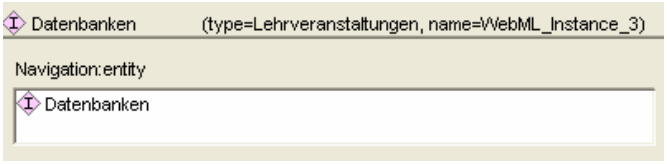


Abbildung 4-11 Instanz der Multidataunit Lehrveranstaltungen in Protégé

Die Instanz der Abbildung 4-11 wird in RDF folgendermaßen repräsentiert:

```
<webml:mdu1 rdf:about="&webml;WebML_Instance_3">
   <navigation:entity rdf:resource="&webml;WebML_Instance_1"/>
</webml:mdu1>
```

Da in Protégé Instanzen der Modelle abgebildet werden können, ermöglicht Protégé prinzipiell die Überprüfung dynamischer Constraints zur Laufzeit. Jedoch unterstützt Protégé nicht die automatische Generierung dieser Instanzen. Werkzeuge, welche Webseiten aus Modellen erstellen, bilden Instanzen üblicherweise in Technologien wie J2EE oder .NET ab. Um Instanzen in Protégé überprüfen zu können, müssten diese daher vom Benutzer selbst oder von Technologien erzeugt werden. Aus diesem Grund werden dynamische Constraints in dieser Arbeit nicht näher betrachtet.

4.5 Generierung und Überprüfung der Constraints

Die Überprüfung der Vorgaben gegen Modelle setzt voraus, dass die Vorgaben in Constraints ausgedrückt werden. Um aus den Vorgaben solche Constraints zu generieren, werden die Vorgaben modelliert und anschließend in RDFS übersetzt. Da die erzeugte RDFS-Repräsentation die Vorgaben in Form einer Ontologie repräsentiert, können Inferenzmechanismen Regeln auf das Vorgabenmodell ausführen, welche die Constraints erzeugen. Anschließend sind die Constraints gegen ein Modell der Web-Anwendung zu überprüfen. Hierfür wird dieses Modell ebenfalls in RDFS übersetzt, um mit Hilfe von Inferenzmechanismen die Einhaltung der Constraints feststellen zu können (vgl. Unterkapitel 4.2).

Dieses Unterkapitel betrachtet Anforderungen hinsichtlich der Constraints und gibt einen Überblick über die Constraintsprache PAL. Die Generierung und die Überprüfung der Constraints werden ebenso behandelt wie die Erweiterung der Vorgaben und die Visualisierung der Abhängigkeiten zwischen Constraints.

4.5.1 Anforderungen

Bei der Übersetzung der Vorgaben in Constraints wird für jede Eigenschaft bzw. Beziehung eines Elements des Vorgabenmodells ein eigener Constraint erzeugt. Eine 1:1-Abbildung hat den Vorteil, dass Verletzungen der Constraints einfach zugeordnet werden können. Ist ein Constraint verletzt, besitzt das Modell die entsprechende Eigenschaft nicht. Würden hingegen mehrere Eigenschaften eines Modellelements in einem Constraint überprüft werden (n:1 bzw. n:m), wäre bei Nichterfüllung des Constraints nicht sofort ersichtlich, welche dieser Eigenschaften das Modell verletzt. In diesem Fall müsste der Benutzer selbst feststellen, welche der Eigenschaften im Modell erfüllt bzw. nicht erfüllt sind. Eine 1:n-Abbildung ist möglich und bedeutet, dass zu einer Eigenschaft mehrere Constraints definiert werden. Diese Constraints können zu einem übergeordneten Constraint zusammengefasst werden (vgl. Anhang A.4, Prädikat evaluate). In dieser Arbeit wird eine 1:n-Abbildung verwendet, wobei im Allgemeinen für jede Eigenschaft ein Constraint definiert wird.

Um Eigenschaften bzw. Beziehungen eines Modellelements unabhängig voneinander überprüfen zu können, ist eine eindeutige Identifikation des Modellelements notwendig. Bezogen auf das Beispielprojekt gilt, dass Modellelemente des Typs Entity, Relationship, Siteview, Page sowie Units (z.B. Multidataunit) eindeutig identifizierbar sein müssen. WebML definiert bereits für jedes Modellelement eine ID. Diese wird allerdings von WebRatio automatisch vergeben, ist wenig aussagekräftig und kann vom Modellierer nicht geändert werden. Aus diesem Grund wird zur eindeutigen Identifikation der Name des Modellelements (bzw. bei Elementen des Typs Relationship der Rollenname) verwendet. Der Name muss innerhalb des jeweiligen Elementtyps eindeutig sein.

Beispiel: *Das Modellelement Multidataunit Lehrveranstaltungen wird mit dem Namen „Lehrveranstaltungen“ identifiziert. Um eine korrekte Überprüfung der Eigenschaften und Beziehungen dieser Multidataunit sicherzustellen, darf keine weitere Multidataunit diesen Namen aufweisen. Die Überprüfung dieser Multidataunit besteht u.a. aus folgenden Teilen:*

- *Es existiert eine Multidataunit namens „Lehrveranstaltungen“.*
- *Die Multidataunit „Lehrveranstaltungen“ besitzt einen Selector, welcher angibt, dass nur Lehrveranstaltungen des Mitarbeiters angezeigt werden.*
- *Die Multidataunit „Lehrveranstaltungen“ zeigt Instanzen der Entität namens „LVA“ an.*

Dieses Beispiel zeigt, dass Eigenschaften bzw. Beziehungen der Multidataunit in eigenen Constraints überprüft werden und die Multidataunit daher eindeutig identifizierbar sein muss. Hinsichtlich der Selector-Eigenschaft gilt, dass der Name

des Selectors beliebig sein kann. Der Name der Entität, deren Instanzen angezeigt werden, muss hingegen ebenfalls eindeutig sein.

Diese Vorgehensweise ermöglicht die Aufteilung der Modellüberprüfung in eine Menge von Constraints. Allerdings setzt die Überprüfung der Constraints voraus, dass die Namen der genannten Modellelemente im Vorgabenmodell und im Modell der Web-Anwendung übereinstimmen, da sie in den Constraints verwendet werden.

In WebRatio kann der Name der Modellelemente beliebig sein. Um eine korrekte Überprüfung sicherzustellen, werden daher in Protégé Constraints über die Modelle definiert, welche die Eindeutigkeit dieser Namen überprüfen.

4.5.2 Protégé Axiom Language

PAL (Protégé Axiom Language) [Crub02] ist eine Sprache zur Definition von Constraints und Abfragen über die Ontologie oder Inhalte der Ontologie, welche über ein Plugin in Protégé integriert wird. Zusätzlich beinhaltet PAL eine Inferenzmaschine, um Constraints überprüfen bzw. Abfragen durchführen zu können. Das Plugin beinhaltet einen Editor zur Definition der Constraints und zur Überprüfung ihrer Syntax. Protégé speichert die Constraints als Instanzen des Frames :PAL-CONSTRAINT.

PAL ist eine limitierte Prädikatenlogik, deren Syntax an KIF (Knowledge Interchange Format) [Gene92] angelehnt ist. Allerdings unterstützt PAL nicht alle KIF Konstante und Prädikate. Der Zweck von PAL als Constraintsprache ist weniger das Ableiten von neuem Wissen, also die Inferenz, sondern vor allem das Aufdecken von Inkonsistenzen. Die Inferenzmaschine prüft, welche Constraints das Modell erfüllt bzw. verletzt.

Ein Constraint in PAL besteht aus Variablenbereichsdefinitionen und logischen Aussagen, welche für die Variablen gelten müssen. Zusätzlich können zu jedem Constraint ein Name und eine Beschreibung angegeben werden.

4.5.2.1 Variablenbereichsdefinitionen

Der Bereich aller Variablen, welche in einem Constraint vorkommen, muss deklariert werden. „defrange" ist das Schlüsselwort für Variablenbereichs-definitionen.

Beispiel: *Die lokale Variable x erstreckt sich über alle Instanzen der Metaklasse Entity.*

```
(defrange ?x :FRAME meta:Entity)
```

4.5.2.2 Aussagen

Eine Aussage besteht aus einer Folge von Sätzen. Diese werden mit logischen Operatoren (and, or, =>, <=>, =, <, >) verknüpft. Aussagen beziehen sich beispielsweise auf Klassen, Instanzen oder Slotwerte, welche PAL zusammenfassend als Symbole bezeichnet.

Alle in einer Aussage vorkommenden Variablen müssen quantifiziert werden. Hierfür definiert PAL einen Allquantor (forall) und einen Existenzquantor (exists).

Beispiel: *Folgende Aussagen überprüfen, ob eine Entität mit dem Namen „LVA“ existiert.*

```
(defrange ?x :FRAME meta:Entity)
(exists ?x
   (= "LVA" (name ?x))
)
```

PAL beinhaltet einige vordefinierte Prädikate und Funktionen. Prädikate sind Beziehungen zwischen Symbolen. Jeder Slot kann als Prädikat verwendet werden, um seinen Wert zu überprüfen.

Beispiel: *Der Name einer Entität kann auch mit dem Prädikat (name ?x „LVA“) überprüft werden. Dieses Prädikat gibt wahr zurück, wenn die Entität x den Namen „LVA“ besitzt.*

PAL definiert weitere Prädikate, um zum Beispiel festzustellen, ob eine Klasse eine Subklasse einer anderen Klasse ist (subclass-of), ob ein Slot zu einer Klasse hinzugefügt wurde (slot-of) oder ob ein Frame eine Instanz einer Klasse ist (instance-of).

Funktionen nehmen eine Menge von Symbolen und produzieren ein neues Symbol. Jeder Slot kann als Funktion verwendet werden, um den Wert des Slots zu ermitteln.

Beispiel: *Die Funktion (name ?x) gibt den Wert des Slots „name“ des Frames x zurück.*

Vordefinierte Funktionen in PAL dienen vor allem der Typumwandlung, um z.B. ein Symbol in einen String zu übersetzen (coerce-to-string).

Beispiel: *Folgender Constraint prüft, ob die Multidataunit „Lehrveranstaltungen“ Instanzen der Entität „LVA“ anzeigt. Der Constraint beinhaltet die zwei lokalen Variablen mdu und ent. Die Variable mdu erstreckt sich über Instanzen der Metaklasse Multidataunit, die Variable ent über Instanzen der Metaklasse Entity. In der Aussage werden diese beiden Variablen mit dem Existenzquantor quantifiziert. Der Constraint überprüft, ob eine Multidataunit namens „Lehrveranstaltungen“ und eine Entität namens „LVA“ existieren, so dass der Slot entity der Multidataunit als Werte Instanzen der Entität annimmt.*

```
(defrange ?mdu :FRAME meta:MultiDataUnit)
(defrange ?ent :FRAME meta:Entity)

(exists ?mdu
   (and
      (="Lehrveranstaltungen"(name ?mdu))
      (exists ?ent
         (and
            (="LVA"(name ?ent))
            (is-allowed-class navigation:entity ?mdu ?ent)
         )
      )
   )
)
```

4.5.2.3 PAL Erweiterungen

PAL beinhaltet keine Prädikate oder Funktionen, um die Ontologie oder Inhalte der Ontologie zu verändern. Daher wurden im Rahmen dieser Arbeit Erweiterungen zu PAL implementiert, um Instanzen erzeugen und ihre Slotwerte festlegen zu können. Dies ist notwendig, damit Mappingregeln neue Constraints generieren können (vgl. Unterkapitel 4.5.3). Zusätzlich wurden Prädikate und Funktionen definiert, welche PAL nicht zur Verfügung stellt, jedoch zur Überprüfung der Modelle benötigt werden (vgl. Anhang A.4).

4.5.3 Generieren der Constraints

Die Generierung der Constraints erfolgt mit Hilfe von Mappingregeln, welche in PAL definiert werden. Diese dienen nicht der Überprüfung von Modellelementen, sondern führen für bestimmte Elemente Aktionen aus, um aus der RDFS Repräsentation des Vorgabenmodells Constraints zu generieren.

[Bowe00] verwendet Mappingregeln, um Information von einer Repräsentation in eine andere zu transformieren (z.B. zur Transformation von RDF Graphen). Eine Transformation zwischen zwei Dokumenten wird als Mapping M definiert. M besteht aus einer Menge von Mappingregeln T => T', wobei T und T' aus einer Menge von Prädikaten zusammengesetzt sind. Ist die linke Seite T der Regel erfüllt, wird die rechte Seite T' generiert.

Die Generierung der Constraints kann ebenfalls als Mapping M betrachtet werden, welches eine Menge von Mappingregeln beinhaltet. Eine Mappingregel besteht aus zwei Teilen. Die linke Seite einer Regel wählt die Modellelemente aus, für welche Constraints generiert werden. Die rechte Seite erzeugt für jedes dieser Modellelemente einen Constraint. Jede Seite besteht aus einer Menge von PAL Prädikaten und Funktionen.

Beispiel: *Folgende Mappingregel erzeugt für jede Entität einen Constraint, welcher überprüft, ob eine Entität mit dem jeweiligen Namen der Entität existiert. Diese*

Constraints haben den Zweck zu überprüfen, ob das Modell der Web-Anwendung jene Entitäten besitzt, welche im Vorgabenmodell festgelegt wurden.

```
1 (defrange ?ent :FRAME meta:Entity)
2 (forall ?ent
    (and
3     (create-instance
        :ELEMENT-EXISTS-CONSTRAINT
        (str-cat "con_" (:NAME ?ent) "_exists")
      )
4     (set-slot
        (str-cat "con_" (:NAME ?ent) "_exists")
       :PAL-RANGE
       "(defrange ?x :FRAME meta:Entity)"
      )
      (set-slot
        (str-cat "con_" (:NAME ?ent) "_exists")
        :PAL-STATEMENT
        (str-cat
5         "(exists ?x (="              ] a
            (quote (name ?ent))       ] b
            "(name ?x)))"             ] c
      )
    )
  )
```

Die Mappingregel setzt sich folgendermaßen zusammen:

1. *Definieren des Variablenbereichs:* Definiert die Variable ent, welche sich über alle Instanzen der Metaklasse Entität erstreckt. Es muss der Bereich aller Variablen deklariert werden, welche in der Regel vorkommen.
2. *Auswählen der Modellelemente,* für welche die Mappingregel Constraints erzeugt. In diesem Fall wird für jede Entität ein Constraint generiert, indem die Schritte 3-5 für jede Entität durchgeführt werden.
3. *Erzeugen eines neuen Constraints:* Erzeugen einer Instanz eines PAL-Constraints und Festlegen des URIs dieses Constraints. Als URI wird der URI des aktuellen Modellelements plus eine Bezeichnung für die Art des Constraints gewählt. Die Funktion str-cat verknüpft eine beliebige Anzahl von Strings.
4. *Definieren der Variablen des erzeugten Constraints:* Definiert die Variable x, welche sich über Instanzen der Metaklasse Entity erstreckt. Der Variablenbereich wird mit dem Slot :PAL-RANGE festgelegt.
5. *Aufstellen der Aussage des erzeugten Constraints:* Der generierte Constraint beinhaltet die Aussage, dass eine Entität mit dem Namen des aktuellen Modellelements existiert. Die Aussage wird mit dem Slot :PAL-STATEMENT definiert und besteht aus folgenden Teilen:
    a. Quantifizieren der Variable x mit dem Existenzquantor.
    b. Ermitteln des Namens des aktuellen Modellelements. Die Funktion quote setzt diesen Namen unter Anführungszeichen und liefert z.B. als Ergebnis „LVA“.

c. Definieren einer Funktion, welche den Namen der Entität x liefert.

Zusätzlich können zu jedem Constraint auch ein Name (Slot :PAL-NAME) und eine Beschreibung (Slot :PAL-DESCRIPTION) angegeben werden.

Beispiel: *Die Mappingregel erzeugt u.a. folgenden Constraint für das Beispielprojekt:*

```
(defrange ?x :FRAME meta:Entity)
(exists ?x
   (= "LVA" (name ?x))
)
```

Sowohl die Mappingregeln als auch die generierten Constraints können in Kategorien eingeteilt werden, indem Subklassen des Frames :PAL-CONSTRAINT angelegt werden. Beispielsweise ordnet obiges Beispiel die erzeugten Constraints in die Kategorie :ENTITY-EXISTS-CONSTRAINT ein.

Die Mappingregeln erzeugen die Constraints und legen fest, welche Angaben zu Modellelementen die Constraints überprüfen. Durch Bearbeiten der Protégé-Frames, welche die Regeln beinhalten, können jederzeit neue Regeln definiert oder bestehende Regeln geändert bzw. gelöscht werden.

4.5.4 Erweitern der Vorgaben

Nicht immer sind alle Vorgaben, welche die Web-Anwendung einhalten soll, mit der gewählten Modellierungssprache ausdrückbar. Zusätzliche Vorgaben können in Protégé mit Hilfe von Modellerweiterungen oder eigenen Constraints ergänzt werden.

4.5.4.1 Modellerweiterungen

Das Protégé-Modell ist im Allgemeinen mächtiger als die Modellierungssprache. Modellelemente, welche die Modellierungssprache nicht beinhaltet, können daher in Protégé ergänzt werden.

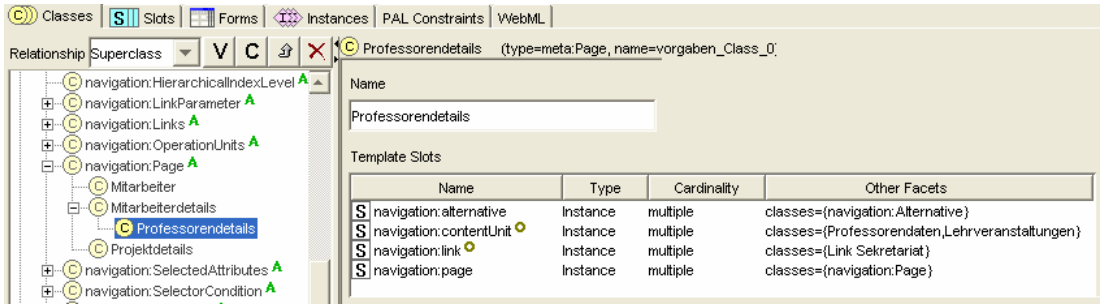


Abbildung 4-12 Modellerweiterung in Protégé

Beispiel: *In WebML ist es nicht möglich, eine Professorenseite als eine spezielle Mitarbeiterseite zu definieren und anzugeben, dass die Mitarbeiterseite, wenn es sich beim Mitarbeiter um einen Professor handelt, zusätzlich einen Link auf das*

Sekretariat besitzt. Abbildung 4-12 zeigt die Definition einer solchen Seite für Details über Professoren in Protégé. Die Seite ist eine Subklasse der Seite „Mitarbeiterdetails" und besitzt daher alle Eigenschaften der Mitarbeiterseite. Für die Professorenseite wird eingeschränkt, dass die Seite nur Daten über Professoren anzeigt (Slot contentUnit referenziert die Dataunit „Professorendaten"). Zusätzlich wird ein Link auf das Sekretariat definiert. Die Einhaltung solcher Modell-erweiterungen wird überprüft, indem Mappingregeln aus den Erweiterungen ebenfalls Constraints erzeugen.

4.5.4.2 Eigene Constraints

Nicht modellierbare Vorgaben können in Form von PAL Constraints ergänzt werden.

Beispiel: *In WebML ist kein Konstrukt vorhanden, um transitive Links zu definieren. Transitiv bedeutet, dass eine Information von einer Seite direkt oder über eine beliebige Anzahl von Links erreichbar ist. Beispielsweise soll der Besucher der Web-Anwendung ausgehend von der Mitarbeiterseite auf Informationen über Projekte zugreifen können. Für diese Art von Vorgabe kann der Benutzer einen eigenen PAL Constraint aufstellen, welcher überprüft, ob ein transitiver Link zwischen Mitarbeiterdaten und Projekten vorhanden ist.*

4.5.5 Abhängigkeiten zwischen Constraints

In Unterkapitel 4.5.1 wurde die Notwendigkeit erläutert, Eigenschaften bzw. Beziehungen eines Modellelements unabhängig voneinander überprüfen zu können und die Überprüfung in eine Menge von Constraints aufzuteilen. Diese Vorgehens-weise hat zur Folge, dass zwischen den Constraints verschiedene Abhängigkeiten bestehen und bestimmte Constraints nur erfüllt sind, wenn andere Constraints gelten. Denn Voraussetzung für die Existenz einer Eigenschaft oder einer Beziehung eines Elements ist, dass das Element selbst bzw. das Element, zu welchem die Beziehung besteht, existieren. Welche Abhängigkeiten zwischen den Constraints bestehen, kann in den Mappingregeln festgelegt werden. Hierzu beinhaltet jeder generierte Constraint zusätzlich einen Slot, welcher die abhängigen Constraints referenziert.

Beispiel: *Folgende Constraints sind voneinander abhängig (vgl. Abbildung 4-14):*

- *Existenz einer Multidataunit Lehrveranstaltungen (1).*
- *Existenz einer Entität LVA (2).*
- *Existenz einer Beziehung zwischen der Multidataunit Lehrveranstaltungen und der Entität LVA (3). Dieser Constraint ist von den beiden anderen abhängig, d.h., dass eine Voraussetzung für die Gültigkeit dieses Constraints ist, dass die Constraints 1 und 2 erfüllt sind.*

4.5.6 Überprüfen der Constraints

Um eine Web-Anwendung auf Einhaltung der Vorgaben zu überprüfen, ist eine formale Repräsentation der Web-Anwendung sowie der Vorgaben notwendig. Das Modell der Web-Anwendung wird zuerst in RDFS übersetzt, um die Modellelemente in Form einer Ontologie zu repräsentieren und Inferenzmechanismen anwenden zu können. Die RDFS Repräsentation des Modells sowie das Metaschema der Modellelemente werden anschließend in Protégé geladen und die Constraints, welche bereits mittels Mappingregeln aus den Vorgaben erzeugt wurden, werden importiert. Zur Überprüfung von PAL Constraints beinhaltet Protégé die PAL-Engine, welche für jeden Constraint feststellt, ob er erfüllt oder nicht erfüllt ist, und das Ergebnis der Überprüfung für den Benutzer kennzeichnet (vgl. Abbildung 4-13). Die Überprüfung der Constraints findet statt, wenn der Benutzer die PAL-Engine startet.

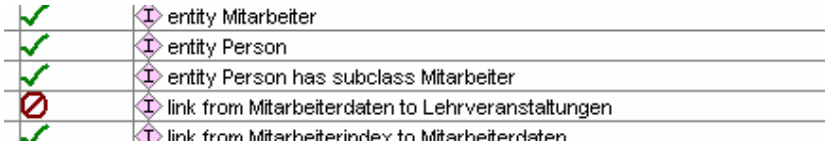


Abbildung 4-13 Überprüfung der Constraints

4.5.7 Visualisieren der Constraints

Die Visualisierung der Abhängigkeiten zwischen Constraints hat den Zweck, die Constraints übersichtlich darzustellen und die Beziehungen zwischen den Constraints zu verdeutlichen. Denn wurden die Vorgaben in Protégé erweitert (vgl. Unterkapitel 4.5.4), ist die WebML-Notation nicht vollständig.

Zur Visualisierung von Ontologien bzw. von Inhalten der Ontologien besitzt Protégé verschiedene Plugins wie OntoViz [Sint04], Jambalaya [Chis04] und TGViz [Alan03]. Diese unterstützen die Analyse und Bewertung von Ontologien und erhöhen das Verständnis über die Ontologie, indem sie Konzepte und ihre Beziehungen darstellen. OntoViz ist für große Ontologien wenig geeignet, da der erzeugte Graph statisch ist und nicht interaktiv durchlaufen werden kann. Jambalaya visualisiert die Ontologie mit ineinander verschachtelten Graphen, durch welche der Benutzer navigieren kann. Das Plugin TGViz erstellt einen interaktiven, dynamischen Graphen und wird in dieser Arbeit als Visualisierungsplugin verwendet, da es die Constraints am verständlichsten und übersichtlichsten darstellt.

TGViz ermöglicht die Visualisierung von Klassen und Instanzen, welche im Graphen als Knoten angezeigt werden. Beziehungen (Slots) werden als Kanten bzw. Links zwischen den Knoten dargestellt. Der Benutzer kann den Graphen auf Subgraphen reduzieren, um die Übersichtlichkeit zu erhöhen. Anschließend kann er interaktiv durch die verbundenen Subgraphen navigieren.

Die Constraints werden im Graphen durch Knoten, die Abhängigkeiten zwischen den Constraints durch Kanten repräsentiert.

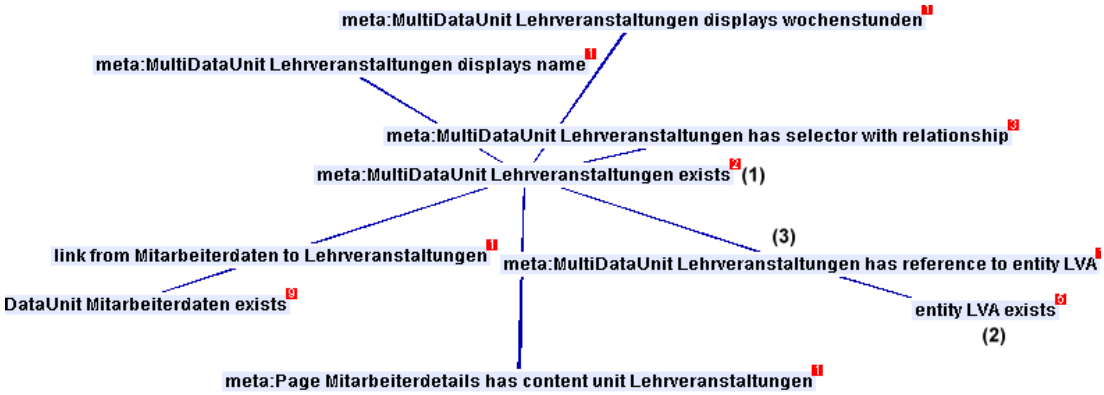


Abbildung 4-14 Visualisierung der Constraint-Abhängigkeiten

Beispiel: *Abbildung 4-14 zeigt einen Teil des erzeugten Graphen für das Beispielprojekt mit der in Unterkapitel 4.5.5 definierten Abhängigkeit.*

4.6 WebML Plugin

Im Rahmen dieser Arbeit wurde ein Plugin für Protégé entwickelt, um die Definition und Überprüfung von Vorgaben zu vereinfachen. Das Hinzufügen dieses so genannten WebML Plugins zu einem Protégé-Projekt bewirkt, dass das WebML Metaschema in das Projekt geladen wird. Weiters beinhaltet das Plugin die im Prozess (vgl. Unterkapitel 4.2) beschriebenen Funktionen:

- Übersetzen und Importieren von WebML Modellen. Der Benutzer wählt die XML-Datei des zu importierenden WebML Modells aus. Anschließend wird das Modell in RDFS übersetzt und in das Projekt geladen. Beinhaltet das Projekt bereits ein WebML Modell, wird bei erneutem Importieren das Modell ersetzt. Eventuelle Schemaerweiterungen bzw. Constraints bleiben erhalten, können jedoch ungültig werden (vgl. Anhang A.5.1.3).
- Ausführen der Mappingregeln bzw. Generieren der Constraints.
- Importieren und Exportieren der Constraints.

Zusammenfassend unterstützt das WebML Plugin bei der Definition von Vorgaben das Laden von WebML Modellen in Protégé, das Generieren von Constraints aus diesen Modellen und das Exportieren der Constraints. Um die Constraints zu überprüfen, ermöglicht das Plugin das Importieren des entsprechenden WebML Modells und der Constraints, welche gegen das Modell geprüft werden sollen.

Voraussetzungen und Einschränkungen, welche bei der Verwendung des WebML Plugins zu beachten sind, werden im Anhang A.6 dargestellt.

Kapitel 5

Zusammenfassung

Verteilte Organisationen streben einerseits ein einheitliches Erscheinungsbild ihrer Web-Anwendungen an, um dem Besucher der Web-Anwendung die Suche nach der gewünschten Information zu erleichtern. Andererseits sollen die einzelnen Organisationseinheiten ihre Web-Anwendung selbst verwalten können. Um diese beiden Anliegen in Einklang zu bringen, ist es gerade für verteilte Organisationen wünschenswert, Vorgaben für Web-Anwendungen definieren zu können, sie für einzelne Unternehmensbereiche zu erweitern und ihre Einhaltung automatisch zu überprüfen.

In dieser Arbeit wurde gezeigt, dass sich zur Definition der Vorgaben Modellierungsmethoden eignen, da diese eine grafische Notation der Vorgaben ermöglichen. Beispiele für solche Methoden sind WebML, die objektorientierten Methoden OOHDM, UWE und OO-H sowie die ontologiebasierten Methoden OntoWebber und OntoWeaver. Diese Modellierungsmethoden trennen die Modellierung einer Web-Anwendung nach Content, Hypertext und Präsentation. Der Content beschreibt den Inhalt der Web-Anwendung, der Hypertext den Aufbau der Seiten und die Navigation zwischen den Seiten, und die Präsentation legt das Layout der Seiten fest. Vorgaben können daher auf diesen drei Ebenen definiert werden.

Die Überprüfung der Vorgaben kann zum Zeitpunkt des Entwurfs gegen ein Modell der Web-Anwendung oder zur Laufzeit gegen Webseiten erfolgen. In dieser Arbeit wurde die Überprüfung gegen ein Modell der Anwendung betrachtet, welches analog zu den Vorgaben in der gewählten Modellierungsmethode modelliert wird. Die Überprüfung gegen ein Modell setzt voraus, dass die Webseiten aus dem Modell

generiert werden. Die Webseiten erfüllen daher die Vorgaben, sobald das Modell die Vorgaben umsetzt.

Um die Vorgaben automatisch gegen das Modell überprüfen zu können, bewährten sich Constraints, welche die Vorgaben in einer logikbasierten Sprache ausdrücken. Zur Übersetzung des Vorgabenmodells in Constraints sowie zur Überprüfung der Constraints werden Inferenzmechanismen eingesetzt. Diese erlauben einerseits das Ausführen von Regeln gegen das Vorgabenmodell, um aus dem Modell Constraints zu generieren, andererseits das Überprüfen der Constraints gegen das Modell der Web-Anwendung. Zur Repräsentation der Modelle bewährten sich Ontologien, welche Konzepte und ihre Beziehungen explizit definieren. Ontologien haben den Vorteil, dass Inferenzmechanismen Schlussfolgerungen über die Ontologie oder ihre Inhalte ziehen können.

Hinsichtlich der Abbildung einer Web-Anwendung erwies sich eine Gliederung in drei Ebenen, der Meta-, Modell- und Instanzebene, als zweckmäßig. Die Metaebene definiert die Modellierungskonstrukte, die Modellebene die Web-Anwendungsmodelle und die Instanzebene die Webseiten.

Zur Definition und Überprüfung von Vorgaben sind folglich eine Modellierungsmethode, eine Ontologiesprache, welche die Modelle repräsentiert, eine Constraint- bzw. Regelsprache und Inferenzmechanismen notwendig. Als Modellierungsmethode wurde in dieser Arbeit WebML in Verbindung mit dem Werkzeug WebRatio gewählt. Da WebML kein explizites Präsentationsmodell besitzt, wurden lediglich Vorgaben bezüglich des Content und des Hypertexts betrachtet. WebML repräsentiert die Modelle in XML. Diese XML Repräsentation ist für Inferenzmechanismen nicht ausreichend, da XML lediglich die Struktur der Dokumente beschreibt und keine explizite Semantik besitzt. Daher müssen die Modelle zuerst in eine Ontologiesprache übersetzt werden. Als Ontologiesprache wurde in dieser Arbeit RDFS und als Ontologieeditor Protégé gewählt. Protégé beinhaltet die Protégé Axiom Language (PAL), welche als Constraint- bzw. Regelsprache und als Inferenzmaschine zur Generierung und Überprüfung der Constraints eingesetzt wird.

Da die gewählte Modellierungsmethode nicht immer alle Vorgaben ausdrücken kann, können Vorgaben in Protégé als Modellerweiterungen oder in Form von Constraints hinzugefügt werden. Außerdem beinhaltet Protégé Visualisierungsplugins, welche eingesetzt werden, um Abhängigkeiten zwischen Constraints darzustellen.

Im Gegensatz zu bestehenden Ansätzen ist dieser Ansatz darauf ausgelegt, Vorgaben auf allen drei Ebenen (Content, Hypertext und Präsentation) zu unterstützen, die Überprüfung gegen ein Modell der Web-Anwendung oder gegen konkrete Webseiten zu ermöglichen, und die Vorgaben soweit wie möglich in einer grafischen Notation

festzulegen. Diese Arbeit behandelt die Definition von Vorgaben bezüglich des Content und des Hypertexts mit einer Modellierungsmethode, den Einsatz von Ontologien zur Generierung der Constraints aus dem Vorgabenmodell bzw. zur Überprüfung der Constraints gegen das Modell der Web-Anwendung sowie die Erweiterung und Visualisierung der Constraints. Zur Unterstützung dieser Schritte wurde ein Plugin für Protégé entwickelt.

Der in dieser Arbeit vorgestellte Ansatz kann in zukünftigen Entwicklungen dahingehend erweitert werden, dass Abhängigkeiten zwischen den Constraints bei der Überprüfung berücksichtigt werden. Bei der Verletzung eines Constraints können beispielsweise abhängige Constraints nicht erfüllt sein und müssen daher nicht überprüft werden. Die Überprüfung der Constraints kann auch um Prioritäten und das Gewichten von Constraints ergänzt werden, um Konflikte zwischen Constraints aufzulösen [Born92]. Mögliche Erweiterungen sind auch das Einbeziehen von Präsentationsvorgaben sowie die Umsetzung von Constraints zur Laufzeit.

Literaturverzeichnis

[Alan03] Alani, H.: *TGVizTab: An Ontology Visualisation Extension for Protégé.* Proceedings of Knowledge Capture (K-Cap'03), Workshop on Visualization Information in Knowledge Engineering, Sanibel Island, Florida, USA, October 2003.

[Alpu04] Alpuente, M.; Ballis, D.; Falaschi, M.: *A Rewriting-based Framework for Web sites Verification.* Proceedings of the 5th International Workshop on Rule-based Programming (RULE 04), Aachen, Germany, June 2004.

[Bare00] Baresi, L.; Garzotto, F.; Paolini, P.: *From Web Sites to Web Applications: New Issues for Conceptual Modeling.* Proceedings of the Workshop on Conceptual Modeling and the Web (ER2000), Salt Lake City, USA, October 2000.

[Beer02] Beer, W.; Birngruber, D.; Mössenböck, H.; Wöß, A.: *Die .NET-Technologie. Grundlagen und Anwendungsprogrammierung.* dpunkt, 2002.

[Bern01] Berners-Lee, T.; Hendler, J.; Lassila, O.: *The Semantic Web.* Scientific American 284 (5), May 2001, pp. 34-43.

[Bodo04] Bodoff, S.; Armstrong, E.; Ball, J.: *The J2EE Tutorial*, 2. Edition. Addison-Wesley 2004.

[Booc99] Booch, G.; Jacobson, L.; Rumbaugh, J.: *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[Born92] Borning, A.; Freeman-Benson, B.; Wilson, M.: *Constraint Hierarchies.* Lisp and Symbolic Computation 5 (3), September 1992, pp. 223-270.

[Born97] Borning, A.; Lin, R.; Marriott, K.: *Constraints for the Web.* ACM Multimedia 1997, Seattle, USA, November 1997, pp. 173-182.

[Bowe00] Bowser, S.; Delcambre, L.: *Representing and Transforming Model-based Information.* Proceedings of the International Workshop on the Semantic Web (SemWeb) at the 4th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2000), Lisbon, Portugal, September 2000.

[Cali02] Calì, A.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.: *A Formal Framework for Reasoning on UML Class Diagrams.* Proceedings of the 13th International Symposium on Methodologies for Intelligent Systems (ISMIS 02), Lyon, France, June 2002.

[Ceri00] Ceri, S.; Fraternali, P.; Bongio, A.: *Web Modeling Language (WebML): A Modeling Language for Designing Web Sites.* Computer Networks 33 (1-6), June 2000, pp. 137-157.

[Ceri02] Ceri, S.; Fraternali, P.; Matera, M.: *Conceptual Modeling of Data-Intensive Web Applications.* IEEE Intelligent Systems, 6 (4), July-August 2002, pp. 20-30.

[Chen76] Chen, P.: *The Entity-Relationship Model. Towards a Unified View of Data.* ACM Transaction on Database Systems 1 (1), March 1976, pp. 9-36.

[Chis04] Chisel Group: *Jambalaya*, http://www.thechiselgroup.org/jambalaya, [letzter Besuch: 2004-12-16].

[Cona03] Conallen, J: *Building Web Applications with UML,* 2. Edition. Addison-Wesley, 2003.

[Corc03] Corcho, O.; Fernández-López, M.; Gómez-Pérez, A.: *Methodologies, Tools and Languages for Building Ontologies. Where is their meeting point?* Data & Knowledge Engineering 46 (1), July 2003, pp. 41-64.

[Crub02] Crubézy, M.: *The Protégé Axiom Language and Toolset.* April 2002, http://protege.stanford.edu/plugins/paltabs/pal-documentation/index.html, [letzter Besuch: 2004-12-16].

[Davi93] Davis, R.; Shrobe, H.; Szolowits, P.: *What is a Knowledge Representation?* AI Magazine 14 (1), 1993, pp. 17-33.

[Deck00] Decker, S.; Melnik, S.; van Harmelen, F.; Fensel, D.; Klein, M.; Broekstra, J.; Erdmann, M.; Horrocks, I.: *The Semantic Web: The Roles of XML and RDF.* IEEE Intelligent Systems 4 (5), September-October 2000, pp. 63-74.

[Desp04] Despeyroux, T.: *Practical Semantic Analysis of Web Sites and Documents.* Proceedings of the 13th World Wide Web Conference (WWW 04), New York, USA, May 2004.

[Fens03] Fensel, D.; Hendler, J.; Lieberman, H.; Wahlster, W.: *Spinning the Semantic Web. Bringing the Word Wide Web to Its Full Potential*. MIT Press, 2003, pp. 1-26.

[Fern99] Fernández, M.; Florescu, D.; Levy, A.; Suciu, D.: *Verifying Integrity Constraints on Web Sites.* Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 99), Stockholm, Sweden, July 1999.

[Frat98] Fraternali, P.; Paolini, P.: *A Conceptual Model and a Tool Environment for Developing More Scalable and Dynamic Web Applications.* Proceedings of the 6th International Conference on Extending Database Technology (EDBT 98), Valencia, Spain, March 1998.

[Garz95] Garzotto, F.; Mainetti, L.; Paolini, P.: *Hypermedia Design, Analysis, and Evaluation Issues.* Communications of the ACM 38 (8), August 1995, pp. 74-86.

[Gene92] Genesereth M.R.; Fikes R.E.: *Knowledge Interchange Format,* Version 3.0, Reference Manual. Technical Report, Logic-92-1, Computer Science Dept., Stanford University, 1992.

[Gome01] Gómez, J.; Cachero, C.; Pastor, O.: *On Conceptual Modeling of Device-Independent Web Applications: Towards a Web-Engineering Approach.* IEEE Multimedia 8 (2), April-June 2001, pp. 26–39.

[Gome02] Gómez, J.; Cachero, C.: OO-H: *Extending UML to Model Web Interfaces.* In: Information Modeling for Internet Applications, van Bommel, P. (Ed.), Idea Group Publications, 2002, pp. 144-173.

[Grub93] Gruber, T.: *A Translation Approach to Portable Ontology Specifications.* Knowledge Acquisition 5 (2), 1993, pp. 199-220.

[Guar95] Guarino, N.; Giaretta, P.: *Ontologies and Knowledge Bases: Towards a Terminological Clarification.* In: Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, Mars, N. (Ed.), ISO Press, 1995, pp. 25-32.

[Ha03] Ha, I.; Kang, B.: *Meta-Validation of UML Diagrams Using OCL Rules.* Proceedings of the International Conference on Software Engineering Research and Practice (SERP 03), Las Vegas, USA, June 2003.

[Harm99] van Harmelen, F.; van der Meer, J.: *WebMaster: Knowledge-based Verification of Web-pages.* Proceedings of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AEI 99), London, UK, April 1999.

[Hend00] Hendler, J.; McGuiness, D. L.: *The DARPA Agent Markup Language.* IEEE Intelligent Systems 15 (6), November 2000, pp. 67-73.

[Horr00] Horrocks, I.; Fensel, D.; Broekstra, J.; Decker, S.; Erdmann, M.; Goble, C.; van Harmelen, F.; Klein, M.; Staab, S.; Studer, R.; Motta, E.: *The Ontology Inference Layer OIL.* August 2000, http://www.ontoknowledge.org/oil/TR/oil.long.html, [letzter Besuch: 2004-12-16].

[Jin01] Jin, Y.; Decker, S.; Wiederhold, G.: *OntoWebber: Model-Driven Ontology-Based Web Site Management.* Proceedings of the 1st International Semantic Web Working Symposium (SWWS 01), Stanford, USA, July-August 2001.

[Jin02] Jin, Y.; Xu, S.; Decker, S.: *OntoWebber: A Novel Approach for Managing Data on the Web.* Proceedings of the 8th International Conference on Extending Database Technology (EDBT 02), Prague, Czech Republic, March 2002.

[Kapp04] Kappel, G.; Pröll, B.; Reich, S.; Retschitzegger, W.: *Web Engineering – Die Disziplin zur systematischen Entwicklung von Web-Anwendungen.* In: Web Engineering: Systematische Entwicklung von Web-Anwendungen, Kappel, G.; Pröll, B.; Reich, S.; Retschitzegger, W. (Eds.), dpunkt, 2004, pp. 1-28.

[Kife95] Kifer, M.; Lausen, G.; Wu, J.: *Logical Foundations of Object-oriented and Frame-based Languages.* Journal of the ACM 42 (4), July 1995, pp. 741-843.

[Klas89] Klas, W.; Neuhold, E.J.; Schrefl, M.: *Tailoring Object-Oriented Data Models through Metaclasses.* Proceedings of the Advanced Database System Symposium '89, Kyoto, Japan, December 1989, pp. 169-178.

[Koch01] Koch, N.; Kraus, A.; Hennicker, R.: *The Authoring Process of the UML-based Web Engineering Approach.* Proceedings of the 1st International Workshop on Web-Oriented Software Technology (IWWOST 01), Valencia, Spain, June 2001.

[Koch02] Koch, N.; Kraus, A.: *The Expressive Power of UML-based Web Engineering.* Proceedings of the 2nd International Workshop on Web-oriented Software Technology (IWWOST 02), Malaga, Spain, June 2002.

[Lass01] Lassila, O.; McGuinness, D. L.: *The Role of Frame-Based Representation on the Semantic Web.* Knowledge Systems Laboratory Report KSL-01-02. Stanford University. 2001

[Lei02] Lei, Y.; Motta, E.; Domingue, J.: *An Ontology-Driven Approach to Web Site Generation and Maintenance.* Proceedings of the 13th International Conference on Knowledge Engineering and Management (EKAW 02), Sigüenza, Spain, October 2002.

[Lei04] Lei, Y.; Motta, E.; Domingue, J.: *Modelling Data-Intensive Web Sites with OntoWeaver.* Proceedings of the International Workshop on Web Information System Modelling (WISM 04), Riga, Latvia, June 2004.

[Loom04] *Loom Project Homepage,* University of Southern California, Information Sciences Institute, http://www.isi.edu/isd/LOOM, [letzter Besuch: 2004-12-16].

[Merz01] Merz, S.: *Model Checking: A Tutorial Overview.* In: Modelling and Verification of Parallel Processes, Cassez, F.; Jard, C.; Rozoy, B.; Ryan, M. D. (Eds.), LNCS 2067, Springer, 2001, pp. 3-38.

[Noy01a] Noy, N.; Sintek, M.; Decker, S.; Crubézy, M.; Fergerson, R.; Musen, M.: *Creating Semantic Web Contents with Protégé-2000.* IEEE Intelligent Systems 48 (2), March-April 2001, pp. 60-71.

[Noy01b] Noy, N.; Fergerson, R.; Musen, M.: *The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility.* Stanford Medical Informatics, June 2001, http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-2000-0830.pdf, [letzter Besuch: 2004-12-16].

[Rous97] Rousset, M.: *Verifying the Web: A Position Statement.* Proceedings of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV 97), Leuven, Belgium, June 1997.

[Schw98] Schwabe, D.; Rossi, G.: *An Object Oriented Approach to Web-Based Application Design.* Theory and Practice of Object Systems 4 (4), October 1998, pp. 207-225.

[Schw04] Schwinger, W.; Koch, N.: *Modellierung von Web-Anwendungen.* In: Web Engineering: Systematische Entwicklung von Web-Anwendungen, Kappel, G.; Pröll, B.; Reich, S.; Retschitzegger, W. (Eds.), dpunkt, 2004, pp. 49-75.

[Sint04] Sintek, M.: *OntoViz Tab: Visualizing Protégé Ontologies.* http://protege.stanford.edu/plugins/ontoviz/ontoviz.html, [letzter Besuch: 2004-12-16].

[W3C99a] World Wide Web Consortium (W3C): *HTML 4.01 Specification,* December 1999, http://www.w3.org/TR/html4, [letzter Besuch: 2004-12-16].

[W3C01a] World Wide Web Consortium (W3C): *XML Schema,* http://www.w3.org/XML/Schema, [letzter Besuch: 2004-12-16].

[W3C01b] World Wide Web Consortium (W3C): *Extensible Stylesheet Language (XSL),* Version 1.0, October 2001, http://www.w3.org/TR/xsl, [letzter Besuch: 2004-12-16].

[W3C01c] World Wide Web Consortium (W3C): *DAML+OIL Reference Description,* March 2001, http://www.w3.org/TR/daml+oil-reference, [letzter Besuch: 2004-12-16].

[W3C04a] World Wide Web Consortium (W3C): *Extensible Markup Language (XML) 1.0 (Third Edition),* February 2004, http://www.w3.org/TR/REC-xml, [letzter Besuch: 2004-12-16].

[W3C04b] World Wide Web Consortium (W3C): *RDF Primer,* February 2004, http://www.w3.org/TR/rdf-primer, [letzter Besuch: 2004-12-16].

[W3C04c] World Wide Web Consortium (W3C): *Resource Description Framework (RDF): Concepts and Abstract Syntax,* February 2004, http://www.w3.org/TR/rdf-concepts, [letzter Besuch: 2004-12-16].

[W3C04d] World Wide Web Consortium (W3C): *RDF/XML Syntax Specification (Revised),* February 2004, http://www.w3.org/TR/rdf-syntax-grammar, [letzter Besuch: 2004-12-16].

[W3C04e] World Wide Web Consortium (W3C): *RDF Vocabulary Description Language 1.0: RDF Schema,* February 2004, http://www.w3.org/TR/rdf-schema, [letzter Besuch: 2004-12-16].

[W3C04f] World Wide Web Consortium (W3C): *OWL Web Ontology Language Overview,* February 2004, http://www.w3.org/TR/owl-features, [letzter Besuch: 2004-12-16].

[WebM03] Webml.org: *WebML DTD,* Version 3.0 rc11, January 2003, http://www.webml.org/webml/upload/ent17/2/webml_dtd.zip, [letzter Besuch: 2004-12-16].

[WebR02] WebRatio Team: *WebML User Guide,* Version 3.0. Politecnico di Milano, September 2002.

[Will03] Wills, G.; Woukeu, A.; Carr, L.; Hall, W.: *The Need for Deeper Design in a Semantic Web.* Proceedings of the 1st International Workshop on Hypermedia and the Semantic Web (HTSW 03), Nottingham, UK, August 2003.

Anhang

Der Anhang beinhaltet Details der Implementierung. In Unterkapitel A.1 werden die verwendeten Programmversionen aufgelistet. Unterkapitel A.2 betrachtet Änderungen, welche an der WebML DTD vorgenommen wurden. Auf Besonderheiten bei der Abbildung von WebML in Protégé wird in Unterkapitel A.3 eingegangen. Unterkapitel A.4 gibt einen Überblick über die Prädikate und Funktionen, um welche PAL im Rahmen dieser Arbeit erweitert wurde. Unterkapitel A.5 fasst die Schritte zur Definition und Überprüfung von Vorgaben anhand eines Beispiels zusammen. Abschließend werden in Unterkapitel A.6 Voraussetzungen und Einschränkungen, welche bei der Verwendung des WebML Plugins zu berücksichtigen sind, betrachtet.

A.1 Verwendete Programmversionen

- WebRatio Academic Edition 3.3 rev. 6
- WebML DTD 3.0 rc11 (8. Jänner 2003)
- Protégé 2.1.2 (Plugins PAL, RDF, TGViz)
- Java Runtime Environment 1.4.2_05
- Entwicklungsumgebung Eclipse 3.0.1

A.2 Änderungen der WebML DTD

Die von WebRatio erzeugte XML Repräsentation der WebML Modelle entspricht nicht vollständig der WebML DTD. Daher wurden an der WebML DTD einige Änderungen vorgenommen, um sie der verwendeten Version von WebRatio anzupassen:

- Das Element SENDEMAILUNIT wurde hinzugefügt.
- Die Kardinalität des untergeordneten Elements LINK wurde bei den Elementen GETUNIT und SCROLLERUNIT auf beliebig (*) geändert, da in WebRatio mehrere Links von diesen Elementen ausgehen können.
- Die XML-Entität WebMLTypes wurde um OID ergänzt. In WebRatio besitzt jede Entität im Strukturmodell das Attribut OID mit dem Typ OID, um Instanzen der Entität eindeutig zu identifizieren.

- Beim Element GLOBALPARAMETER wurde der Attributtyp des Attributs „type“ von CDATA auf %WebMLTypes; geändert.
- Der Attributtyp des Attributs „relationship“ wurde bei den Elementen CONNECTUNIT und DISCONNECTUNIT von CDATA auf IDREF geändert, da dieses Attribut auf das Element RELATIONSHIP verweist.

A.3 Besonderheiten bei der Abbildung von WebML in Protégé

Bei der Abbildung von WebML in Protégé (vgl. Unterkapitel 4.4) sind sowohl auf der Meta- als auch auf der Modellebene einige Besonderheiten zu berücksichtigen. Auf diese wird nachfolgend eingegangen.

A.3.1 Metaebene

Das Strukturmodell legt die Informationsstruktur fest, genauer gesagt die Entitäten, ihre Attribute und Beziehungen. Eine Entität ist mit einer Klasse in Protégé vergleichbar, welche analog zu Entitäten Instanzen besitzen kann. In Protégé wird eine Metaklasse Entity definiert, welche das Aussehen einer Entität festlegt. Einzelne Entitäten sind Instanzen dieser Metaklasse und daher auch Klassen im Sinne von RDFS. In RDFS ist es daher möglich, die Klassen in einer Vererbungshierarchie anzuordnen. Die DTD hingegen drückt Vererbung lediglich mit Hilfe eines Attributs aus, welches die Superentität referenziert.

In der DTD werden Attribute und Beziehungen einer Entität mit den untergeordneten Elementen ATTRIBUTE und RELATIONSHIP definiert. RDFS drückt Attribute und Beziehungen einer Klasse im Allgemeinen mit Slots aus. Damit die Slots dieselben Eigenschaften wie in WebML festlegen können, werden die Elemente ATTRIBUTE und RELATIONSHIP mit Metaslots abgebildet. Bei der Abbildung wird soweit wie möglich auf Protégé spezifische Eigenschaften zurückgegriffen. Beispielsweise drückt der Datentyp Symbol in Protégé benutzerdefinierte Datentypen für Attribute aus. Hinsichtlich der Beziehungen können die Protégé-Eigenschaften für die Definition des Slotwerts, für inverse Beziehungen und für Kardinalitätseinschränkungen wieder verwendet werden.

Benutzerdefinierte Datentypen werden in WebML mit den Elementen DOMAIN und DOMAINVALUE definiert. Attributdeklarationen können dann diese Datentypen referenzieren. Protégé bildet benutzerdefinierte Datentypen mit Hilfe des Datentyps Symbol ab. Dieser stellt sicher, dass auf Instanzebene Attribute nur die erlaubten Werte annehmen können. Allerdings ist die Definition der möglichen Werte nicht mehr global, sondern wird für jeden Slot festgelegt.

In WebML ist jedem Element das Element COMMENT untergeordnet, um Modellelementen Kommentare hinzufügen zu können. Für dieses Element wird

keine Metaklasse erstellt, sondern Kommentare werden in RDFS mit rdfs:comment abgebildet und in Protégé als Dokumentation angezeigt.

Das Strukturmodell von WebML beinhaltet auch die so genannte Metastruktur, welche WebRatio spezifische Entitäten enthält. Diese wurden nicht in Protégé übersetzt.

A.3.2 Modellebene

Bei folgenden Elementen des Struktur- bzw. des Hypertextmodells weicht die Übersetzung von der in Unterkapitel 4.4.2 beschriebenen Vorgehensweise ab.

A.3.2.1 Strukturmodell

Attribute von Entitäten werden in Instanzen des Metaslots meta:Attribute übersetzt. In WebML besitzt jede Entität das Attribut OID, welches Schlüssel für Instanzen der Entität ist. Dieses Attribut wird in RDFS in der Superklasse Entity definiert und muss daher nicht für jede Entität übersetzt werden. Ob eine Instanz einer Entität ein Attribut besitzen muss, ist in WebML nicht definierbar. In RDFS hingegen können Kardinalitäten zu Slots angegeben werden. Bei der Transformation wird definiert, dass alle Attribute optional sind. Für das Festlegen des Datentyps von Attributwerten gibt es in WebML eigene Datentypen, welche nicht mit jenen von Protégé übereinstimmen. Attribute in RDFS beinhalten den in WebML definierten Datentyp, welcher zusätzlich soweit wie möglich in den entsprechenden Protégé-Datentyp übersetzt wird. Benutzerdefinierte Datentypen werden in Protégé mit dem Datentyp Symbol ausgedrückt.

Für Beziehungen zwischen Entitäten werden in RDFS Instanzen des Metaslots meta:Relationship angelegt. Bestimmte Eigenschaften von Beziehungen werden in Protégé spezifische Eigenschaften übersetzt. Dies gilt für Kardinalitäts-einschränkungen (a:minCardinality bzw. a:maxCardinality), inverse Beziehungen (a:inverse) und für die Definition des Wertebereichs (rdfs:range) einer Beziehung. rdfs:domain gibt an, von welcher Entität die Beziehung ausgeht.

A.3.2.2 Hypertextmodell

WebML definiert nicht für jedes Element des Hypertextmodells eine ID. Da jedoch die ID bei der Übersetzung als URI verwendet wird, wird für diese Elemente (Displayattribute, Sortattribute, Selektoren) der URI im XSLT Stylesheet generiert.

Besitzt eine Dataunit in WebML das Element DISPLAYALL, bedeutet dies, dass die Dataunit alle Attribute der ausgewählten Entität anzeigt. Werden hingegen nur bestimmte Attribute bei der Modellierung ausgewählt, sind der Dataunit DISPLAYATTRIBUTE Elemente untergeordnet. Um in RDFS einheitlich die anzuzeigenden Attribute darzustellen, wird bei der Transformation das Element

DISPLAYALL als eine Menge von DISPLAYATTRIBUTE Elementen behandelt. Für jedes Attribut der Entität wird analog zu DISPLAYATTRIBUTE Elementen eine Beziehung zwischen der Dataunit und dem anzuzeigendem Attribut erstellt.

Einige Attribute in WebML können mittels des Attributtypen IDREFS mehrere Elemente referenzieren. Die IDs der Elemente werden in XML durch Leerzeichen getrennt. Um die Beziehung zu jedem Element zu übersetzen, besitzt das Stylesheet ein Template für einen Tokenizer, welcher rekursiv die IDs auflöst.

A.4 PAL Erweiterungen

PAL wurde um einige Prädikate bzw. Funktionen ergänzt, um Constraints generieren und die Modelle überprüfen zu können. Jedes Prädikat bzw. jede Funktion ist eine Java Klasse. Im Folgenden werden der Name, die Argumente (in Großbuchstaben) und eine Beschreibung für jede Erweiterung angegeben.

Zusätzliche Prädikate:

- *create-instance (CLASS, STRING):* Erzeugt eine neue Protégé-Instanz der angegebenen Klasse mit dem angegebenen Namen.
- *evaluate (INSTANCE):* Evaluiert einen PAL Constraint. Dieses Prädikat wird verwendet, um Constraints aus anderen Constraints zusammensetzen zu können.
- *is-allowed-class (SLOT, CLASS, CLASS):* Gibt an, ob der Slot einer Klasse eine bestimmte Klasse als erlaubten Wert besitzt. Das erste Argument ist der Slot, das zweite Argument die Klasse, welcher der Slot zugeordnet ist, und das dritte Argument gibt an, Instanzen welcher Klasse der Slot als Werte annehmen darf.
- *set-multiple-slot (STRING, SLOT, {ANY}):* Legt die Werte eines Slots eines Frames fest. Das erste Argument ist der URI des Frames, das zweite Argument der Slot und die folgenden Argumente sind die Slotwerte. Das Prädikat ist nur für Slots anwendbar, für welche mehrere Werte erlaubt sind.
- *set-slot (STRING, SLOT, ANY):* Setzt den Wert eines Slots eines Frames. Das erste Argument ist der URI des Frames, das zweite Argument der Slot und das dritte Argument der Slotwert.
- *transitive-link (CLASS, CLASS):* Gibt an, ob zwischen zwei Klassen ein transitiver Link existiert.

Zusätzliche Funktionen:

- *str-cat ({STRING}):* Verknüpft eine beliebige Anzahl von Strings und gibt den neuen String zurück.

- *number-of-allowed-slot-values (SLOT, CLASS):* Liefert die Anzahl der erlaubten Werte für den Slot einer bestimmten Klasse (z.B. Instanzen wie vieler Klassen der Slot als Werte annehmen darf).
- *quote (STRING):* Setzt einen String unter Anführungszeichen und gibt das Ergebnis als String zurück.
- *slot-value (SLOT, FRAME):* Ermittelt den Slotwert eines Frames als String.

A.5 Zusammenfassendes Beispiel

Im Folgenden werden die Schritte zur Definition und Überprüfung von Vorgaben zusammengefasst und anhand des Beispielprojekts erläutert. Die Durchführung dieser Schritte erfordert die Werkzeuge WebRatio und Protégé. Die Protégé-Plugins PAL, RDF, TGViz sowie das in dieser Arbeit entwickelte WebML Plugin für Protégé müssen installiert sein.

A.5.1 Definieren der Vorgaben

Um Vorgaben zu definieren, werden diese in WebML modelliert und anschließend in Protégé übersetzt, um aus den Vorgaben Constraints generieren zu können.

A.5.1.1 Modellieren der Vorgaben

Die Vorgaben werden in WebRatio modelliert. Aktuell werden Vorgaben des Struktur- und des Hypertextmodells unterstützt.

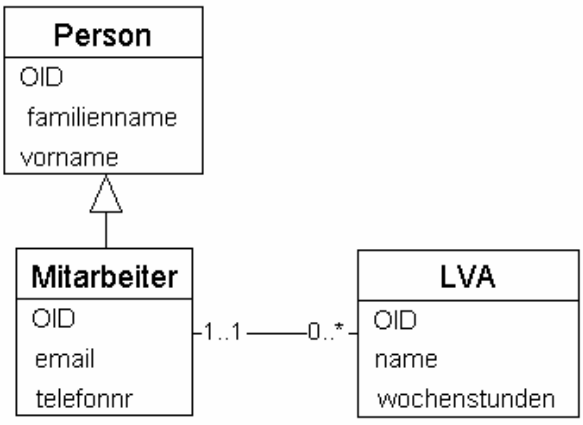


Abbildung A-1 Beispiel Vorgaben Strukturmodell

Abbildung A-1 zeigt mögliche Vorgaben bezüglich des Inhalts einer Web-Anwendung. Die Vorgaben legen fest, dass die Web-Anwendung Daten über Mitarbeiter (Name, Email-Adresse, Telefonnummer) sowie Daten über Lehrveranstaltungen (Name, Wochenstunden) anzeigt.

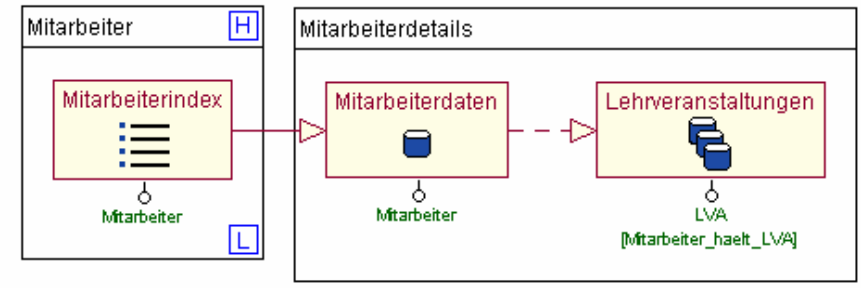


Abbildung A-2 Beispiel Vorgaben Hypertextmodell

Abbildung A-2 definiert Vorgaben des Hypertexts, welche festlegen, dass eine Seite mit Mitarbeiterdetails über einen Mitarbeiterindex erreichbar ist. Die Mitarbeiterdetails umfassen Daten über den gewählten Mitarbeiter sowie über seine Lehrveranstaltungen.

A.5.1.2 Anlegen eines neuen WebML Projekts

Um die Modelle in Protégé laden zu können, ist ein neues RDFS Projekt in Protégé anzulegen. Dieses Projekt muss dahingehend konfiguriert werden, dass die Plugins PAL Constraints Tab und WebML Tab zum Projekt hinzugefügt werden. Beim Speichern des Projekts ist darauf zu achten, dass als Namespace für das neue Projekt http://protege.stanford.edu/webml# gewählt wird. Das Hinzufügen des WebML Tabs bewirkt das Laden des Metaschemas, welches die Modellkonstrukte und ihre Beziehungen festlegt, sowie der Mappingregeln und der Modell-Constraints.

A.5.1.3 Importieren des WebML Modells in Protégé

Das WebML Plugin beinhaltet einen Menüpunkt zum Importieren von WebML Modellen. Der Benutzer wird aufgefordert, die XML Datei, in welcher das Modell repräsentiert ist, zu wählen. Wurde bereits ein Modell in das Projekt geladen, wird dieses ersetzt. Eventuelle Schemaerweiterungen und Constraints bleiben erhalten. Jedoch können Schemaerweiterungen ungültig werden, wenn sie ein Modellelement referenzieren, welches im neuen Modell nicht mehr existiert oder dessen ID sich geändert hat. Protégé erzeugt für Referenzen auf fehlende Elemente Klassen, welche nicht in die im Metaschema definierte Klassenhierarchie eingeordnet sind. Auf diese Weise werden ungültige Schemaerweiterungen gekennzeichnet.

A.5.1.4 Erweitern der Vorgaben

Dieser Schritt umfasst das Erweitern der Vorgaben mit Hilfe von Modellerweiterungen oder eigenen Constraints, da WebML nicht immer alle Vorgaben ausdrücken kann. Modellerweiterungen betreffen das Hinzufügen von Modellelementen, für welche WebML keine Konstrukte definiert. Abbildung A-3 definiert beispielsweise eine Professorenseite als eine spezielle Mitarbeiterseite, welche zusätzlich einen Link auf das Sekretariat besitzt.

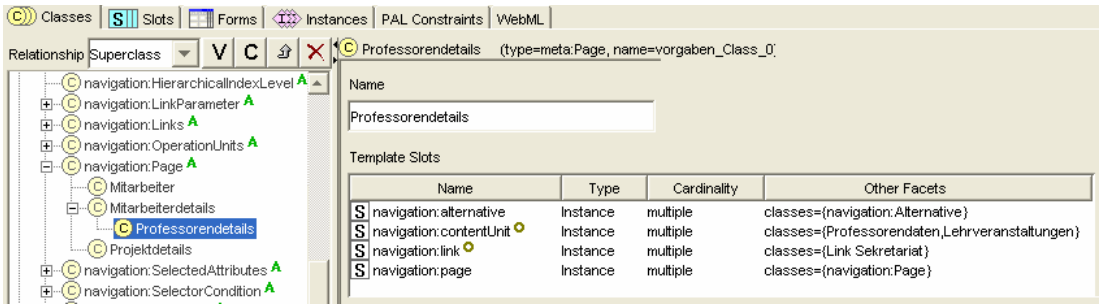


Abbildung A-3 Beispiel Modellerweiterung Professorenseite

Vorgaben können auch in Form von PAL Constraints als Instanzen der Klasse :USER-CONSTRAINT ergänzt werden. Folgender Constraint überprüft, ob ein transitiver Link zwischen den Mitarbeiterdaten und einem Index über die Projekte existiert. Der Constraint ist erfüllt, wenn ausgehend von den Mitarbeiterdaten ein Index über die Projekte entweder direkt oder über eine beliebige Anzahl von Links erreichbar ist.

```
(defrange ?mit :FRAME meta:DataUnit)
(defrange ?pro :FRAME meta:IndexUnit)
(defrange ?ent :FRAME meta:Entity)
(exists ?mit
   (and
      (="Mitarbeiterdaten" (name ?mit))
      (exists ?pro
         (and
            (exists ?ent
               (and
                  (="Projekt" (name ?ent))
                  (is-allowed-class navigation:entity ?pro ?ent)
                  (transitive-link ?mit ?pro)
               )
            )
         )
      )
   )
)
```

A.5.1.5 Erzeugen der Constraints

Um die Constraints aus dem Vorgabenmodell zu erzeugen, beinhaltet das WebML Plugin den Menüpunkt „Generate Constraints“. Wurden die Constraints bereits in einem früheren Schritt generiert, werden diese Constraints gelöscht und neu erzeugt.

A.5.1.6 Exportieren der Constraints

Das WebML Plugin unterstützt das Exportieren der generierten Constraints sowie der User-Constraints. Die Constraints werden in ein neues Protégé-Projekt exportiert, dessen Name und Pfad vom Benutzer anzugeben sind.

A.5.2 Überprüfen der Vorgaben

Um eine Web-Anwendung auf Einhaltung der Vorgaben zu überprüfen, wird sie in WebML modelliert und in Protégé gemeinsam mit dem Projekt, welches die Vorgaben in Form von PAL Constraints beinhaltet, (vgl. A.5.1.6) geladen. Anschließend kann mit Hilfe der PAL-Engine überprüft werden, welche Constraints die Web-Anwendung erfüllt bzw. verletzt.

A.5.2.1 Modellieren der Web-Anwendung

Die Web-Anwendung wird analog zu den Vorgaben in WebRatio modelliert.

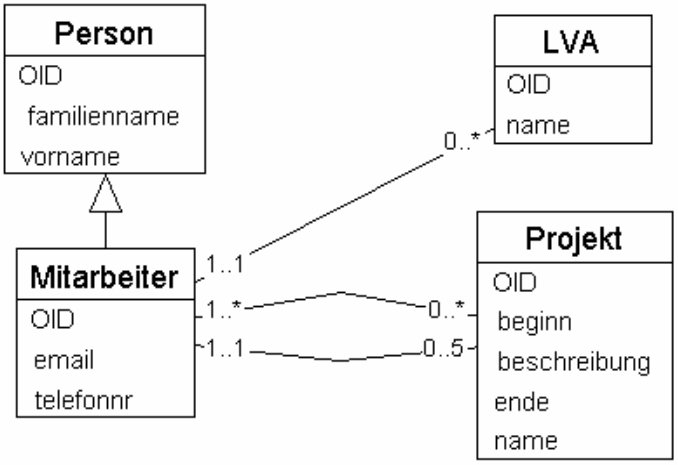


Abbildung A-4 Beispiel Web-Anwendung Strukturmodell

Abbildung A-4 zeigt ein Beispiel für ein Strukturmodell einer Web-Anwendung. Die Web-Anwendung besitzt zusätzlich zu den in den Vorgaben definierten Entitäten die Entität Projekt. Im Gegensatz zum Vorgabenmodell beinhaltet die Entität LVA jedoch nicht das Attribut „Wochenstunden“.

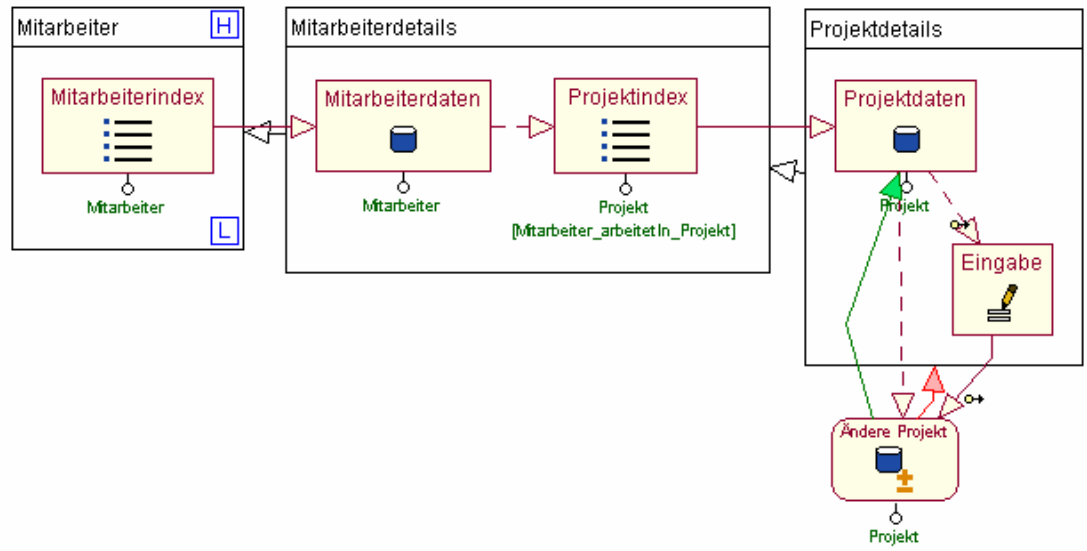


Abbildung A-5 Beispiel Web-Anwendung Hypertextmodell

Das Hypertextmodell der Beispielanwendung ist in Abbildung A-5 dargestellt. Die Mitarbeiterseite der Anwendung beinhaltet zusätzlich zu den Mitarbeiterdaten auch einen Index über die Projekte. Wählt der Benutzer der Web-Anwendung ein Projekt aus, gelangt er zu einer Seite mit Projektdetails, welche die Änderung der Projektdaten erlaubt. Die Mitarbeiterseite beinhaltet im Gegensatz zu den Vorgaben keine Daten über Lehrveranstaltungen.

A.5.2.2 Anlegen eines neuen WebML Projekts

Das Anlegen eines neuen Projekts funktioniert wie in A.5.1.2. beschrieben.

A.5.2.3 Importieren der Constraints

Das WebML Plugin beinhaltet einen Menüpunkt, um Constraints aus einem Projekt zu importieren. Wurden die Constraints bereits importiert, werden sie ersetzt. Beim Importieren ist darauf zu achten, dass das Projekt, welches die Constraints beinhaltet, im selben Verzeichnis wie das aktuelle WebML Projekt liegt.

A.5.2.4 Evaluieren der Constraints

Die Evaluierung der Constraints erfolgt mit Hilfe des PAL Constraints Tab. Der Benutzer wählt die zu evaluierenden Constraints aus (z.B. Instanzen von :MODEL-CONSTRAINT für Constraints über das Modell, Instanzen von :USER-CONSTRAINT für eigene Constraints, Instanzen von :WEBML-CONSTRAINT für generierte Constraints). Nach Evaluierung der Constraints gibt die PAL-Engine für jeden Constraint an, ob er erfüllt oder nicht erfüllt ist (vgl. Abbildung A-6).

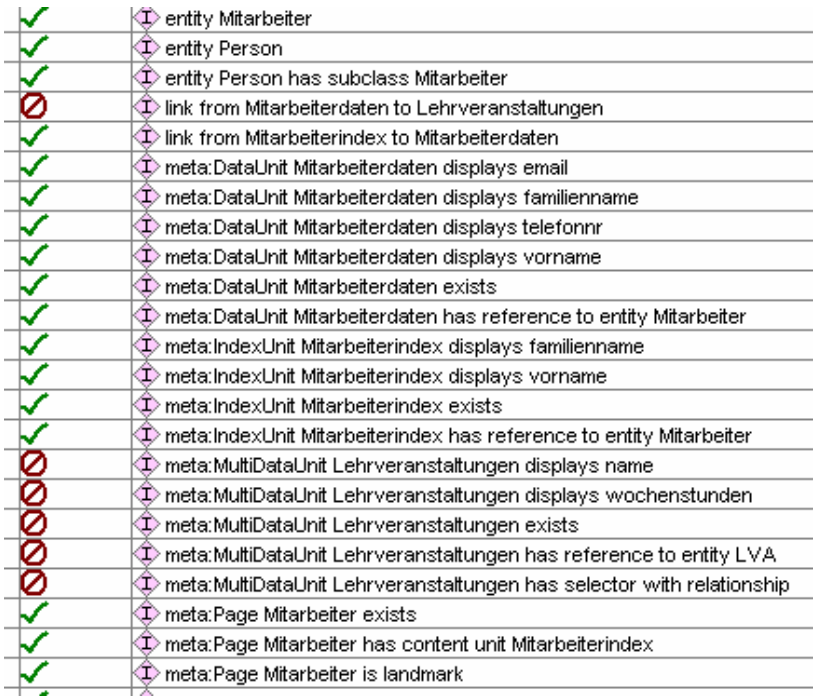


Abbildung A-6 Beispiel Überprüfung der Constraints

Bezogen auf die Beispielanwendung sind folgende Vorgaben nicht erfüllt:

- Die Entität LVA besitzt das Attribut Wochenstunden.
- Die Seite Mitarbeiterdetails beinhaltet die Multidataunit Lehrveranstaltungen.
- Die Dataunit Mitarbeiterdaten verlinkt auf die Multidataunit Lehrveranstaltungen.
- Die Anwendung besitzt eine Multidataunit Lehrveranstaltungen, welche Name und Wochenstunden der Lehrveranstaltungen des aktuellen Mitarbeiters anzeigt.

A.5.3 Visualisieren der Constraint-Abhängigkeiten

Die Abhängigkeiten, welche zwischen den Constraints bestehen, werden beim Erzeugen der Constraints mit Hilfe des Slots ASSOCIATED-CONSTRAINT festgelegt. Um diese Abhängigkeiten zu visualisieren, ist das TGViz Tab dem Projekt hinzuzufügen und der Graph für Instanzen der Klasse :WEBML-CONSTRAINT zu erzeugen. Der Radius legt die Größe des Graphen fest. Das Drücken der rechten Maustaste über einem Knoten des Graphen öffnet ein Menü, welches es ermöglicht, den Knoten zu expandieren, zusammenzuklappen, zu verstecken oder anzuzeigen. Ein Doppelklick auf einen Knoten bewirkt, dass sich der Graph ausgehend von diesem Knoten neu aufbaut. Abbildung A-7 zeigt einen Ausschnitt des Graphen für das Beispielprojekt.

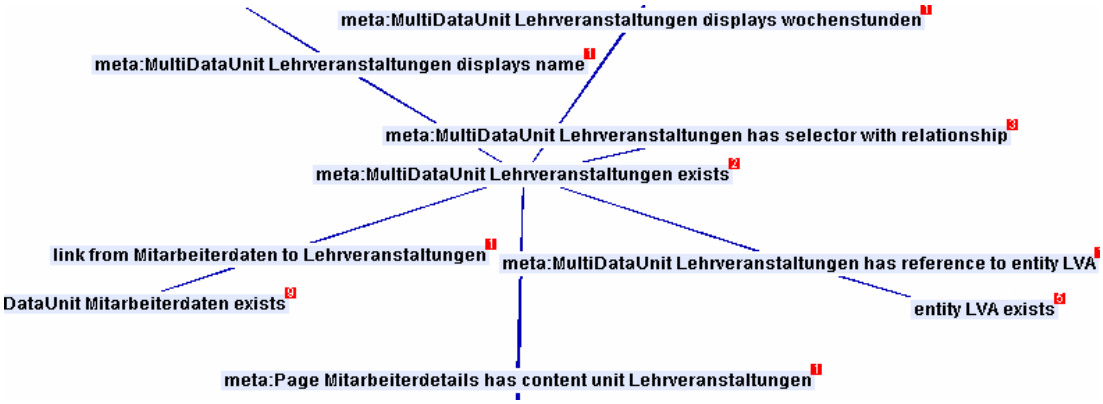


Abbildung A-7 Beispiel Visualisierung der Constraint-Abhängigkeiten

A.5.4 Mappingregeln

Das Metaschema des WebML Plugins für Protégé beinhaltet eine Reihe von Mappingregeln, welche Instanzen der Klasse :GENERATE-CONSTRAINT sind. Durch Bearbeiten dieser Instanzen können die Mappingregeln geändert oder gelöscht werden. Um neue Mappingregeln hinzuzufügen, sind neue Instanzen dieser Klasse anzulegen.

A.6 WebML Plugin

Im Rahmen dieser Arbeit wurde ein Plugin für Protégé entwickelt, welches die Definition von Vorgaben mit WebML und ihre Überprüfung gegen Web-Anwendungsmodelle unterstützt (vgl. Unterkapitel 4.6). Dieses Unterkapitel betrachtet Voraussetzungen und Einschränkungen bei der Verwendung des Plugins.

A.6.1 Voraussetzungen

Um das Plugin zu verwenden, sind eine Java Runtime Environment sowie die Werkzeuge Protégé und WebRatio notwendig. Hinsichtlich der Systemvoraussetzungen sind die Voraussetzungen von WebRatio bzw. Protégé zu beachten. Das Plugin wurde für folgende Versionen entwickelt:

- Java Runtime Environment 1.4.2_05.
- Protégé 2.1.2 mit Plugins für RDF, PAL, TGViz, WebML.
- WebRatio Academic Edition 3.3 rev. 6.

A.6.2 Einschränkungen

Folgende Einschränkungen sind bei der Definition und Überprüfung von Vorgaben zu beachten:

- Der Name der Modellelemente „Entity“, „Siteview“, „Page“, „Area“, „Alternative“ sowie der Content Units und Operation Units muss innerhalb des jeweiligen Elementtyps eindeutig sein. Für Elemente des Typs „Relationship“ gilt dies für den Rollennamen. Attribute müssen innerhalb der jeweiligen Entität einen eindeutigen Namen aufweisen. Diese Einschränkungen können mit Hilfe von vordefinierten Constraints, welche der Kategorie :MODEL-CONSTRAINT zugeordnet sind, überprüft werden.
- Für die oben genannten Modellelemente gilt, dass ihr Name im Vorgabenmodell und im Modell der Web-Anwendung übereinstimmen muss (vgl. Unterkapitel 4.5.1).
- Wird ein Modell in Protégé durch nochmaliges Importieren ersetzt, können Schemaerweiterungen oder eigene Constraints ungültig werden. Dies kann z.B. der Fall sein, wenn Modellelemente gelöscht wurden oder sich die ID von Modellelementen geändert hat (vgl. Unterkapitel A.5.1.3).
- Im Rahmen dieser Arbeit wurde eine Reihe von Mappingregeln zur Erzeugung von Constraints aus den Vorgaben definiert (vgl. Unterkapitel 4.5.3). Die Mappingregeln sind jedoch nicht vollständig und überprüfen nachfolgende Vorgaben bezüglich des Strukturmodells:
    - Existenz von Entitäten.
    - Vererbungsbeziehungen zwischen Entitäten.

- Attribute von Entitäten (Name und Attributtyp).
- Beziehungen zwischen Entitäten (Rollenname, Kardinalität).

Hinsichtlich des Hypertextmodells werden folgende Vorgaben überprüft:

- *Siteviews:* Es wird überprüft, ob die Siteviews die vorgegebenen Operation Units und Seiten beinhalten und welche Seite die Homepage einer Siteview ist.
- *Seiten:* Eine Seite muss die vorgegebenen Content Units beinhalten. Außerdem wird überprüft, ob dieselben Seiten vom Menü aus erreichbar sind, welche im Vorgabenmodell angegeben wurden.
- *Units:* Je nach Art der Unit wird überprüft, auf welche Entität sich die Unit bezieht, welche Attribute der Entität angezeigt werden (Displayattribute) und welche Instanzen der Entität ausgewählt werden (Selector).
- *Links* zwischen Seiten bzw. Units.

A.6.3 Anmerkungen

Bei der Verwendung des WebML Plugins für Protégé sind folgende Punkte zu berücksichtigen:

- Beim Anlegen eines neuen WebML Projekts werden mehrere Projektdateien automatisch generiert, welche im Verzeichnis des Projekts abgelegt werden. Diese Dateien beinhalten das Metaschema sowie das importierte WebML Modell. Daher dürfen diese Dateien nicht gelöscht werden.
- Zur Generierung und Überprüfung der Constraints wurde für WebML Projekte der Namespace http://protege.stanford.edu/webml# definiert.
- Sowohl das Metaschema, das importierte Modell als auch die Constraints werden nicht im WebML Projekt gespeichert, sondern als Projekt inkludiert. Diese inkludierten Projekte können im aktuellen WebML Projekt nicht geändert werden. Diese Einschränkung kann durch das Zusammenführen der Projekte aufgehoben werden.
- Das WebML Projekt besitzt Referenzen auf inkludierte Projekte. Wird das Projekt in ein anderes Verzeichnis verschoben, werden diese Referenzen ungültig.
- Projekte, welche in das WebML Projekt inkludiert werden, müssen im selben Verzeichnis wie das WebML Projekt liegen, sofern sie keine absoluten URIs auf die inkludierten Projektdateien besitzen. Beispielsweise gilt dies für das Projekt, welches beim Exportieren der Constraints erzeugt wird. Diese Einschränkung ist auf die Funktionsweise des Inkludierens von Projekten in Protégé zurückzuführen.