D I P L O M A R B E I T

# WebWidgets – Einfache Entwicklung guter Web-Oberflächen durch semantische Beschreibungen

von

Alexander Grosul

eingereicht am 31 August 2005 beim
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
der Universität Karlsruhe

Referent: Prof. Dr. Rudi Studer
Betreuer: Dipl.-Inform. Max Völkel

Anschrift:
Karlsbader Str. 10
76228 Karlsruhe

# WebWidgets –

## Einfache Entwicklung guter Web-Oberflächen durch semantische Beschreibungen

Diplomarbeit

Institut für
Angewandte Informatik und Formale Beschreibungsverfahren
Lehrstuhl Studer
Fakultät für Wirtschaftswissenschaften
Universität Karlsruhe (TH)

**Alexander Grosul**

Betreuer:
Dipl.-Inform. Max Völkel

verantwortlicher Betreuer:
Prof. Dr. Rudi Studer

Tag der Anmeldung: 03. Februar 2005
Tag der Abgabe: 31. August 2005

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Das World Wide Web (WWW) ist die Schnittstelle zwischen Mensch und Maschine, die in den letzten Jahren die größte Verbreitung erfuhr. In manchen Bereichen werden bereits auf dem lokalen Desktop Weboberflächen eingesetzt, wie zum Beispiel das Active Desktop von Microsoft und das Google Desktop. Die Webangebote stellen dabei nicht mehr reine Präsentationen von statischen Daten dar, sondern modellieren komplexe Anwendungsschnittstellen zur Nutzung im Browser.

Die Qualität der Web-Oberflächen wird meist nicht gebührend berücksichtigt. Einerseits besteht die Intention, die Erstellung schnell und kostengünstig zu gestalten. Häufig wird dabei die Qualität der Oberflächen vernachlässigt. Andererseits fehlt es, bei einer großer Vielfalt der Qualitätskriterien, die ihren Ursprung sowohl in der Gestaltpsychologie als auch im Design und in der Architektur haben, an allgemein anerkannten Standards für Web-Anwendungen. Diese wurden bislang nur ansatzweise formuliert und sollen nicht nur die zeitliche Effizienz sondern auch die subjektiv empfundene Arbeitsqualität berücksichtigen.

Die Entwicklung einer qualitativ hochwertigen Web-Anwendung ist derzeit sehr aufwendig und erfordert oft den Einsatz eines Entwicklungsteams, das in der Regel aus mindestens einem Web-Designer und einem Anwendungsentwickler besteht. Die innerhalb eines Teams entstehenden Missverständnisse führen oft zu hohen Entwicklungskosten. Wird an Entwicklungskosten gespart, führt dies unter Umständen zu einer fragwürdigen Benutzbarkeit der Anwendung.

Die Entwickler der Web-Anwendungen können bislang auf nur wenige Werkzeuge zurückgreifen, die das Erstellen einer Web-Anwendung im gewünschten Maße erleichtern würden. So müssen beispielsweise die zur Verfügung stehenden Frameworks den jeweiligen Anforderungen entsprechend kombiniert und angepasst wer-

den. Meistens müssen noch elementarere Bausteine (Links, Buttons, Formularfelder) zu komplexen Komponenten zusammengefügt und daraus Web-Oberflächen gestaltet werden. Dieser Prozess beinhaltet mehrere häufig wiederkehrende Aufgaben. Die Entwickler müssen diese jeweils neu lösen, was mit hohem Zeit- und Kostenaufwand verbunden ist. Zusätzlich entsteht durch unterschiedliche Lösungen der ähnlichen Aufgaben wie z.B. das Erstellen einer Suchmaske oder eines Anmeldungsformulars, eine Vielfalt der Web-Oberflächen. Für den Anwender ist dies häufig verwirrend und erfordert jeweils aufs neue Orientierungs- und Einarbeitungszeit. Ein Anwender, der die gewünschten Aufgaben nicht rasch und mühelos erledigen kann, wird durch die Komplexität der Web-Oberfläche abgeschreckt. Für den Anbieter birgt dies die Gefahr des Abwanderns der Anwender.

## 1.2 Idee

Die web-spezifische Darstellung und Interaktion mit einem Datenmodell erfordert typischerweise einen hohen Aufwand, der mit der Vielfältigkeit und Komplexität der Schnittstelle Internet und Browser zusammenhängt.

Die Idee dieser Arbeit besteht nun darin, ausgehend von einer Ontologie (hier im Sinne eines semantisch beschriebenen Datenmodells) automatisch eine Web-Anwendungsschnittstelle zu erzeugen. Dabei soll diese – anders als bei anderen Ansätzen – nicht nur “brauchbar“ sondern auch “gut benutzbar“ sein. Hierzu werden Qualitätskriterien zusammengestellt und angewandt.

Teilweise kann eine gute Qualität durch Wiederverwenden von spezifizierten Bausteinen (WebWidgets[1]) erreicht werden. Diese Bausteine werden anhand der Qualitätskriterien und der benutzerdefinierten Ontologie dynamisch ausgewählt und verwendet. Die Bibliothek von Komponenten ist dabei explizit durch die Entwickler erweiterbar.

Durch die Automatisierung der Entwicklung unter Berücksichtigung von Qualitätskriterien kann ein hochwertiges Ergebnis erzielt werden. Die zeitlichen und finanziellen Aufwände werden verringert, da der Entwickler lediglich mit einem

[1] “Das englische Wort *Widget* steht für ein unbestimmtes, namenloses technisches Produkt und wird in hypothetischen Beispielen als Platzhalter verwendet. Auf Deutsch bezeichnet man Widgets meist als *Steuerelemente*. Zu den am häufigsten genutzten Steuerelementen gehören die Schaltflächen und Bildlaufleisten.“ [wik05]

semantischen Datenmodell interagiert. Dieses wird vom System automatisch in Form einer interaktiven Anwendungsschnittstelle im Browser abgebildet.

Sollte sich dieser Ansatz durchsetzen, so würden Bedienung und Funktionalität von Web-Anwendungen einheitlicher und dadurch wesentlich benutzungsfreundlicher.

## 1.3 Aufgabenstellung

Das Ziel dieser Arbeit bestand darin, ein System zu entwickeln, das ausgehend von einer Ontologie automatisch eine qualitativ hochwertige Web-Anwendungsschnittstelle generiert. Das System nennt sich im Rahmen dieser Arbeit "WebWidgets-System". Innerhalb der Ontologie sollen sich Daten und deren zugewiesene semantische Beschreibung befinden. Der End-Entwickler soll sich lediglich mit einer einfachen Ontologie-Schnittstelle auseinandersetzen müssen.

Anwendungsbeispiel "Benutzer- und Gruppenverwaltung":

Einige Web-Anwendungen besitzen beispielsweise eine Benutzer- und Gruppenverwaltung. Diese verwaltet Benutzer und Gruppen sowie deren Relation. Benutzer und Gruppen werden jeweils durch eine Menge von Attributen beschrieben. Oft zeigen diese Anwendungen in Datenbanken enthaltene Daten an und bieten dem Anwender Möglichkeiten, diese zu modifizieren. Das verwendete Schema ist vergleichsweise einfach, die Umsetzung in eine HTTP/HTML Schnittstelle verursacht jedoch meist einen größeren Aufwand.

Für diese und ähnliche Fälle soll das WebWidgets-System kostengünstig eine hohe Benutzbarkeit gewährleisten. Folgende dazu notwendige Voraussetzungen sind zu untersuchen und zu erarbeiten:

- Welche Qualitätskriterien können für eine Web-Anwendung im Allgemeinen definiert werden?
- Wie können die definierten Qualitätskriterien unter dem Aspekt der automatischen Erstellung einer Web-Anwendungsschnittstelle technisch umgesetzt werden?
- Wie können die vorhandenen Daten visualisiert werden?

- Wie kann die Interaktion mit der Ontologie im Browser adäquat umgesetzt werden?
- Wie erfolgt die Synchronisation zwischen Dienstgeber, der die Daten verwaltet, und Ontologie?

Das System soll als Prototyp implementiert und anhand einer Web-Anwendung ("Benutzer- und Gruppenverwaltung") evaluiert werden.

## 1.4 Aufbau der Arbeit

In Kapitel 2 werden dem aktuellen Stand der Technik entsprechende Systeme vorgestellt, die eine gewisse Verwandtschaft zum WebWidgets-System ausweisen. Des Weiteren werden die konzeptionellen Grundlagen der JSF-Technologie erläutert. Diese werden im Prototyp des WebWidgets-Systems eingesetzt.

Kapitel 3 behandelt auf Web-Anwendungen bezogene Qualitätskriterien und stellt diese in einer Liste zusammen.

Kapitel 4 befaßt sich mit dem Entwurf und stellt ausführlich die Funktionsweise, die Systemarchitektur sowie konzeptionelle Details des WebWidgest-Systems dar.

In Kapitel 5 wird die Implementierung des Prototyps des WebWidgets-Systems erörtert.

In Kapitel 6 wird die Funktionstüchtigkeit des WebWidgets-Systems anhand einer Beispielanwendung bewertet. Zudem werden die Vorteile des Systems betrachtet.

Kapitel 7 fasst die wichtigsten Inhalte dieser Arbeit kurz zusammen.

# 2 Stand der Technik

Die in diesem Kapitel betrachteten Anwendungen beinhalten gewisse Verwandtschaften zur Idee des WebWidgets-Systems. Sie agieren jedoch nicht unter dem Aspekt von Qualitätskriterien und deren automatischer Umsetzung.

## 2.1 Vesuf

Das an der Universität Hamburg geführte Projekt *Vesuf* [PBB+02] beschäftigt sich mit der systematischen Erstellung von Anwendungsschnittstellen für Applikationen im Kontext des Ubiquitous Computing[1]. Das *Vesuf* System stellt damit eine universelle Plattform zur Verfügung, die die Ausführung beliebiger Anwendungen als Dienste ermöglicht.

Das im Rahmen dieses Projektes realisierte System erlaubt es, Applikationen auf einfache deklarative Weise mit verschiedenen Anwendungsschnittstellen-Modalitäten auszustatten und auf der Grundlage abstrakter Modelle für jede Modalität eine optimale Oberfläche zu konstruieren, ohne die Fachlogik modifizieren zu müssen. Die Autoren setzen sich dabei das Ziel, in erster Linie ablauffähige Oberflächen zu generieren. Qualitative Aspekte werden dabei erst durch schrittweise manuelle Ergänzungen und Verfeinerungen der Modellinformationen umgesetzt.

Das *Vesuf* System beruht auf UML-Semantik. Die Einsätze der Wiederverwendbarkeit, Austauschbarkeit, Komposition und Wartbarkeit der einzelnen Teilkomponenten des Gesamtsystems werden dabei auf einer hohen abstrakten Ebene spezifiziert. Die im *Vesuf* System realisierten Möglichkeiten zur Erstellung von Anwendungsschnittstellen decken ein breites Spektrum verschiedener Arten ab. So können nicht nur webbasierte (HTML), sondern auch graphisch interaktive (Java AWT), mobil zugreifbare (WML) und sprachbasierte (VoiceXML) Schnittstellen realisiert werden.

[1] Ubiquitous Computing kann mit der Allgegenwärtigkeit von miteinander vernetzten Computern übersetzt werden.

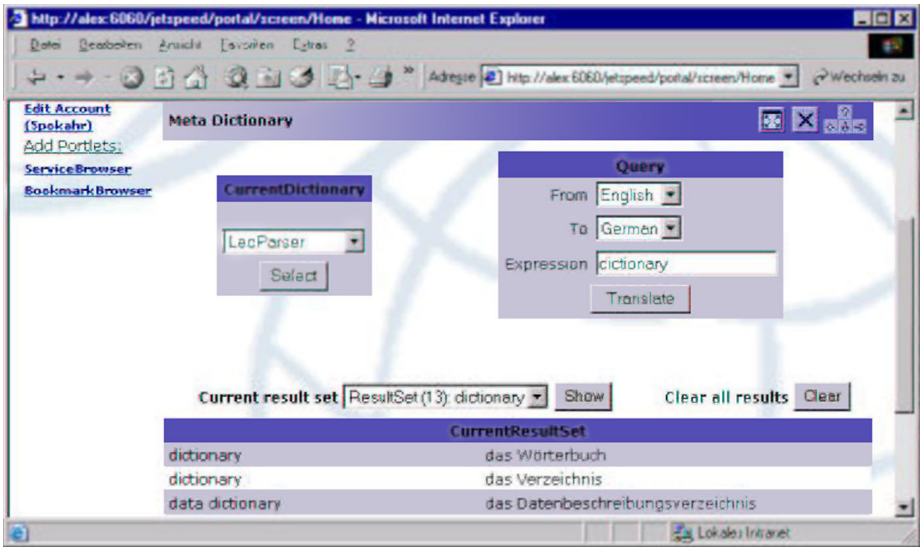


*Abbildung 2.1:* Vesuf

## 2.2 Web DynPro

*Web DynPro* [Dyn05] ist ein Programmiermodell von SAP[2] für die Entwicklung von Benutzeroberflächen mit Hilfe atomarer, wieder verwendbarer Bausteine. Die Architektur basiert auf dem Model-View-Controller-Programmiermodell und hat folgende Schwerpunkte:

- Klare Trennung von Business- und Anzeigelogik,
- Einheitliches Metamodell für alle Arten von Benutzeroberflächen,
- Ausführung in einer Vielzahl von Client-Plattformen,
- Weitestgehende Plattformunabhängigkeit der Schnittstellen.

Für die Erzeugung einer *Web DynPro* Anwendung werden zuerst mit Hilfe spezieller Entwicklungswerkzeuge Metadaten erzeugt, die die Funktionen und Eigenschaften der Anwendung beschreiben.

[2] SAP AG ist der Marktführer in Betriebswirtschaftlicher Standardsoftware mit Sitz in Walldorf, Baden-Württemberg


*Abbildung 2.2:* Web DynPro

Zur Laufzeit wird aus den Metadaten automatisch ein Quellcode generiert. Zusätzlich ist es möglich, die Elemente des Modells dynamisch zu ändern, um die Benutzeroberfläche beispielsweise plattformabhängig zu erweitern.

Als *Web DynPro* Client kann eine beliebige Software agieren, die über einen Framework die abstrakte Definition der Benutzeroberfläche und der Anwendungsdaten auswertet. Im Falle eines Internetbrowsers ist es eine JavaScript-basierte Bibliothek. Angeboten werden ebenfalls standalone oder mobile Clients sowie Flash-Filme[3].

## 2.3 Salvita Kompass

*Salvita Kompass* [Sal05] ist ein Software-Produkt, das in erster Linie die Visualisierung von beliebigen Prozessen ermöglicht. Die Visualisierung von *Salvita Kompass* ist über das gesamte Anwendungsspektrum hinweg konsistent.

*Salvita Kompass* beruht auf einer Eingabe-Schnittstelle, die unterschiedliche Aspekte, Planungen und Charakteristiken von Prozessen abbildet. Die Visualisierung aktualisiert sich permanent in Abhängigkeit durchgeführter Eingaben.

[3] Flash ist ein auf Vektorgrafiken basierendes Grafik- und Animationsformat der amerikanischen Firma Macromedia

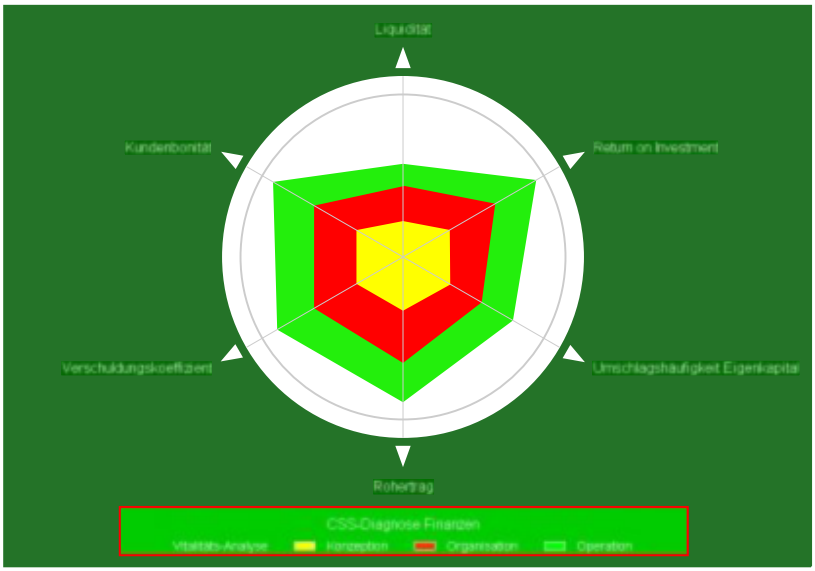


*Abbildung 2.3:* Salvita Kompass

Das Konzept des Produktes sieht die besondere Behandlung von Qualitätsaspekten in Bezug auf Ergonomie und Benutzbarkeit vor. Die Lösungsansätze des Systems werden durch die innovative Art der grafischen Darstellung realisiert. Dies weicht von üblichen Schemata deutlich ab und erhöht die subjektiv empfundene Arbeitsqualität.

## 2.4 JavaServer Faces

Aus den derzeit bekannten Web-Technologien wurde JavaServer Faces (JSF) [Jsf05] für die Implementierung des Prototyps von WebWidgets-System ausgewählt. Die Entscheidung wurde aufgrund der unten genannten Vorteile, die mit JavaServer Faces verbunden sind, getroffen. Sicherlich hätte auch eine andere Web-Technologie verwendet werden können.

Unter dem Aspekt der Erleichterungsmechanismen für den Web-Entwickler kann JSF ebenfalls als verwandte Arbeit angesehen werden. In diesem Abschnitt werden die konzeptionellen Grundlagen der JSF-Technologie vorgestellt. Dabei wurde der Schwerpunkt auf die für diese Arbeit relevanten Themen gelegt.

### 2.4.1 Einführung

JavaServer Faces ist der neue offizielle Standard von Sun, das die Entwicklung von Web-Anwendungen definiert und spezifiziert. JSF wurde vom Java Community Process[4] (JCP) als so genannter Java Specification Request (JSR) 127 entwickelt.

JavaServer Faces zielt darauf, die Entwicklung von Web-Anwendungen schneller und einfacher zu gestalten. JSF nimmt dem Entwickler viele komplexe Vorgänge ab, so dass er sich auf die Anwendungslogik konzentrieren kann. Dadurch soll die Fehleranfälligkeit der Anwendung gesenkt werden.

JSF ist keine komplett neuartige Technologie, sondern baut auf einer Reihe von existierenden Technologien auf, wie Servlets, JSP, JavaBeans oder Tag-Bibliotheken. JSF ist in erster Linie ein Framework zum Erstellen von graphischen Benutzeroberflächen für Web-Anwendungen. Es besitzt eine Komponenten-Architektur und hält große Anzahl von Standard-Komponenten, wie zum Beispiel Buttons, Hyperlinks, Checkboxen etc. bereit. Neben den Standard-Komponenten stellt JSF ein Modell zum Erstellen benutzerdefinierter Komponenten und ein weiteres Modell zum Bearbeiten der durch einen Anwender im Browser generierten Ereignisse, wie zum Beispiel das Ändern des Textes in einem Eingabefeld oder das Anklicken eines Buttons.

JSF bietet eine Möglichkeit zur klaren und strikten Trennung von Anwendungslogik und Darstellung. Dies stellt zwar keine Neuerung dar, ist jedoch in der JSF Technologie sehr konsequent und detailliert implementiert.

Die Trennung von Anwendungslogik und Darstellung erlaubt es, an einer Web-Anwendung mit verteilten Rollen zu arbeiten. Mittels JSF ist eine genaue Trennung nach Aufgabenbereich vorgesehen und mit vier Rollen beschrieben: Seitenautoren / Webdesigner, Applikationsentwickler, Komponentenentwickler, Tool-Hersteller.

JSF lässt sich auch mit anderen Frameworks wie zum Beispiel Struts kombinieren.

[4] http://www.jcp.org

### 2.4.2 Die Architektur von JSF

Die Architektur von JSF umfasst sechs verschiedene Ebenen:

Das *UI Component Model* beschreibt hauptsächlich die Funktionalität der einzelnen Komponenten, die verschiedene Möglichkeiten von Eingabe, Auswahl und Gruppierung repräsentieren. Das UI Component Model ist erweiterbar: mit wenig Aufwand können eigene Komponenten erstellt, die bereits vorhandenen Komponenten erweitert oder zusammengesetzt werden.

Das *Rendering Model* definiert die Art der Darstellung einer Komponente. Diese kann auf verschiedene Weise erfolgen, wenn verschiedene Renderer für eine Komponente zur Verfügung gestellt werden. Diese Entkopplung von Darstellung und Funktionalität der Komponente bietet einen weiteren Vorteil: dieselbe Komponente wird auf einem HTML Client (zum Beispiel Web Browser) oder auch einem WML Client (zum Beispiel Mobil-Telefon) darstellbar.

Das *Event Model* definiert Listener- und Event-Klassen, die von der Web-Anwendung gebraucht werden, um von Komponenten erzeugte Ereignisse zu verarbeiten.

Das *Validation Framework* stellt einen Mechanismus zur Verfügung, mit dem Anwendereingaben überprüft werden können. Im Falle eines verhafteten Verhaltens werden damit entsprechenden Fehlermeldungen erzeugt, die dem Anwender zur Anzeige gebracht werden.

Das *Page Navigation Framework* ermöglicht es, den Seitenfluss innerhalb einer Web-Anwendung deklarativ zu beschreiben.

Das *Internationalization Framework* stellt einen einfachen Mechanismus für die Internationalisierung von statischen und dynamischen Inhalten sowie Meldungen in einer Web-Anwendung zur Verfügung.

### 2.4.3 Konfigurationsdatei

Abgesehen von einem Deployment Deskriptor (*web.xml*) benötigt JavaServer Faces die JSF spezifische Anwendungskonfigurationsdatei *faces-config.xml*. In dieser Konfigurationsdatei erfolgt u. a. die Festlegung der Navigation der Anwendung, das zentrale Bean-Management (Siehe Abschnitt 2.4.6), sowie das Bekannt machen der benutzerspezifischen Komponenten und Renderer (Siehe Ab-

schnitt 2.4.7).

### 2.4.4 Lebenszyklus

Eine JSF-Seite ist zunächst einmal eine JSP-Seite, die in den Kontext einer JSF-Anwendung eingebunden ist. Sie durchläuft daher auch die gleichen sieben Phasen wie eine JSP-Seite:

1. *Seite übersetzten* – aus der JSP-Seite wird ein Servlet erzeugt
2. *Seite kompilieren* – das automatisch erzeugte Servlet wird kompiliert
3. *Seite laden* – die als Servlet kompilierte Seite wird durch den Classloader geladen
4. *Instanz erzeugen* – eine Instanz für die JSP-Seite wird angelegt
5. *Seite initialisieren* – die in der Servlet-Spezifikation hinterlegte *init*-Routine des Servlets/der JSP-Seite wird aufgerufen
6. *Service-Routine* – pro Aufruf der Seite wird ein Task gestartet, der die Service-Methode des Servlets ausführt
7. *Servlet entfernen* – wird das Servlet / die JSP-Seite nicht mehr benötigt bzw. wird zum Beispiel der Servletcontainer heruntergefahren, wird das Servlet korrekt entladen

Zusätzlich bietet JSF eine Vielzahl von Funktionen, die den Ablauf einer JSF-Seite beeinflussen und steuern. Der Request[5] einer JSF-Seite durchläuft bei einem Webserver einen umfangreichen Lebenszyklus. Dieser hängt davon ab, ob es sich um einen so genannten *Faces-Request* oder einen *Non-Faces-Request* handelt. Bei Faces Requests entstammt die Anfrage selbst aus einer JSF-Seite oder sie richtet sich an eine JSF-Seite. Ein Non-Faces-Request ist dagegen eine Anfrage an den Server nach einer anderen als einer JSF-Ressource zum Beispiel an eine HTML-Seite.

Ähnlich wie Requests unterscheiden sich auch Responses[6] in *Faces-Response* und

[5] Ein Request ist eine Anfrage eines Browsers an den Webserver

[6] Eine Response ist eine nach einem Request erfolgte Antwort eines Webservers an einen Browser

*Non-Faces-Response*. Bei einer Faces Response wird eine JSF-Seite, bei einer Non-Faces Response dagegen eine nicht JSF-Seite zurückgeliefert.

Basierend auf den unterschiedlichen Request- und Response-Arten sind verschiedene Szenarien und Kombinationen möglich. In der folgenden Erklärung wird davon ausgegangen, dass der Request von einer JSF-Anwendung erzeugt wurde und das Ergebnis wiederum eine JSF-Seite darstellt. Andere Szenarien sind Sonderfälle des so genannten *Standard Lebenszyklus*, der in der Abbildung 2.4 dargestellt ist.

Für das Verständnis der Verarbeitung innerhalb des Zyklus ist es wichtig zu wissen, dass innerhalb einer JSF-Anwendung jede Seite (View) als Baumstruktur dargestellt wird. Diese Baumstruktur wird als View-Objekt im so genannten *FacesContext* abgelegt. Der FacesContext repräsentiert Request-spezifische Informationen, die für die Verarbeitung eines eingehenden Requests und das Erstellen der entsprechenden Response benötigt werden.

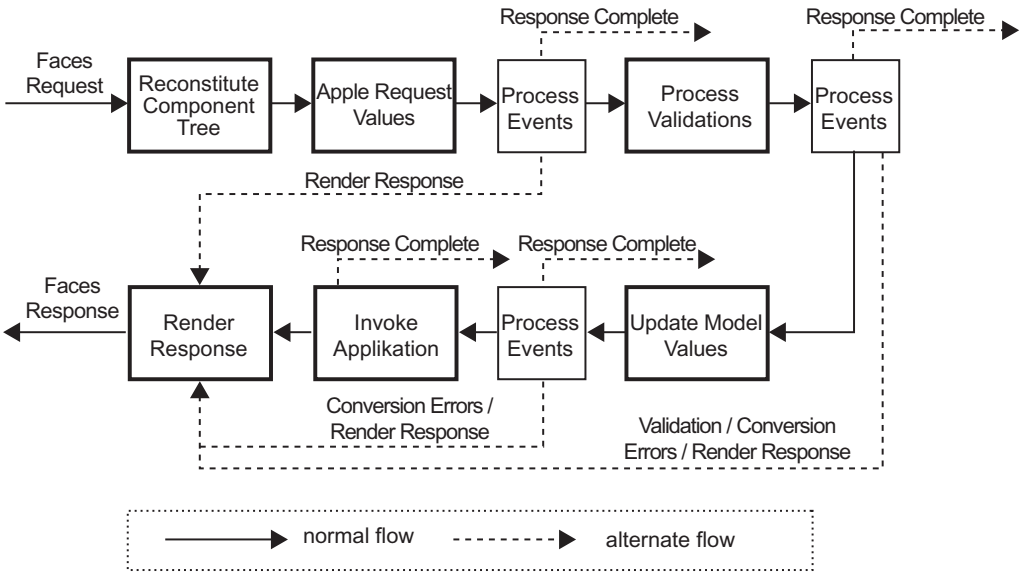


*Abbildung 2.4:* JavaServer Faces Standard Request Processing Lifecycle

Ein Faces Request durchläuft insgesamt neun Phasen. Von diesen neun können sechs klar abgegrenzt werden:

1. In der Phase *Reconstitute Request Tree* wird zuerst im Falle eines Non-Faces Requests ein leeres View-Objekt erzeugt und im FacesContext abgelegt. Im Falle eine Faces Requests wird ein bereits vorhandenes View-Objekt aus dem FacesContext geladen

2. In der Phase *Apply Request Values* holt sich jede einzelne Komponente des in der vorherigen Phase aufgebauten Komponentenbaumes den neuen Wert aus dem Request, konvertiert diesen in den entsprechenden Datentyp, so wie dieser im Modellobjekt hinterlegt ist, und speichert ihn bei sich lokal ab. Sollten in dieser Phase Fehler vorkommen, wird eine entsprechende Fehlermeldung generiert und im FacesContext gespeichert. Sind an einzelnen Komponenten entsprechende Eventlistener registriert, werden diese in der aktuellen Phase bereits benachrichtigt

3. In der Phase *Process Validations* werden sämtliche Überprüfungen durchgeführt, die in Form so genannter *Validatoren* an eine Komponente gekoppelt sind. Sollten in dieser Phase Fehler vorkommen, wird ebenfalls eine entsprechende Fehlermeldung generiert und im FacesContext gespeichert. Dann wird der Verarbeitungsablauf unterbrochen und auf die gleiche Seite zurückgewiesen

4. In der Phase *Update Model Values* werden die neuen Werte des Requests, die bereits in den Komponenten lokal gespeichert sind, in das Modellobjekt übertragen. Auch in dieser Phase können eventuell registrierte Eventlistener benachrichtigt werden

5. In der Phase *Invoke Application* werden sämtliche Events auf Anwendungsebene abgearbeitet. Dies ist meist die eigentliche Weiterleitung zu einer Folgeseite. Dabei wird wiederum der Komponentenbaum für die neue Seite aufgebaut, in FacesContext abgelegt und die Steuerung anschliessend an die nächste Phase übergeben

6. In der Phase *Render Response* wird im Normalfall der Komponentenbaum gerendert, der von der vorherigen Phase in FacesContext abgelegt wurde. Sollten jedoch in dieser Phase Fehler vorkommen, wird die ursprüngliche Seite nochmal dargestellt und zusätlich mit den entsprechenden Fehlermeldungen versehen

### 2.4.5 Standardkomponenten

JSF stellt einer Reihe der Standard-Komponenten zur Verfügung. Auf die einzelnen Komponenten wird hier nicht weiter eingegangen. Wichtig sind die fünf Interfaces, auf welche die Komponenten zurückgehen können:

Das Interface *ActionSource* wird von Komponenten implementiert, die Action Events auslösen, um diese durch den Default-ActionListener verarbeiten zu lassen.

Das Interface *NamingContainer* wird von Komponenten implementiert, die den Anspruch erheben, dass all ihre Kind-Komponenten eindeutig identifiziert werden können.

Das Interface *StateHolder* wird von Komponenten implementiert, deren Zustandsinformationen über den aktuellen Request hinaus erhalten bleiben sollen.

Das Interface *ValueHolder* wird von Komponenten implementiert, die Daten auf der Benutzeroberfläche darstellen oder von dort entgegennehmen und mit dem Modell über diese Daten kommunizieren.

Das Interface *EditableValueHolder* ist eine Erweiterung vom Interface ValueHolder. Sie stellt zusätzliche Methoden für editierbare Komponenten zur Verfügung.

### 2.4.6 Managed Bean

Die Daten aus dem Modell werden über so genannte *Managed Beans* den Views zur Verfügung gestellt. Managed Beans sind JavaBeans, die gemäß ihrer Konfiguration in der Konfigurations-Datei *faces-config.xml* von JSF selbständig verwaltet werden. Sie besitzen einen eindeutigen Namen. Sobald eine Bean benötigt wird, kann sie über diesen Namen angesprochen werden. In diesem Fall wird ein Objekt dieser Bean automatisch erzeugt. Die Managed Beans besitzen auch so genannte *Managed Properties*, welche in einer automatisch durchgeführten Initialisierung mit Werten gefüllt werden. Dazu müssen die Beans gemäß den JavaBeans-Konventionen die entsprechenden *getter*- und *setter*-Methoden zur Verfügung stellen.

### 2.4.7 Erstellen eigener Komponenten

Die Architektur von JSF ist so ausgelegt, dass eine Erweiterung von Standard-Komponenten jederzeit möglich ist. Die Entwickler können eigene benutzerspezifische Komponenten oder neue Renderer für eine bereits bestehende Komponente erstellen. Es soll dabei aufgabenbezogen vorgegangen werden:

- Besteht das Anliegen darin, ein neues Erscheinungsbild zu erzeugen, so ist die komplette Neuentwicklung einer Komponente unnötig und es genügt, einen speziellen Renderer dafür zu entwickeln
- Geht es darum, Werte in einem bestimmten Format aus der Komponente zurückgeliefert zu bekommen, ist dies eine Aufgabe für benutzerspezifische Konverter
- Wird jedoch ein komplett neuartiges Verhalten oder eine Funktionsweise einer Komponente benötigt, ist die Entscheidung zur Entwicklung einer benutzerspezifischen Komponente genau richtig

Die Entwicklung einer benutzerspezifischen UI-Komponente vollzieht sich in mehreren Arbeitsschritten [Bos04]:

1. Schreiben einer Taghandler-Klasse, die dafür verantwortlich ist, den dazugehörigen Renderer zurückzuliefern, sowie den Typ der Komponente zu bestimmen. Übernimmt die Komponente selbst deren Darstellung, wird kein Renderer zurückgeliefert
2. Erzeugen eines Tag Library Deskriptors (TLD), der die Beschreibung des verwendeten Tags enthält
3. Entwicklung der Komponentenklasse, die keinen speziellen Renderer zurückliefert. Im Einzelnen ist eine Komponentenklasse für die Codierung, Decodierung, Zustandsspeicherung, Validierung, Aktualisierung des Modells sowie Verarbeitung der Ereignisse zuständig
4. Bekannt machen der Komponente in der Konfigurations-Datei *faces-config.xml*
5. Einbinden der neuen Taglib in eine JSF-Seite sowie Verwendung und Test der Komponente

Wenn man die Darstellung an einen neuen Renderer übergeben möchte, so erweitern sich die oben genannten Schritte durch die folgenden:

1. Erweiterung der Taghandler-Klasse – jetzt muss ein spezieller Renderer zurückgeliefert werden
2. Entwickeln des neuen Renderers
3. Eintragen des Renderers in der Konfigurations-Datei *faces-config.xml*
4. Anpassungen in der Komponentenklasse vornehmen

### 2.4.8 JSF-Vorteile

Die einzelnen Vorteile, die JSF bietet, sind hier zusammengefasst:

- Mit JSF gibt es einen ersten offiziellen Standard von Sun zur Erstellung von Web-Anwendungen
- JSF ist keine komplett neuartige Technologie, sondern eher eine konsequente Weiterentwicklung bestehender Technologien, die sich bereits etabliert haben wie Servlets, JSP, JavaBeans oder Tag-Bibliotheken
- Klare und strikte Trennung von Anwendungslogik und Darstellung – an sich nichts neues, jedoch ist die Technologie in JSF sehr konsequent und detailliert implementiert
- Rollenkonzept – die Trennung von Anwendungslogik und Darstellung erlaub es, an einer Web-Anwendung mit verteilten Rollen zu arbeiten. Mit dem JSF-Rollenkonzept ist eine genaue Trennung nach Aufgabenbereich in vier Rollen vorgesehen – Seitenautoren / Webdesigner, Applikationsentwickler, Komponentenentwickler und Tool-Hersteller
- JSF stellt eine Tag-Bibliothek für die Darstellung von graphischen Komponenten, deren Verwendung vereinheitlicht und vereinfacht ist und die beim Bedarf auch angepasst und erweitert werden können
- JSF liefert eine Vielzahl an vordefinierten Routinen und Funktionen, die eine Anwendungsentwicklung erheblich vereinfachen und beschleunigen – zum Beispiel Datenvalidierung, Event Handling oder Internationalisierung.

Dazu gehören auch serverseitige Helper-Klassen, die zum Beispiel für eine Kommunikation via RMI oder Corba oder für Datenbankzugriffe eingesetzt werden können

- Modulare Aufbau von JSF ermöglicht komfortables Entwickeln eigener benutzerdefinierten Komponenten

# 3 Qualität von Anwendungsschnittstellen im Internet

## 3.1 Warum Qualität für Web-Anwendungen?

Die Häufigkeit und die Frequentierung einer Web-Anwendung durch Anwender sind abhängig von ihrer Attraktivität für den Anwender. In erster Linie sind Web-Anwendungen als Informationsquellen bzw. -schnittstellen zu sehen. Allerdings entscheidet der Anwender eben nicht nur nach reinen Inhaltskriterien, wie oft und ob überhaupt eine Webseite von ihm besucht wird. Um die Inhalte nutzen zu können, müssen sie zum einen verständlich sein, zum anderen in einer Form dargeboten werden, die das Aufnehmen angenehm und komfortabel macht. Die Erfüllung dieser Anforderungen setzt eine gewisse Qualität der Gesamtheit voraus. Viele Anwender beurteilen komplexe Gebilde wie eine Webseite intuitiv. Sie setzen sich nicht ausführlich mit der Frage auseinander, warum eine Anwendung gut oder schlecht ist, sondern ob sie den eigenen Ansprüchen genügt oder nicht. Unter diesem Aspekt soll im Folgenden auf Webspezifische Qualitätsaspekte eingegangen werden. Dabei wird eine Auflistung von Qualitätskriterien angestrebt, die im Folgenden eine automatische Qualitätssicherung mit Daten versorgt. Ist es möglich, den Entwicklern ein derartiges automatisches Qualitätsmanagement innerhalb der Ontologie anzubieten, wird dies Entwicklungs- und Anwendungsprozesse von Web-Anwendungen konstruktiv unterstützen: Der Entwickler profitiert durch verkürzte Entwicklungszeiten und ein garantiertes Qualitätsniveau seiner Entwicklung. Der Anwender profitiert durch ergonomisch optimierte Webseiten und infolgedessen kürzerer Einarbeitungsphasen.

Die folgenden Ausführungen stützen sich auf Publikationen folgender Autoren: [Kru02], [Thi01], [Shn98] und [Nie00]. Die Inhalte dieser verschiedenen Quellen entsprechen sich größtenteils, sind jedoch quantitativ sehr verschieden. Einige Kriterien gründen sich zudem auf eigene Beobachtungen und Analysen.

## 3.2 Aufbau einer Webseite

Laut [Thi01] kann eine Webseite in folgende Gruppen von Elementen aufgeteilt werden:

- *Orientierungs-Elemente* – diese Elemente dienen der Orientierung des Anwenders innerhalb des Systems
- *Navigations-Elemente* – diese Elemente ermöglichen dem Anwender eine aktive Beweglichkeit im multimedialen Raum. Durch sie besteht die Möglichkeit, gezielt in bestimmte Bereiche zu gehen bzw. diese auszuwählen
- *Inhalts-Elemente* – sie sind in Form von aufbereiteten Daten vorhanden. Inhalts-Elemente stellen in Form von Texten, Bildern, Tönen, Videos oder Animationen alle möglichen Formen von Informationen dar
- *Screenlayout-Elemente* – die Organisation des Aufbaus einer Bildschirmseite wird durch diese Elemente ermöglicht. Sie bringen Inhalte in Beziehung zueinander und sorgen für ein stimmiges Gesamtbild
- *Interaktions-Elemente* – diese Elemente geben dem Anwender Rückmeldungen auf seine Eingaben: Sie reagieren auf verschiedene Aktivitäten des Anwenders
- *Motivations-Elemente* – sie machen das System für den Anwender attraktiv und erhöhen die Chance der wiederholten Benutzung

Diese Elemente sind für diesen Abschnitt ausschlaggebend. Im Folgenden wird die Webseite an sich Betracht genommen.

## 3.3 Anforderungen an eine Webseite

Webseiten werden gelegentlich besucht. Ihre Nutzung sollte für den Anwender daher möglichst unkompliziert sein. Eine Einarbeitung oder langes Nachdenken werden in diesem Zusammenhang negativ bewertet. Eine gute Webseite sollte folgende Fragen unmittelbar beantworten:

- Wer ist der Betreiber der Seite?
- Wer soll durch die Seite angesprochen werden?

- Welche Inhalte werden durch die Seite beschrieben?
- Welches Ziel wird durch die Seite verfolgt?
- Auf welchem aktuellen Stand sind die Inhalte?

Oft werden Webseiten im Browser durch einen Quereinstieg in Form von Verlinkungen von anderen Seiten geöffnet. Daher sollten alle oben stehenden Fragen auf jeder Inhaltsseite der Web-Anwendung zu finden sein. Eine Beschränkung dieser Informationen allein auf die Startseite reicht also in der Regel nicht aus. Ist es nicht möglich, alle diese Fragen in den Inhaltsseiten zu beantworten, sollte zumindest ein klar aufgezeigter Weg zum Erreichen der Startseite vorhanden sein. In jedem Falle sollte sich der Anwender innerhalb einer Web-Anwendung stets im Klaren darüber sein, von wem sie kommt. Damit wird die Frage vermieden, ob er sich noch bei der gleichen Web-Anwendung befindet, die er ursprünglich angewählt hat.

## 3.4 Orientierungs- und Navigations-Elemente

Die Webseite soll dem Anwender ein einfaches Zurechtfinden ermöglichen. Er soll leicht nachvollziehen können, wie die inhaltliche Struktur der Seite aufgebaut ist. Folgende Punkte sollte der Anwender jederzeit beantworten können:

- Wo befindet er sich gerade?
- Wie kehrt er auf die Startseite zurück?
- Welche Bereiche hat er schon aufgesucht?
- Wo kann er noch hingehen?

Unterm Strich kann man also sagen, dass eine klare Orientierung und ein einfaches Navigieren innerhalb einer Web-Anwendung elementare Kriterien für deren praxisgerechte Benutzung darstellen. Um eine Navigation mit hohem Bedienkomfort zu erstellen, ist ihre Optimierung für den Anwender notwendig. Hierbei kann schon die Analyse der angestrebten Zielgruppe einer Web-Anwendung Hilfestellung zur Erfüllung des Lösungsweges geben: Hat man Vorlieben bestimmter Gruppen in Bezug auf die Bedienung von Web-Anwendungen ermittelt, kann

die Navigation auf die ermittelten Kriterien hin abgestimmt werden. Bei Web-Anwendungen mit breit gefächerten Zielgruppen bietet es sich an, die Funktionen durch die Ausführung der Navigation mehrfach zu belegen, um viele verschiedene Gewohnheiten zu bedienen. So kann beispielsweise eine aus Links bestehende Menüleiste in einer Seite zweimal auftauchen, als horizontale und noch als vertikale Einheit. Einzelne Links können zugleich in Form von Begriffen, Symbolen oder Bildern vorhanden sein. Zudem sollte ein Navigationselement auf seine Funktion abgestimmt sein: Durch eine entsprechende Semantik wird sofort klar, was ein solches Element tut. Dabei sollten die Navigationselemente die eigentlichen Inhalte nicht dominieren. Sind all diese Punkte berücksichtigt, kann sich der Anwender im Idealfall intuitiv innerhalb der Web-Anwendung bewegen.

Um Möglichkeiten einer leicht verständlichen Orientierung und Navigation aufzuzeigen, wurde im Folgenden eine Liste zusammengestellt, die praktische Hilfen aufzeigt:

- Assistenten und geführte Touren können besonders dann hilfreich sein, wenn es sich um größere Web-Anwendungen handelt, die ein breites Inhaltsspektrum bieten. Diese Hilfen müssen jedoch gut gemacht sein: Ein Assistent, der beispielsweise Benutzerfragen nicht versteht, ist genauso lästig wie einer, der zu dominant auftritt oder sich nicht deaktivieren lässt

- Mittels Metaphern ist es möglich, dem Anwender etwas Bekanntes oder Vertrautes aus dem Alltag zu bieten. Werden solche Eigenschaften effektiv eingesetzt, wird der intuitive Umgang des Anwenders unterstützt: Ein Button mit dem Bild “Warenkorb“, wie er oft in Internet-Shops verwendet wird, braucht beispielsweise keine Beschriftung, da die Anwender daran gewöhnt sind und das Element sofort erkennen

- Orientierungselemente können durch verschiedene Aspekte in Erscheinung treten. So können Farben und Strukturen sowie Piktogramme und Kürzel Merkmale bilden, die intuitiv und schnell wieder erkannt werden

- Die Zusammenfassung von Navigation und Orientierungshilfe in einem Element kann den Vorteil haben, dass deren Zusammenhang mit den Inhalten deutlicher wird. Zudem kann die ästhetische Ausgestaltung der Anwendung deutlich gewinnen. Unter Umständen ist aber ein Navigationselement in diesem Fall schlecht als solches zu erkennen. Wird beispielsweise auf einen

"home"-Button verzichtet, so kann der Anwender vermuten, dass das Logo des Anbieters zu Startseite führt – Gewissheit erhält er jedoch erst beim Versuch, es anzuklicken

- Bei großen Web-Anwendungen mit umfangreichem Inhalt ist es sinnvoll, Seitenübersichten, Suchfunktionen oder Indexverzeichnisse anzubieten, um dem Anwender Zeit zu sparen. Bei kleinerem Umfang ist eine solche Einrichtung überflüssig oder sogar störend

## 3.5 Inhalts-Elemente

Informationen sind besonders schwer zu verarbeiten, wenn sie auf einem Bildschirm dargestellt werden. Es ist daher wichtig, die Inhalte möglichst in einer Form zu präsentieren, die diesen Umstand teilweise kompensiert. Schlecht aufbereitete Inhalte stoßen den Anwender ab und werden trotz möglicherweise einfachem Inhalt als schwer empfunden.

Inhalte sollten prinzipiell informativ und für den Anwender relevant sein und sollten keine Unwahrheiten aufweisen. Das wichtigste Qualitätskriterium ist jedoch die Einfachheit.

### 3.5.1 Text

Für ein gutes Verständnis sollten Texte kurz, klar und deutlich verfasst sein. Lange und verschachtelte Sätze sind ebenso zu vermeiden wie unverständliche Wörter. Fachbegriffe sollten erklärt und Mehrdeutigkeiten vermieden werden.

Da Web-Anwendungen oft zum schnellen Aufnehmen von Informationen aufgesucht werden und ein Großteil der Inhalte in Form von Bildern dargestellt wird, werden Texte oft nur sekundär betrachtet. Viele Anwender neigen dazu, Texte flüchtig oder gar nicht zu lesen. Es ist sinnvoll, Texte inhaltlich, vor allem aber auch optisch zu strukturieren. Überschriften und maßgebliche Stichwörter sollten hervorgehoben werden, Wesentliches von Unwesentlichem unterschieden werden. Bei einer großen Auswahl an verschiedenen Textpassagen ist es vorteilhaft, die Texte mit einer kurzen Zusammenfassung beginnen zu lassen, ähnlich Suchmaschinen. Lange Texte sollten nicht zusammen an selber Stelle mit kurzen Texten stehen.

Ein sehr wichtiger Punkt ist die Wahl der Einstellungen in Bezug auf Typografie und Schriftgröße. In jedem Fall sollte der Text gut lesbar sein. Durch eine Schriftart kann der Text eine eigene Charakteristik erhalten, die den Inhalt des Textes unterstreicht oder abschwächt. Zu beachten ist, dass die gewählte Typografie möglicherweise nicht auf allen Rechnern verfügbar ist. Für diesen Fall sollte eine Alternative angeboten werden, die dem Original-Text am nächsten kommt. Zudem müssen bei Textblöcken, Worttrennungen und Zeilenumbrüchen entsprechende Umstände berücksichtigt werden. Worttrennungen sind hier beispielsweise problematisch: Liegt ein Wort, das mit Bindestrich getrennt ist, in der einen Schriftart direkt am Zeilenumbruch, taucht es in einer anderen Typografie in der Mitte einer Zeile auf. Das verwirrt den Anwender dann oder lässt den Lesefluss stocken. Kursive, vor allem aber fette Texte sind in Web-Anwendungen schlecht lesbar. Hier kann beispielsweise auf monochrome Farbabstufungen zurückgegriffen werden, um Wörter oder Textpassagen hervorzuheben. Auch der ausschließliche Einsatz von Großbuchstaben sollte nur sparsam verwendet werden. Ein Mischen von Schriftarten in einem Fließtext ist ebenso störend wie optische Effekte (Blinken, ständige Farbwechsel o. ä.). Da in Web-Anwendungen unterstrichene Wörter allgemein als Links zu anderen Seiten anerkannt sind, sollte man dieses Mittel zur alleinigen Hervorhebung ebenfalls vermeiden. Zum Schluss sollte mach noch berücksichtigen, dass verschiedene Anwender auch verschiedene Auflösungen auf ihren Bildschirmen eingestellt haben. Schriftgrößen sollten daher weder zu groß, noch zu klein sein.

Sehr lange oder kurze Zeilen erschweren das Lesen. Als Richtwert kann man grob sagen, dass 40 bis 60 Zeichen pro Zeile einen angenehmen Wert darstellen, je in Abhängigkeit zur Schriftart und -größe. Nicht nur Schriftgrößen, sondern auch die Zeilenabstände sollten überprüft und entsprechend angepasst werden, um einen guten Lesefluss zu erzielen. Im Zuge der Strukturierung des Textes sollte dieser Leerräume enthalten, die als Ruhepunkte dienen. Linksbündige Texte sind leichter zu lesen als rechtsbündige. Bei längeren Passagen können Blocktexte das grafische Layout einer Seite positiv unterstreichen und diese optisch ruhiger machen.

Passt eine Seite nicht ganz auf den Bildschirm, aktivieren sich automatisch die Scrollerbalken rechts und unten am Bildschirm. Dieses Phänomen ist bei den Anwendern nicht sehr beliebt. Vor allem der horizontale Scroller wird nicht gern gesehen, da er im Gegensatz zum Vertikalbalken nicht mit dem Scrollerrad bedient

werden kann, sofern dies vorhanden ist. Besonders kritisch sind diese Balken, wenn die Seite nur minimal zu groß ist. Bei langen Textpassagen sollten, wenn diese unumgänglich sind, für diese eigene Frames erstellt werden, innerhalb derer lediglich der Text gescrollt wird. Zudem sollte dem Anwender die Möglichkeit zum Ausdrucken angeboten werden, da Texte ohnehin meist in ausgedrucktem Zustand gelesen werden.

Um den Text gut in eine Webanwendung einzubinden, können folgende Punkte als Richtlinien genannt werden:

- Der Text sollte nicht zu viel Raum innerhalb des gesamten Layouts einnehmen
- Der Hintergrund darf nicht zu dominant sein – andernfalls wird unter Umständen das Lesen erschwert oder der Anwender ist einer permanenten Ablenkung ausgesetzt
- Hohe Helligkeitskontraste zwischen Text und Hintergrund erleichtern das Lesen; starke Farbkontraste hingegen können einen Text schlecht lesbar machen
- Ein weißer Hintergrund kann schnell zu Ermüdungen der Augen führen

### 3.5.2 Bild

Der Umfang an zu beachtenden Regeln ist im Hinblick auf einen qualitativ hochwertigen Einsatz von Texten recht umfangreich, wie der vorhergehende Abschnitt verdeutlicht. Ebenso gelten aber auch bestimmte Regeln für den Einsatz von Bildmaterial innerhalb einer Web-Anwendung. “Ein Bild sagt mehr als tausend Worte“ – dieses Sprichwort lässt sich auch im Internet anwenden. Da jedoch, wie eingangs erwähnt, viele Anwender Web-Anwendungen aufsuchen, um schnelle Informationen zu erhalten, können Bilder unter Umständen zu einer oberflächlichen Betrachtung oder Wahrnehmung der Inhalte einer Seite führen. Dies kann bei ungeschickt ausgeführten oder platzierten Bildern vom eigentlichen Inhalt einer Seite ablenken.

Bilder können innerhalb einer Web-Anwendung mehrere Funktionen haben:

- Bilder können bestimmte Themen veranschaulichen; sie können beispielsweise einem Text als Ergänzung zur Seite gestellt werden. In diesem Fall

ist zu beachten, dass die Bilder im richtigen Kontext angeordnet werden und nicht willkürlich auf der Oberfläche platziert werden. Bilder mit hoher Farbsättigung können eventuell von einem textlichen Inhalt ablenken. Die Alternative von monochromen oder schwarzweißen Bildern bietet sich an, wenn die Farbe keinen Einfluss auf die Aussage des Bildes hat. Das Bild fügt sich dann besser in den Kontext ein

- Der Einsatz von Bildern ermöglicht die Unterstützung von Strukturierungen bestimmter Themen. Somit können Bilder die Aufgabe von Orientierung und Navigation erfüllen

- Bilder können für dekorative Zwecke eingesetzt werden. In diesem Fall spielt das Thema ästhetischer Kontext eine sehr große Rolle: Siehe Punkt 1

### 3.5.3 Icons

Zur Unterstützung von Navigation und Orientierung bietet sich der Einsatz von Icons an. Icons sind Symbole, die Objekte, Tätigkeiten, Funktionen oder Zusammenhänge symbolisieren können. Sie sind jedoch nur dann sinnvoll, wenn eindeutig aus ihnen hervorgeht, was sie bedeuten. Icons sollten also selbsterklärend sein. Auch hier ist wieder die Abstimmung auf Zielgruppen empfehlenswert: Dadurch erhöht sich die Wahrscheinlichkeit, dass die Anwender schnelle Assoziationen zum Informationsgehalt des Icons herstellen können und sich somit die Bedienung der Anwendung vereinfacht bzw. beschleunigt. Im Zweifelsfall der Verständlichkeit ist einem Icon der Begriff zu Seite zu stellen. Zudem sind Icons in erster Linie nach funktionalen Gesichtspunkten zu gestalten. Im Idealfall sollten Icons innerhalb einer Anwendung einen durchgängigen ästhetischen Ausdruck haben, der auf das gesamte Layout abgestimmt ist.

Die Wirkungsweise von Icons ist stark Abhängig von der Umgebung, in der sie eingebunden sind:

- Icons sollten sparsam und gezielt zum Einsatz kommen. Zu viele Icons können verwirren und die Nutzung erschweren, da die Informationsfülle zu groß und zu unübersichtlich ist

- Die räumliche Anordnung von Icons sollte einem logischen Prinzip folgen

- Icons können sich gegenseitig beeinflussen. Es ist daher stets im Kontext zu prüfen, ob ein Icon richtig interpretiert werden kann

- Bei der Anwendung von Icons ist die Interpretation durch den Anwender zu berücksichtigen. Verschiedene Bildungsmilieus und Altersgruppen nehmen Symbole auf unterschiedliche Art und Weise war. Zudem ist das kulturelle Verständnis verschiedener Gruppen und Nationen zu berücksichtigen

### 3.5.4 Video, Animation und Ton

Die Aspekte Video, Animation und Ton sollen hier nur am Rande erwähnt werden, da sie nicht direkter Bestandteil der Arbeit sind. Allerdings finden sie in zahlreichen Anwendungen einen unterstützenden oder ergänzenden Einsatz.

Beim Einsatz von animierten Einleitungen ist darauf zu achten, dass die Einleitung mit allgemein verfügbaren oder vorhandenen Playern abgespielt werden soll. Steht dem Anwender kein entsprechender Player zu Verfügung, sollte ein Link zum Download der entsprechenden Komponente angeboten werden. Die Intro sollte nicht zu lang sein. Lange Ladezeiten werden negativ aufgenommen. Der Anwender sollte die Möglichkeit zum Überspringen der Intro haben. Diese Überspringungsfunktion ist deutlich zu kennzeichnen. Animationen sollten innerhalb der Anwendung sparsam verwendet werden. Sich ständig wiederholende Animationen lenken ab und lassen den Anwender schneller ermüden. Animationen, die durch den Cursor aktiviert werden, sollten keine zu extremen Reaktionen erzeugen. Die Anwendung sollte auch ohne die Animation funktionieren können.

Ton ist stets als untermahlendes Element zu betrachten: akustische Signale oder musikalische Untermahlung sollten eine leicht erkennbare Stummschaltungsfunktion aufweisen. Nicht jeder Anwender verfügt über die Möglichkeit, Sounds abzuspielen. Bei der Verwendung von Videos ist ein guter Kompromiss zwischen Dauer der Ladezeit und Qualität des Film umzusetzen: Nichts enttäuscht mehr, als ein Video in mangelhafter Qualität, das zuvor mühsam heruntergeladen werden musste.

In jedem Fall ist für Video, Animation und Ton das Erstellen verschiedener Qualitätslevels empfehlenswert: Der Anwender kann dann entsprechend der Ausstattung seines Rechners bzw. seiner Internetverbindung die geeignete Variante auswählen.

## 3.6 Screen-Layout-Elemente

Um die Inhalte einer Web-Anwendung zu einer funktionalen und ansprechenden Gesamtheit werden zu lassen, müssen die einzelnen Elemente Bezug aufeinander nehmen und sich gegenseitig ergänzen. Bei der Gestaltung der Bildschirmseite gilt das Prinzip der Einfachheit: Ist die Gestaltung der Seite klar und übersichtlich gestaltet, kann sie schneller erfasst und verstanden werden. Der Zugriff auf einzelne Elemente wird erleichtert. Das ausgewählte Layout sollte innerhalb einer Web-Anwendung konsistent bleiben. Weder sollte der Anwender durch die Gestaltung überfordert, noch darf er gelangweilt sein. Die für Text und Bild geltenden Kriterien sind auch auf das gesamte Grundlayout der Web-Anwendung anwendbar.

Für eine wahrnehmungsfreundliche Gestaltung einer Web-Anwendung sind die Gesetze aus der Gestaltungspsychologie von Bedeutung. Im Folgenden werden die wichtigsten Kriterien aufgelistet:

- *Das Gesetz der Nähe*: Nah zusammenliegende Elemente werden als einander zugehörig angesehen
- *Das Gesetz der Ähnlichkeit*: Ähnlich aussehende Elemente werden kognitiv einander zugeordnet und als Gruppe empfunden
- *Das Gesetz der Geschlossenheit*: Um eine geschlossene Form wahrzunehmen, reicht eine Bezugnahme von offenen Elementen zueinander aus
- *Das Gesetz der Symmetrie*: Symmetrisch angeordnete Elemente werden als Einheit im Vordergrund aufgefasst, Asymmetrien auf den Hintergrund bezogen
- *Das Gesetz der Prägnanz*: Einfach und konsistent angeordnete Elemente bilden klare Strukturen und heben sich von Hintergrund ab
- *Das Gesetz der guten Fortsetzung*: Kontinuierlich angeordnete Elemente werden als einander zugehörig gelesen
- *Das Gesetz der Erfahrung*: Die visuelle Wahrnehmung kann durch das Zurückgreifen auf Erfahrungen unvollständige Abbildungen komplettieren

Farben können die emotionale Wahrnehmung der Menschen stark beeinflussen. Die Wirkung von Farben ist jedoch ein eigenes Kapitel, dessen Umfang im Rahmen dieser Arbeit nicht ausgeführt werden kann. Die Kombination verwandter Farben sowie der Einsatz monochromer Farbabstufungen wirken ruhig und ausgeglichen. Komplementäre Farben können Spannungen erzeugen, Farben mit hohem Sättigungsgrad haben Signalcharakter. Großflächig eingesetzte Farben mit hoher Intensität können ermüdend wirken. Dunkle Farben werden meist als angenehm empfunden. Kalte Farben werden räumlich weiter wahrgenommen als warme Farben.

## 3.7 Interaktion-Elemente

Dem Anwender sollte stets ein sicheres Gefühl in der Benutzung seines Computers vermittelt werden. Dies gilt besonders für Web-Anwendungen, da sie unter Umständen von gewohnten Handlungen des Anwenders abweichen können. Um trotzdem eine gewisse Sicherheit zu gewährleisten, können nachfolgende Punkte berücksichtig werden:

- Der Anwender möchte wissen, was der Computer für ihn tun kann
- Was tut der Computer zurzeit? Der Anwender sollte darüber auf dem Laufenden gehalten werden, damit u. a. unnötige Eingaben vermieden werden
- Der Computer sollte alle Eingaben quittieren: Hat er die Eingaben verstanden?
- Es sollte sofort erkennbar sein, ob der Computer noch ordnungsgemäß funktioniert

Die Rückmeldungen als Reaktion auf diese Fragen erhält der Anwender in Form von visuellen und akustischen Signalen. Damit diese richtig interpretiert werden können, ist folgendes zu berücksichtigen:

- Sämtliche anklickbare Elemente einer Art sollten stets auf dieselbe Art und Weise reagieren. Dies gilt sowohl für visuelle als auch für akustische Signale. Die Konsistenz dieser Elemente muss durchgängig sein
- Auf jede Aktion des Anwenders sollte eine Gegenreaktion des Computers erfolgen

- Interaktive Elemente sollten in ihrer Darstellung nicht mit statischen Elementen gleichgestellt oder vermischt werden
- Durch die Anwendung geweckte Erwartungen sollten möglichst erfüllt werden

Die Kriterien für das Design ergonomischer Benutzerschnittstellen werden auch in der europäischen Norm EN ISO 9241 definiert. Auf das Webdesign können die in Teil 10 definierten "Grundsätze der Dialoggestaltung" übertragen werden:

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Steuerbarkeit
- Erwartungskonformität
- Fehlertoleranz
- Individualisierbarkeit
- Lernförderlichkeit

## 3.8 Motivations-Elemente

Eine Web-Anwendung sollte den Anwender motivieren, sie gern zu benutzen und sich mit ihr zu beschäftigen. Der Erste Eindruck einer Web-Anwendung ist maßgeblich für eine solche Motivation. Der Anwender sollte den Nutzwert der Seite für seine eigenen Interessen schnell erkennen und ein mögliches Potential der Inhalte sehen können. Eine gute Web-Anwendung sollte an die Neugier und die Aufmerksamkeit des Anwenders appellieren.

Motivationselemente können sich in verschiedenen Bereichen ausdrücken. In den vorhergehenden Abschnitten wurde auf alle Kriterien eingegangen, die zu einer Förderung der Motivation beitragen. Im abstrakten Bereich können als Beispiele folgende inhaltliche Kriterien genannt werden: Vergnügen, Informationsgehalt, Seriosität, Insiderinformationen, Hochwertigkeit, Spannung, Humor, Verständnis etc. Idealerweise werden diese Kriterien durch die praktische Gestaltung der Anwendung ausgedrückt. Diese sollen nicht nur die zeitliche Effizienz sondern auch

die subjektiv empfundene Arbeitsqualität berücksichtigen. Da sie auch durch Animation, Video und Ton dargestellt werden können, sollen sie auch den oben genannten Kriterien entsprechen.

## 3.9 Liste der Qualitätskriterien

*Informationen*

Q1 Wer ist der Betreiber der Seite?

Q2 Wer soll durch die Seite angesprochen werden?

Q3 Welche Inhalte werden durch die Seite beschrieben?

Q4 Welches Ziel wird durch die Seite verfolgt?

Q5 Auf welchem aktuellen Stand sind die Inhalte?

*Orientierung*

Q6 Wo befindet er sich gerade?

Q7 Wie kehrt er auf die Startseite zurück?

Q8 Welche Bereiche hat er schon aufgesucht?

Q9 Wo kann er noch hingehen?

*Hilfen*

Q10 Assistenten und geführte Touren können bei großen Inhalten helfen

Q11 Metaphern bieten dem Anwender vertraute Elemente

Q12 Orientierungselemente sollten intuitiv und schnell wieder erkannt werden

Q13 Die Zusammenfassung von Navigation und Orientierungshilfe in einem Element ist nicht immer von Vorteil

Q14 Zeitsparende Hilfen (zum Beispiel Inhaltsverzeichnisse) sind bei großem inhaltlichen Umfang angebracht

*Inhalte*

Q15 Schlecht aufbereitete Inhalte stoßen den Anwendern ab

*Text*

Q16 Texte sollten kurz und klar verfasst sein

Q17 Texte sollten inhaltlich und optisch strukturiert werden

Q18 Wesentliche Stichwörter der Inhalte sollten hervorgehoben sein

Q19 Lange Texte sollten mit einer kurzen Zusammenfassung beginnen

Q20 Lange Texte sollten von kurzen Texten getrennt werden

Q21 Typografie und Schriftgrößen beeinflussen den Text

Q22 Für jede Typografie sollte eine abgestimmte Alternative angeboten werden

Q23 Textblöcke, Worttrennungen und Zeilenumbrüche müssen berücksichtigt werden

Q24 Kursive und fette Texte sind in Web-Anwendungen schlecht lesbar

Q25 Der ausschließliche Einsatz von Großbuchstaben sollte vermieden werden

Q26 Schriftarten sollten nicht in einem Text gemischt werden

Q27 Optische Effekte innerhalb des Textes stören

Q28 Verschiedene Bildschirmauflösungen sind zu berücksichtigen

Q29 Sehr lange oder kurze Zeilen erschweren das Lesen (Richtwert: 40 bis 60 Zeichen pro Zeile)

Q30 Zeilenabstände sollten abgestimmt werden

Q31 Der Text sollte Leerräume enthalten, die als Ruhepunkte dienen

Q32 Linksbündige Texte sind leichter zu lesen als rechtsbündige

Q33 Scrollerbalken sollten vermieden werden, vor allem horizontale

Q34 Dem Anwender sollte das Ausdrucken des Textes angeboten werden

Q35 Der Text sollte nicht zu viel Raum innerhalb des gesamten Layouts einnehmen

Q36 Der Hintergrund darf nicht zu dominant sein

Q37 Hohe Helligkeitskontraste zwischen Text und Hintergrund erleichtern das Lesen

Q38 Starke Farbkontraste können einen Text schlecht lesbar machen

Q39 Ein weißer Hintergrund kann schnell zu Ermüdungen der Augen führen

*Bilder*

Q40 Bilder können Texte ergänzen

Q41 Bilder sollten nicht willkürlich auf der Oberfläche platziert werden

Q42 Bilder mit hoher Farbsättigung können vom Text ablenken

Q43 Monochrome Bilder fügen sich besser in den Kontext ein

Q44 Bilder können die Aufgabe von Orientierung und Navigation erfüllen

Q45 Bilder können für dekorative Zwecke eingesetzt werden

*Icons*

Q46 Icons sollten selbsterklärend sein

Q47 Die Bedeutung von Icons kann durch nebenstehende Begriffe klarer werden

Q48 Icons sind in erster Linie nach funktionalen Gesichtspunkten zu gestalten

Q49 Icons sollten einen durchgängigen ästhetischen Ausdruck haben

Q50 Icons sollten sparsam und gezielt zum Einsatz kommen

Q51 Die räumliche Anordnung von Icons sollte einem logischen Prinzip folgen

Q52 Icons sollten sinnvoll gruppiert werden

Q53 Icons können sich gegenseitig beeinflussen

Q54 Die Interpretation durch den Anwender ist zu berücksichtigen

*Animationen, Video und Ton*

Q55 Animationen sollten mit gängigen Playern wiedergegeben werden können

Q56 Es sollten Download-Links für Player angeboten werden

Q57 Eine Intro sollte nicht zu lang und überspringbar sein

Q58 Eine Überspringungsfunktion ist deutlich zu kennzeichnen

Q59 Lange Ladezeiten sollten vermieden werden

Q60 Eine Anwendung sollte auch ohne die Animation funktionieren können

Q61 Ton ist stets als untermahlendes Element zu betrachten

Q62 Es sollte eine Stummschaltungsfunktion angeboten werden

Q63 Zwischen Ladezeit und Qualität eines Videos sollte es einen Kompromiss geben

Q64 Verbindung und Computer des Anwenders sollten berücksichtigt werden

*Screen-Layout-Elemente*

Q65 Einzelne Screen-Layout-Elemente sollten in Bezug und Ergänzung zueinander stehen

Q66 Das Layout einer Web-Anwendung sollte einfach – klar und übersichtlich – gestaltet werden

Q67 Das für eine Web-Anwendung ausgewählte Layout sollte innerhalb der ganzen Web-Anwendung konsistent bleiben

*Gestaltungsgesetze*

Q68 Nah zusammen liegende Elemente werden als einander zugehörig angesehen

Q69 Ähnlich aussehende Elemente werden kognitiv einander zugeordnet

Q70 Um eine geschlossene Form wahrzunehmen, reicht eine Bezugnahme von offenen Elementen zueinander aus

Q71 Symmetrisch angeordnete Elemente werden als Einheit im Vordergrund aufgefasst, Asymmetrien auf den Hintergrund bezogen

Q72 Einfach und konsistent angeordnete Elemente bilden klare Strukturen und heben sich von Hintergrund ab

Q73 Kontinuierlich angeordnete Elemente werden als einander zugehörig gelesen

Q74 Unvollständige Abbildungen werden durch die visuelle Wahrnehmung komplettiert

*Farben*

Q75 Die Kombination verwandter Farben und monochromer Farbabstufungen wirkt ruhig und ausgeglichen

Q76 Komplementäre Farben können Spannungen erzeugen

Q77 Farben mit hohem Sättigungsgrad haben Signalcharakter

Q78 Intensive Farben sollten nicht großflächig eingesetzt werden

Q79 Dunkle Farben werden meist als angenehm empfunden. Kalte Farben werden räumlich weiter wahrgenommen als warme Farben

*Interaktions-Elemente*

Q80 Der Anwender möchte wissen, was der Computer für ihn tun kann

Q81 Was tut der Computer zurzeit?

Q82 Der Computer sollte alle Eingaben quittieren

Q83 Es sollte sofort erkennbar sein, ob der Computer noch ordnungsgemäß funktioniert

Q84 Die Konsistenz anklickbarer Elemente muss durchgängig sein

Q85 Auf jede Aktion des Anwenders sollte eine Gegenreaktion des Computers erfolgen

Q86 Interaktive Elemente sollten in ihrer Darstellung nicht mit statischen Elementen gleichgestellt oder vermischt werden

Q87 Durch die Anwendung geweckte Erwartungen sollten möglichst erfüllt werden

*Grundsätze der Dialoggestaltung*

Q88 Aufgabenangemessenheit

Q89 Selbstbeschreibungsfähigkeit

Q90 Steuerbarkeit

Q91 Erwartungskonformität

Q92 Fehlertoleranz

Q93 Individualisierbarkeit

Q94 Lernförderlichkeit

*Motivations-Elemente*

Q95 Eine Web-Anwendung sollte den Anwender motivieren, sie gern zu benutzen und sich mit ihr zu beschäftigen

Q96 Die subjektiv empfundene Arbeitsqualität sollte berücksichtigt werden

# 4 Entwurf

Für ein besseres Verständnis werden die einzelne Entwicklungsschritte und Architekturkomponenten anhand des in Abschnitt 1.3 genannten Anwendungsbeispiels “Benutzer- und Gruppenverwaltung“ beschrieben. Diese Beispielanwendung beschäftigt sich mit der Pflege von Benutzer- und Gruppenlisten. Sie bedient sich in diesem Fall einer relativ kleinen Menge von Eigenschaften, um die Funktionsweise des Systems leichter verständlich zu machen.

## 4.1 Konzept

In diesem Kapitel soll das Konzept des WebWidgets-Systems beschrieben werden. Das System hat die Aufgabe, Web-Oberflächen automatisch zu generieren. Wichtigster Aspekt ist die Einbeziehung von Qualitätskriterien. Der Ausgangspunkt für das Konzept ist eine Ontologie (hier: semantisch beschriebenes Datenmodell), welche ein einheitliches Schema für die Daten bereitstellt. Des Weiteren ermöglicht die Ontologie das Beschreiben dieser Daten mittels Zusatzinformationen. Die für den Einsatz des Systems notwendigen, durch den Entwickler durchzuführenden Schritte sind ebenfalls Bestandteil des Entwurfs. Auf diese Schritte wird im Text gesondert hingewiesen.

### 4.1.1 Anforderungen

Um eine funktionierende Web-Anwendung zu generieren, müssen bestimmte Anforderungen erfüllt werden. Diese Anforderungen sollen in jedem Fall durch eine vom System automatisch generierte Anwendungsschnittstelle erfüllt werden:

- Die vorhandenen Daten sollen visuell dargestellt werden
- Die notwendigen Interaktionen sollen vollständig unterstützt und im Kontext optimal dargestellt werden

- Die durch die Ausführung der Interaktionen erfasste Eingaben und Änderungen sollen im Form von Daten gespeichert werden
- Die gesamte Oberfläche soll sich nach jeder ausgeführten Interaktion stets dem neuesten Stand der Daten bedienen
- Falls es gewünscht ist, soll Mehrsprachigkeit unterstützt werden

### 4.1.2 Praktische Anwendung

Bestandteil des Konzepts ist eine durch den Entwickler jederzeit mögliche Erweiterung bestimmter Stellen innerhalb Angebots, das im WebWidgets-System enthalten ist. Diese Stellen werden im Folgenden genauer erläutert. Durch die Vorteile des Systems sollen die Entwickler ihrerseits dazu angeregt werden, Ergänzungen einzufügen und somit der Allgemeinheit zugänglich zu machen. Hierdurch wird unter anderem vermieden, dass wiederkehrende Aufgaben mehrfach gelöst werden müssen. Das System reift aufgrund dieser Erweiterungen und deckt ein immer größeres Feld an Bedürfnissen und Anforderungen ab. Ein wichtiger Punkt ist die Einhaltung des Gesamtkonzeptes. Daher werden externe Eingriffe nur auf bestimmte Bereiche beschränkt, deren Überarbeitung sinnvoll erscheint.

Wie diese Punkte erfüllt werden, wird im folgenden Abschnitt anhand der Architektur des Gesamtsystems beschrieben.

### 4.1.3 Architektur des Gesamtsystems

Bei WebWidgets-System erfolgt die Generierung der Anwendungsschnittstelle automatisch aus vorhandenen Daten. Diese Daten werden vom Dienstgeber bereitgestellt. Sie bestimmen die Darstellung der Anwendungsschnittstelle (hier: Webseite). Dies ist schematisch in der Abbildung 4.1 dargestellt.

Die Ontologie hält ein Schema für die Daten bereit, in welches die Daten vom Entwickler eingefügt werden (1). Dadurch werden die Daten in eine einheitliche und strukturierte Form gebracht. Die Daten werden innerhalb der Ontologie des Weiteren durch Zusatzinformationen (Annotationen) beschrieben (2).

Innerhalb des Systems werden die Daten anhand einer Liste von Qualitätskriterien analysiert(4). Das Ergebnis dieser Analyse ermöglicht eine Visualisierung der Daten in einem bestimmten Qualitätslevel (5).

Der Entwickler kann den Annotationen bei Bedarf bestimmte Werte zuweisen (3). Dementsprechend werden die Daten noch zweckorientierter visualisiert und dadurch in Form einer höheren Qualität dargestellt (6).

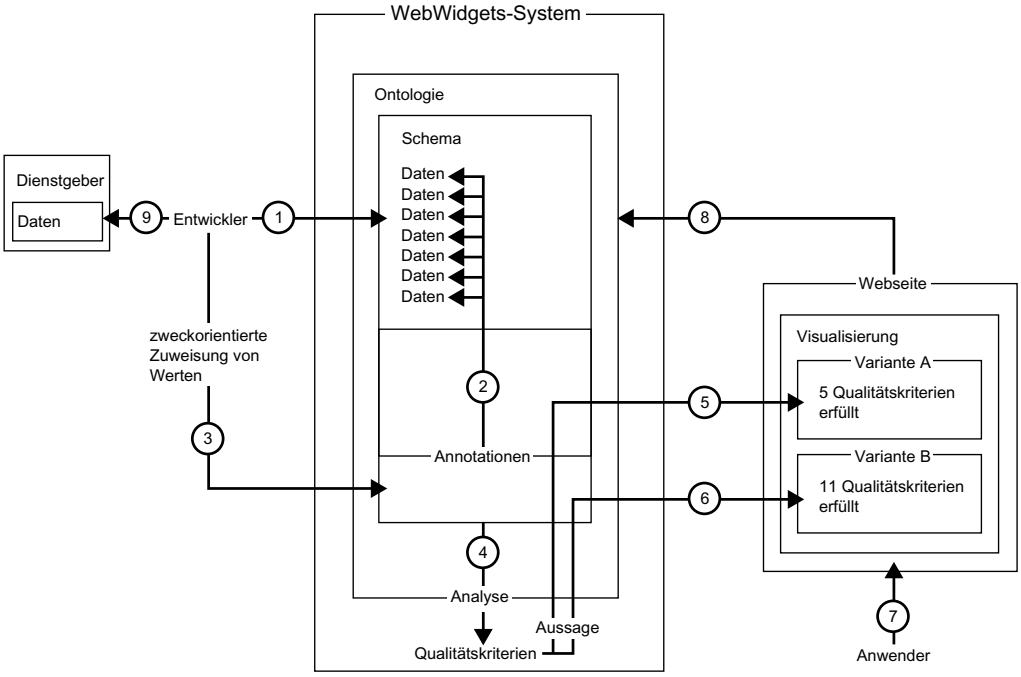


*Abbildung 4.1:* Architektur des Gesamtsystems

Der Anwender kann nun anhand der visualisierten Daten seine Eingaben innerhalb der Anwendungsschnittstelle machen (7). Diese Eingaben werden durch das System der Ontologie mitgeteilt (8). Die Ontologie informiert den Dienstgeber über den neuesten Stand der Daten (9). Der Entwickler hat prinzipiell dafür zu sorgen, dass der Daten- und Informationsfluss zwischen der Ontologie und dem Dienstgeber gewährleistet ist.

Die einzelnen Architekturkomponenten und deren Zusammenhänge werden in den folgenden Abschnitten ausführlich detailliert behandelt.

## 4.2 Ontologie des Systems

Eine Ontologie [UG96] ist im Allgemeinen ein formal definiertes System von Objekten und/oder Konzepten und Relationen zwischen diesen Objekten. In einer Ontologie werden Inhalte vom Schema getrennt. Das Schema wird durch die Konzepte einer Ontologie definiert. Die Inhalte werden als Instanzen abgebildet, die jeweilige Ausprägungen darstellen. Mittels Regeln sind die logischen Zusammenhänge zwischen Objekten der Ontologie definierbar.

Die definierten Regeln jeder üblichen Ontologie, sind vielfältig und können durch verschiedene Möglichkeiten definiert werden. Möglichkeiten sind beispielsweise Wenn-dann-Beziehungen, Zuweisungen, Negationen, logische Verknüpfungen oder mathematische Operationen.

### 4.2.1 Aufgabe und Anforderung

Die Ontologie[1] des WebWidgets-Systems bildet einen der Schlüsselbausteine der Systemarchitektur. Sie stellt ein einheitliches, für die Visualisierung verwendbares Schema für die durch den Dienstgeber bereitgehaltenen Daten zur Verfügung. Sie definiert neben der Struktur für die Dienstgeber-Daten die semantischen Zusatzinformationen, die für die automatische Visualisierung notwendig sind. Der Aufbau der Ontologie soll transparent, gut verständlich und einfach in der Anwendung sein: Geringe Komplexität ist in diesem Bereich anzustreben. Die funktionalen Vorteile der Ontologie müssen zugleich in vollem Umfang gewährleistet sein. Dadurch wird dem Entwickler eine deutliche Erleichterung der Handhabung ermöglicht.

### 4.2.2 Architektur und Anwendung

Die Ontologie muss alle Arten von Daten abbilden können. Gleichzeitig muss die Architektur der Ontologie einfach sein. Aus diesem Grund bietet die Ontologie keine Unterstützung für native Datentypen (Zahlen, Angabe des Datums usw.). Sämtliche Daten liegen lediglich in Textform vor. Es ist dann die Aufgabe eines Dienstgebers, die Umwandlung der Datentypen zu übernehmen.

Geht man von den beim Dienstgeber vorhandenen Daten aus, um eine Architektur der Ontologie zu konzipieren, wird das Ergebnis zu komplex. Einer der

[1] Die Grundidee für die Architektur der Ontologie kam vom Betreuer Dipl.-Inform. Max Völkel

Gründe dafür ist die Vielfältigkeit der vom Dienstgeber zur Verfügung gestellten nativen Datentypen. Ein weiterer Grund sind die unterschiedlichen Formen der Datenhaltung bei verschiedenen Dienstsgebern. Das Beschreibungsschema von HTML, das auf der Visualisierungsseite eingesetzt wird, ist vergleichsweise einfach aufgebaut. Es beschreibt zudem eine Hierarchie der HTML-Elemente. Daher wird im konkreten Fall von dieser Seite ausgegangen.

Die graphische Benutzeroberfläche einer Web-Anwendung kann als eine Zusammensetzung verschiedener Elemente (*Instanzen*) gesehen werden. Diese stehen zueinander in Form von definierbaren Verbindungen (*Links*). Diese einzelnen Elemente können dabei entsprechenden Typen (*Klassen*) zugeordnet werden. Die Datentypen stehen ebenfalls in definierbaren Verbindungen zueinander (*Relationen*). Relationen bestimmen ihrerseits Verbindungen zwischen einzelnen Elementen. Diese in der Visualisierung verwendete Struktur wird auf die Architektur der Ontologie übertragen.

Das Grundmodell der Ontologie ist in der Abbildung 4.2 zu sehen. Die beim Dienstgeber vorhandenen Daten werden nach deren semantischer Bedeutung in verschiedene Klassen eingeteilt. Dieser Schritt kann als Klassifikation der Daten in Typen betrachtet werden. Einzelne inhaltliche Ausprägungen von Daten werden in Form von Instanzen eingelesen. Diese Instanzen weisen eine Zugehörigkeit zu den definierten Klassen auf (1). Durch eine Relation wird eine Verbindung zwischen zwei Klassen beschrieben (2). Zwei Instanzen werden mit einem Link verbunden (3). Jeder Link ist einer Relation zugeordnet (4). Klassen und Relationen bilden somit gemeinsam das Schema. Instanzen und Links stellen die Inhalte dar. Überschriften, die sich auf die durch die Instanzen abgebildeten Inhalte beziehen, werden auf der Klassenebene festgelegt.

Anzahl, Benennungen und Zusammenhänge der verschiedenen Klassen sind vom jeweiligen Entwickler festzulegen. Sie sind abhängig vom Zweck der Anwendung. Im Beispiel der Benutzer- und Gruppenverwaltung wären dies folgende Arbeitsschritte:

- Klassifizierung der Benutzerliste mit zugehörigen Benutzern und deren Namen und Passwörtern
- Klassifizierung der Gruppenliste mit zugehörigen Gruppen und deren Gruppen-IDs

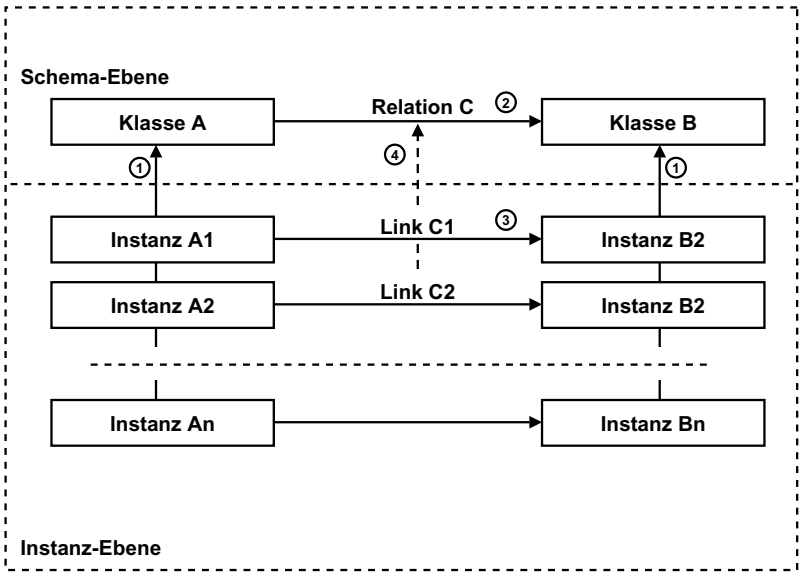


*Abbildung 4.2:* Architektur der Ontologie

- Definition aller relevanten Relationen
- Anlegen inhaltlicher Ausprägungen

Einzelne Ontologie-Komponenten werden innerhalb der jeweiligen Ontologie eindeutig identifiziert. Dies ermöglicht sowohl die Synchronisation von der jeweiligen Ontologie zu deren Visualisierung als auch zum jeweiligen Dienstgeber.

### 4.2.3 Der Ontologie-Baum

Die Hierarchie der Visualisierungs-Elemente bildet einen Baum. Die Ontologie-Komponenten müssen ebenfalls einen Baum bilden. So wird eine eindeutige Visualisierung der Ontologie-Daten ermöglicht. Aus dem Ontologie-Baum wird also ein Visualisierungs-Baum generiert. Die miteinander durch Relationen verbundenen Klassen finden auf der Schemaebene einen Klassenbaum. Parallel bilden die durch Links miteinander verbundenen Instanzen auf der Instanzen-Ebene einen Instanzen-Baum. Durch diese Trennung können beispielsweise Überschriften von Inhalten getrennt dargestellt werden.

### 4.2.4 Das Wurzelelement

Das Wurzelelement spielt in der Ontologie eine besondere Rolle. Seine genaueren Aufgaben werden weiter unten beschrieben. Es beinhaltet unter anderem anwendungsbezogene Informationen wie zum Beispiel den Titel der Anwendung, deren Autor oder deren Datenstand. Alle Instanzen erster Ebene stehen in direkter Anbindung zum Wurzelelement.

In der Abbildung 4.3 ist der Klassen-Baum des Beispiels “Benutzer- und Gruppenverwaltung“ dargestellt. Um diesen Baum zu erstellen, werden Klassen ohne Inhalte benötigt. Diese inhaltsfreien Klassen ermöglichen innerhalb des Baumes eine Gruppierung bzw. Zuordnung mehrerer Klassen mit definierten Inhalten. Eine Benutzerliste bzw. eine Gruppenliste wird durch die Menge zugehöriger Benutzer bzw. Gruppen visualisiert und hält diese gleichzeitig zusammen. Ein einzelner Benutzer wird durch seinen Namen, sein Passwort und seine Gruppe visualisiert, eine einzelne Gruppe durch ihre Gruppen-ID.

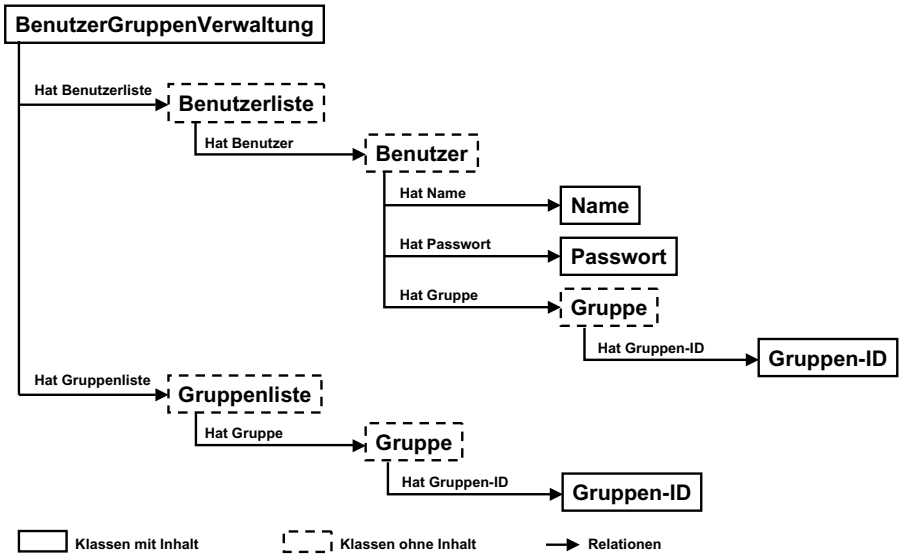


*Abbildung 4.3:* Klassen-Baum des Beispiels “Benutzer- und Gruppenverwaltung“

Die Klasse “Gruppe“ taucht an zwei Stellen auf. Innerhalb der Benutzerliste soll sie eine Auswahlmöglichkeit visuell darstellen. Für einen Benutzer wird eine bereits angelegte Gruppe ausgewählt. Innerhalb der Gruppenliste soll eine Ände-

rungsmöglichkeit für eine konkrete Gruppe dargestellt werden.

Anhand der vorhergehenden sowie weiteren allgemeinen Überlegungen wurde eine Liste möglicher Annotationen zusammengestellt.

### 4.2.5 Annotationen

Um dem Entwickler eine Erleichterung beim Arbeiten mit der Ontologie zu ermöglichen, werden die Annotationen mit Standardwerten versehen. An dieser Stelle werden die im Rahmen dieser Arbeit erstellten Annotationen aufgelistet:

1. Für Wurzelelement soll Autor definiert werden, da diese Information nicht der Gegenstand der durch beim Dienstgeber vorhandenen Daten ist
2. Für Wurzelelement soll Stand der Daten definiert werden, auch nicht Gegenstand der durch beim Dienstgeber vorhandenen Daten, soll aber laut dem Qualitätskriterium 5 in einer Web-Anwendung vorkommen
3. Für eine Klasse muss definiert werden, ob deren Instanzen Inhalte aufnehmen können. Die Gruppierung bzw. Zuordnung mehrerer Klassen mit definierten Inhalten geschieht durch eine Klasse ohne Inhalte
4. Für eine Klasse muss definiert werden, ob Inhalte ihrer Instanzen geändert werden dürfen. Allein aus dem Inhaltswert einer Instanz kann nicht bestimmt werden, ob sie geändert werden darf oder nicht
5. Für eine Klasse muss definiert werden, ob deren vorhandene Instanzen gelöscht werden dürfen. Allein aus dem Inhaltswert einer Instanz kann auch dies nicht bestimmt werden
6. Für eine Klasse muss definiert werden, ob neue Instanzen angelegt werden dürfen. Allein aus dem Inhaltswert einer Instanz kann dies nicht bestimmt werden
7. Für eine Klasse kann die Beschriftung festgelegt werden. Diese erscheint auf der Oberfläche der Anwendung
8. Für eine Verbindung zwischen zwei Klassen müssen Beziehungseinschränkungen definiert werden, welche die maximal und minimal mögliche Anzahl der Elemente auf beiden Seiten wiedergeben. Diese werden zum Beispiel beim Löschen oder Hinzufügen einer Instanz überprüft

9. Für eine Verbindung zwischen zwei Klassen kann festgelegt werden, ob sie eine Auswahlmöglichkeit darstellt. Die Notwendigkeit einer Auswahlmöglichkeit wurde oben bereits erwähnt

10. Um den Ontologie-Baum zu visualisieren, wäre es wichtig zu wissen, in welche Richtung die Visualisierung des Kinderknotens erfolgen soll. Diese Information könnte für eine Relation definiert werden. Eine Webseite wird durch den Browser von der linken Ecke nach rechts und/oder nach unten weiter aufgebaut. Mögliche Werte für die Richtung wären somit entweder "von links nach rechts" oder "von oben nach unten"

11. Für einen Link zwischen zwei Instanzen kann definiert werden, ob dieser aktiviert oder deaktiviert ist. Soll ein Link deaktiviert werden, so wird der ganze folgende Teilbaum deaktiviert. Im Zuge dessen kann er beispielsweise aus der Visualisierung ausgeschlossen werden. Dadurch können verschiedene Zugangsberechtigungen einer Ontologie in Abhängigkeit von vergebenen Benutzer-Rechten auf die Daten realisiert werden

12. Abgesehen von dem eigentlichen Inhalt und dessen Bezeichnung kann für eine Instanz ein Kommentar definiert werden

13. Für eine Instanz kann definiert werden, ob sie aktiviert oder nicht aktiviert ist. Dementsprechend können zwei verschiedene Visualisierungen realisiert werden. Diese Annotation könnte für eine Kartei-Darstellung von Nutzen sein (zum Beispiel Kartei geöffnet, geschlossen)

14. Für eine Instanz kann angegeben werden, dass ihre Inhalte Zeilenumbrüche zulassen, zum Beispiel für mehrzeilige Beschreibung. Allein aus dem Inhalt einer Instanz ist dies nicht zu ersehen

15. Für eine Instanz kann festgelegt werden, ob deren Inhalte verborgen werden sollen. Dies ist beispielsweise beim Einsatz von geheimen Passwörtern notwendig. Allein aus dem Inhalt einer Instanz ist dies nicht zu ersehen

Die für die Instanzen geltenden Annotationen können auch auf der Klassen-Ebene definiert werden. Sie würden somit für alle Instanzen der entsprechenden Klasse gelten. Die für eine Relation definierten Annotationen gelten für deren Links.

### 4.2.6 Interaktionen

Die Benutzeranwendungen stellen allerdings nicht nur die Daten und deren Verbindungen selbst dar, sondern sollen auch die Möglichkeit bieten, die Datenverarbeitung durchzuführen (Interaktionen). Die Interaktionen müssen ebenfalls definiert werden. Um die maximale Automatisierung der Erstellung einer Benutzer-Oberfläche zu erreichen und die Flexibilität gleichzeitig zu erhalten, werden die Interaktionen bereits auf der Ebene der Ontologie festgelegt.

Alle Interaktionen haben Folgendes gemeinsam:

- Sie sind Ontologie-Komponenten zugeordnet
- Sie müssen auf der Oberfläche visualisiert werden
- Sie müssen in der endgültigen Anwendung bestimmte Funktionen erfüllen

Wie das realisiert wird, wird durch das System vordefiniert. Der Entwickler soll sich an diese Vordefinitionen halten.

## 4.3 Visualisierung der Ontologie

In diesem Abschnitt wird beschrieben, wie die in der Ontologie vereinheitlichten Daten in einer Web-Anwendung dem Anwender in einer möglichst nützlichen Form automatisch präsentiert werden. Diese Aufgabe übernimmt das WebWidgets-System.

Zunächst wird das Grundkonzept beschrieben, das die Grundlage der automatischen Visualisierung bildet. Erst dieses Konzept ermöglicht diese Art der Darstellung. Danach werden für das Beispiel “Benutzer- und Gruppenverwaltung“ verschiedene Möglichkeiten der Visualisierung betrachtet. Anschließend wird das Grundkonzept durch zusätzliche Komponenten erweitert, die verschiedene Visualisierungsmöglichkeiten unterstützen, aus denen die optimale Möglichkeit ausgewählt wird.

### 4.3.1 Grundkonzept

Laut dem Abschnitt 4.2 werden die vorhandenen Daten in einer Ontologie mittels Klassen und Relationen, Instanzen und Links sowie eines abgesonderten Wurzelelements, das anwendungsspezifische Informationen enthält, abgebildet. Dem

entsprechend stellt das WebWidgets-System fünf Grundkomponenten zur Verfügung, die jeweils für die Visualisierung einer Ontologie-Komponente zuständig sind:

- *Applikations-Komponente* ist die Komponente, welche die Visualisierung des Ontologie-Wurzelelements übernimmt
- *Klassen-Komponente* ist die Komponente, welche die Visualisierung einer Ontologie-Klasse übernimmt
- *Relations-Komponente* ist die Komponente, welche die Visualisierung einer Ontologie-Relation übernimmt
- *Instanz-Komponente* ist die Komponente, welche die Visualisierung einer Ontologie-Instanz übernimmt
- *Link-Komponente* ist die Komponente, welche die Visualisierung eines Ontologie-Links übernimmt

Das Grundkonzept der Visualisierung ist in der Abbildung 4.4 dargestellt. Ausgehend von der Applikations-Komponente (1) wird der vollständige Ontologie-Baum oder ein Teil davon dynamisch und vollständig in einem Durchlauf mit Hilfe dieser Grundkomponenten visualisiert (2). Die Applikations-Komponente nimmt die durch den Anwender auf der Oberfläche vorgenommenen Eingaben und Änderungen entgegen (3) und leitet diese Informationen an die Ontologie zurück (4).

Die Visualisierung erfolgt durch das Abbilden der entsprechenden Ontologie-Inhalte auf geeignete HTML-Elemente (Textfelder, Auswahllisten etc.).

Für das Beispiel “Benutzer- und Gruppenverwaltung“, deren Ontologie-Baum in der Abbildung 4.3 zu finden ist, ist eine mögliche Visualisierung in der Abbildung 4.5 darstellt.

### Visualisierung von Interaktionen

Die Interaktionen sind einer Ontologie-Komponente zugeordnet. Deren Visualisierung ist ein Teil der Visualisierung der entsprechenden Ontologie-Komponente.

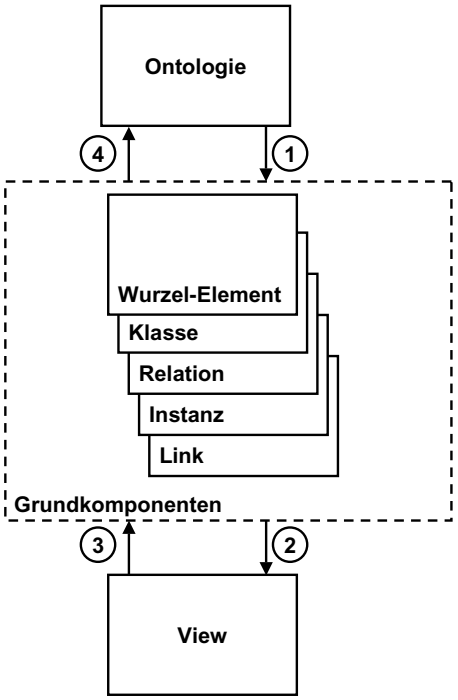


*Abbildung 4.4:* Grundkonzept der Visualisierung

### Sortierung

Die Daten der Ontologie bilden unsortierte Mengen und Listen. Lässt man diese unsortiert, so ist die Konsistenz der Visualisierung nicht gewährleistet, was den Verstoß gegen den Qualitätskriterium 67 bedeutet. Das System bietet verschiedene Sortierungsmöglichkeiten an. Beispiele hierfür können die Sortierung der Ontologie-Komponenten nach deren Hinzufügung zum Ontologie-Baum oder die Sortierung dieser Komponenten in alphabetischer Reihenfolge ihrer jeweiligen Inhalte sein.

Jeweils eine der angebotenen Sortierungsmöglichkeiten wird als Standard festgelegt und bei der Visualisierung angewandt. Jede vom System angebotene Sortierung kann in Form von Interaktionen in die Anwendungsschnittstelle eingebunden werden. Damit wird dem jeweiligen Anwender deren Steuerung offeriert.

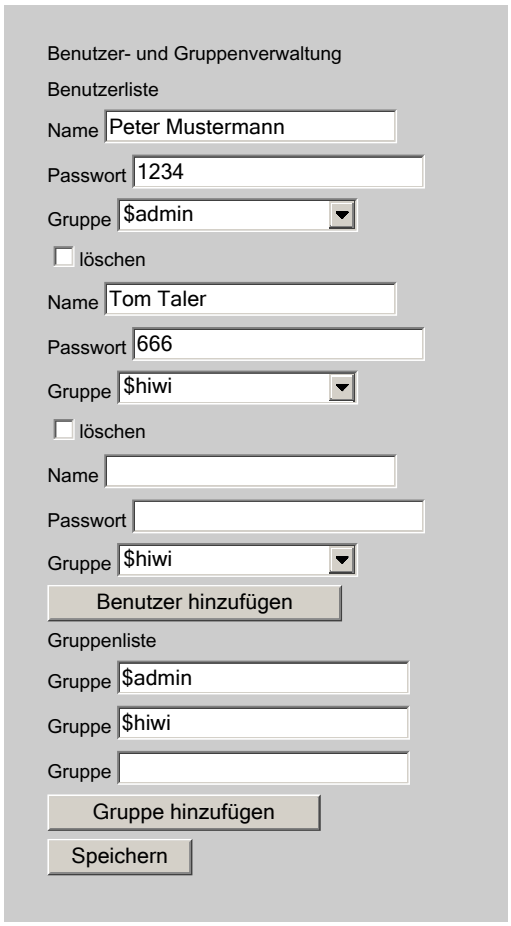


*Abbildung 4.5:* Benutzer- und Gruppenverwaltung

### 4.3.2 Visualisierungsmöglichkeiten

Die in Abschnitt 4.3.1 beschriebene Visualisierungsmöglichkeit hat im Angesicht der Benutzbarkeit einige Nachteile:

- Die Seite baut sich von oben nach unten auf, rechts entstehen Leerräume. Dies ist keine optimale Nutzung des für die Seite verfügbares Platzes. Die Seite wird dadurch auch schon bei kleineren Datenmengen lang, so dass die Anwender viel scrollen müssen – Verstoß gegen das Qualitätskriterium 33

- Der Titel der Anwendung ist durch die gleiche Schriftgröße dargestellt wie

die restlichen Inhalte und ist von diesen schwer zu unterscheiden – Verstoß gegen das Qualitätskriterium 17

- Die Benutzerliste und die Gruppenliste sowie einzelne Benutzer und Gruppen sind optisch schwer zu trennen – Verstoß gegen das Qualitätskriterium 17
- Die Beschriftungen Name, Passwort und Gruppe wiederholen sich für jeden einzelnen Benutzer und einzelne Gruppe – Verstoß gegen das Qualitätskriterium 16

Bessere Visualisierung, bei der die oben aufgelisteten Nachteile nicht mehr vorhanden sind, würde eine Tabellenform aus der Abbildung 4.6 liefern:

- Der Platz wird besser verwendet
- Der Titel ist kaum zu übersehen und wird als solche interpretiert
- Die Benutzerliste ist klar von der Gruppenliste durch Leerräume getrennt. Auch einzelne Benutzer bzw. Gruppen sind voneinander leichter zu unterscheiden, da jede in einer Zeile präsentiert ist und der Anwender jede neue Zeile als neuen Benutzer bzw. neue Gruppe visuell empfinden kann
- Die Beschriftungen sind im Tabellenkopf ausgegliedert und wiederholen sich nicht mehr

Sicherlich gäbe es eine weitere Visualisierungsmöglichkeit, die noch besser oder vergleichbar gut ist als die in der Abbildung 4.6 vorgeführte Tabellendarstellung, bei der ein Anwender im Falle einer längeren Benutzerliste wiederum gezwungen ist, zu scrollen, um den Gruppen-Verwaltungs-Bereich zu erreichen. Die in der Abbildung 4.7 dargestellte Karteidarstellung hätte sich für die Beispielanwendung auch gut geeignet und würde mit sich weitere Vorteile bringen:

- Die Aufmerksamkeit des Anwenders wird auf jeweils nur ein Hauptthema gelenkt – entweder auf die Verwaltung von Benutzern oder von Gruppen – ein weiteres Qualitätskriterium 66 ist erfüllt
- Der Übergang von einem Hauptthema zum anderen erfolgt schnell durch nur einen Klick und ohne viel scrollen zu müssen – und auch das Qualitätskriterium 33 ist erfüllt


*Abbildung 4.6:* Benutzer- und Gruppenverwaltung als Tabelle

### 4.3.3 Erweitertes Konzept

Nachdem nun festgestellt wurde, dass verschiedene Visualisierungsmöglichkeiten einer Ontologie existieren, stellen sich folgende Fragen:

- Wie werden die einzelnen Visualisierungsmöglichkeiten der Ontologie umgesetzt?
- Welche der möglichen Visualisierung ist optimal?

Die erste Frage wird durch den Einsatz verschiedener Komponenten (Renderer) gelöst. Ein Renderer kann die Visualisierung einer oder mehreren Ontologie-Komponenten übernehmen. Außer der eigentlichen Visualisierungsaufgabe soll ein Renderer auch die Frage beantwortet können, für welche Ontologie-Komponente eine Visualisierung geliefert werden kann und unter welchen Bedienungen dies geschieht. Ein “Kartei“-Renderer beispielsweise übernimmt die Visualisierung des Ontologie-Baumes, wenn die Anzahl der Instanzen der ersten Ebene kleiner als 7 ist. Ein “Textarea“-Renderer visualisiert Instanzen, deren Inhalt Zeilenumbrüche

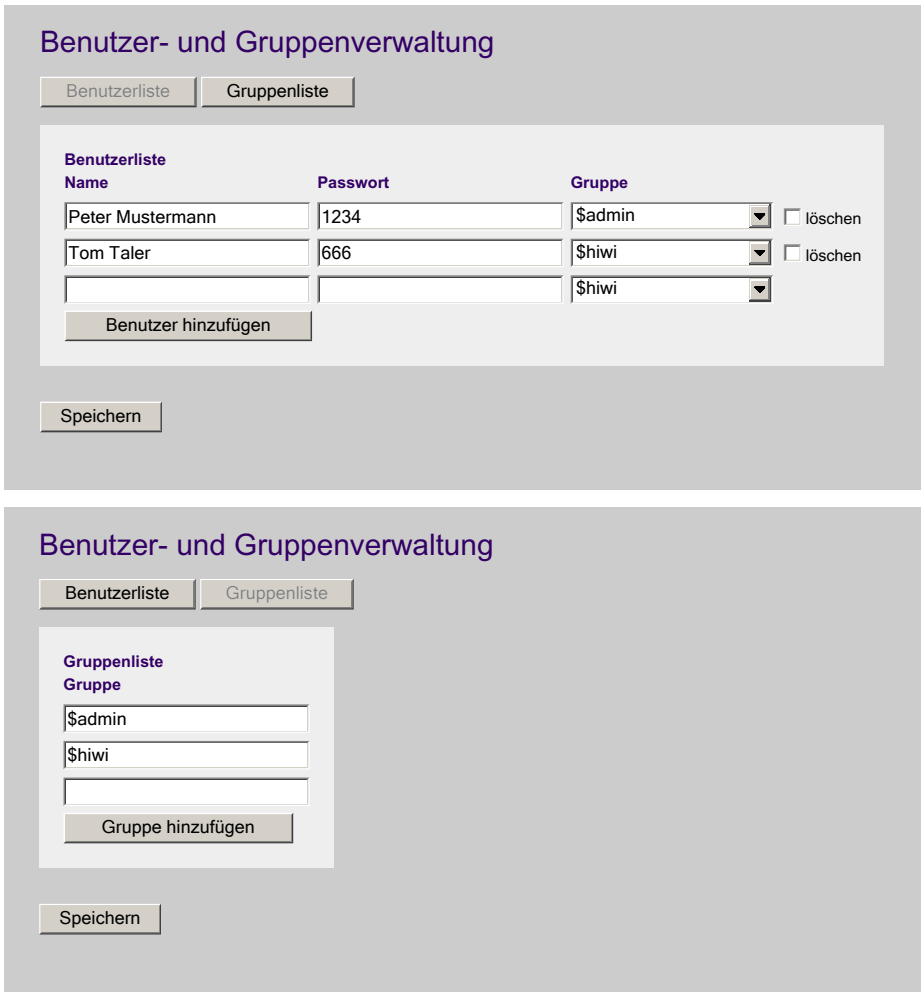


*Abbildung 4.7:* Benutzer- und Gruppenverwaltung als Kartei

zulässt. Das heißt, dass ein Renderer in der Lage sein sollte, eine Ontologie-Komponente sowohl anhand der definierten Annotationen als auch anhand weiterer Merkmale zu analysieren. Diese weiteren Merkmale lassen sich unmittelbar aus der Struktur der Ontologie ermitteln, wie zum Beispiel die Anzahl der Knoten, Nummer der entsprechenden Ebene, Größe der inhaltlichen Datenmenge einer Instanz etc.

Um unter den verschiedenen Renderern den optimalen auswählen zu können,

müssen die einzelne Renderer auch qualitativ bewertet werden. Für jeden Renderer wird ein Qualitätskoeffizient bestimmt und hinterlegt. Die Bestimmung des Qualitätskoeffizienten richtet sich nach der Anzahl erfüllter Qualitätskriterien. Diese werden dabei entsprechend ihrer Wichtigkeit unterschiedlich gewichtet. Die Auswahl des bestgeeigneten Renderers übernimmt der UI-Server. Dieser enthält eine Liste aller fertig gestellten Renderer.

Der vollständige Visualisierungsvorgang ist in der Abbildung 4.8 dargestellt. Beim Beginn der Visualisierung fragen die Grundkomponenten beim UI-Server an, ob er ihnen einen Renderer anbieten kann, der die Visualisierung der Ontologie-Komponente, für die sie zuständig sind, übernehmen kann. Dabei übergeben sie die entsprechende Ontologie-Komponente (zum Beispiel eine Instanz oder eine Verbindung zwischen zwei Instanzen) an den UI-Server (1). Der UI-Server fordert die in seiner Liste enthaltenen Renderer zu einer Analyse dieser Komponente auf (2). Darauf hin bekommt er von jedem Renderer eine Antwort, ob sie in der Lage sind, die Visualisierung zu übernehmen, und wenn ja, mit welchem Qualitätskoeffizienten (3). Der UI-Server wählt unter den in Frage kommenden Renderern denjenigen mit dem höchsten Qualitätskoeffizienten aus und übergibt dessen Namen an die Grundkomponenten. Wenn der UI-Server keinen Renderer liefern kann, so wird auch dies den Grundkomponenten mitgeteilt (4). Ist ein geeigneter Renderer vorhanden, so wird dieser von der Grundkomponente mit der Visualisierung beauftragt (5). Die Visualisierung wird dann vom Renderer durchgeführt (6). Existiert kein geeigneter Renderer, so visualisiert sich die Grundkomponente selbst nach der in Abschnitt 4.3.1 beschriebenen Art und Weise (7). Wiederum übernimmt die Applikations-Komponente den Datentransfer zwischen der Ontologie und Visualisierung.

Die fertig gestellten Renderer bilden in Form einer Bibliothek die Grundlage der Auswahl einer optimalen Komponente. Erst wenn diese Bibliothek eine ausreichende Anzahl an guten Renderern aufweist, ist eine Auswahl möglich.

Trotz einer umfangreichen Renderer-Bibliothek ist es möglich, dass ein Entwickler keine seinem Zweck entsprechende Visualisierungsmöglichkeit vorfindet. In diesem Fall findet der in Abschnitt 4.1 erwähnte Bereich für externe Eingriffe Anwendung: In diesem Fall erstellt der Entwickler einen eigenen Renderer. Der Name des Renderers wird dann der Liste des UI-Servers hinzufügt. Erst mit diesem Schritt ermöglicht der Entwickler sich die Anwendung des Renderers. Durch

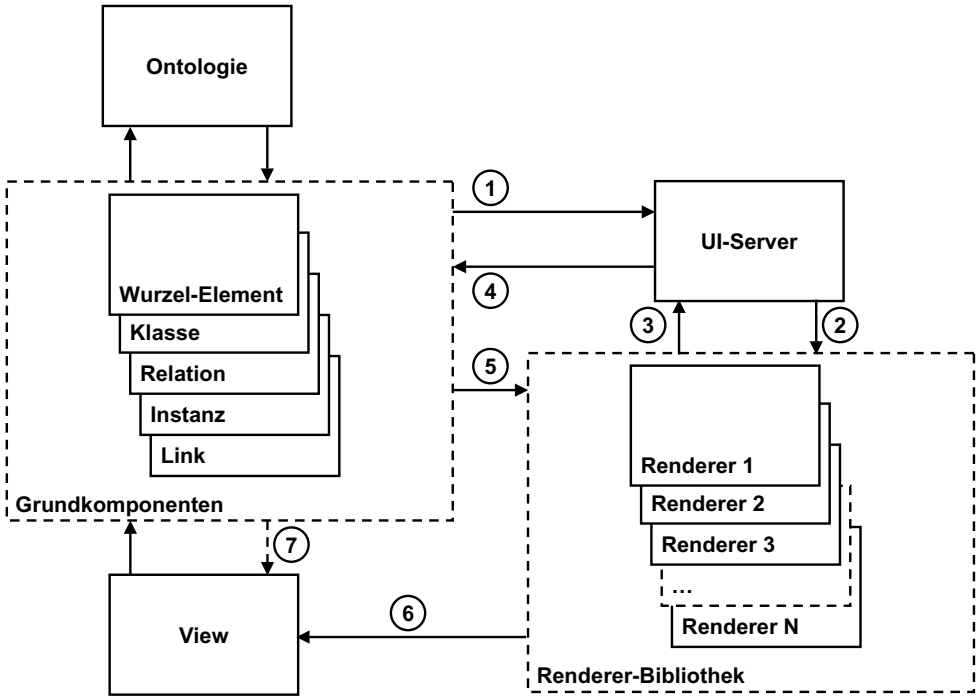


*Abbildung 4.8:* Auswahl der am besten geeigneten Visualisierung

eine Veröffentlichung wird jeder neu entwickelte Renderer auch für andere Entwickler nutzbar.

Aufgrund der Einfachkeit der Ontologie-Architektur wird eine begrenzte Anzahl an Annotationen vorgesehen. Die im WebWidgets-System enthaltenen Annotationen sollten die Visualisierung der Ontologie ausreichend unterstützen. Wird jedoch beispielsweise ein Renderer in einer unvorhergesehenen Art und Weise angewandt, kann es geschehen, dass sich die Annotationsliste als unzureichend erweist. In diesem Fall muss der Entwickler die Ontologie mit den entsprechenden Annotationen erweitern, die somit dem Renderer zur Verfügung stehen. Soll ein neu entwickelter Renderer der Allgemeinheit zur Verfügung gestellt werden, so müssen auch die zugehörigen Annotationen veröffentlicht werden.

## 4.4 Synchronisation zwischen Ontologie und Visualisierung

Unter der Synchronisierung zwischen der Ontologie und der Visualisierung werden zwei Fragen verstanden:

- Wie die durch eine Ontologie bereitgestellten Daten und definierte Interaktionen in einer Anwendungsschnittstelle dargestellt werden?
- Wie die durch einen Anwender in einer Anwendungsschnittstelle erfassten Eingaben und vorgenommenen Änderungen in der jeweiligen Ontologie widergespiegelt werden?

Die erste Frage wurde in Abschnitt 4.3 ausführlich behandelt. In diesem Abschnitt ist die zweite Frage zu beantworten.

Jede Ontologie-Komponente ist innerhalb der jeweiligen Ontologie eindeutig identifiziert. In Abschnitt 4.2 wurde geschrieben, dass die Visualisierung durch das Abbilden der in der entsprechenden Ontologie-Komponente gehaltenen Inhalte auf geeignete HTML-Elemente erfolgt. Jedem HTML-Element kann auch ein Name vergeben werden, durch den dieser seinerseits innerhalb einer Webseite eindeutig identifiziert ist. Die Verbindung zwischen der jeweiligen Ontologie und deren Visualisierung wird aufgebaut, indem man für jedes HTML-Element die gleiche Identifizierung auswählt, wie bei der entsprechenden Ontologie-Komponente, deren Inhalt das HTML-Element visualisiert.

Die durch einen Anwender vorgenommenen Änderungen werden an die Ontologie in Form einer Paar-Menge von Identifikationsbezeichnungen und eventuell aktualisierten Werten übergeben (Request-Map). Ähnlich wie die Visualisierung wird das Aktualisieren des jeweiligen Ontologie-Baums vom Wurzelelement angestoßen und an die Kinderknoten so lange weitergeleitet, bis der ganze Baum aktualisiert ist. Anhand seiner Identifizierung kann dabei jede Ontologie-Komponente dem Request-Map seinen neuen Wert entnehmen und sich damit aktualisieren.

Ist einer Klasse bzw. Instanz eine Interaktion zugeordnet, so wird deren Ausführung neben dem Aktualisieren der jeweiligen Klasse bzw. Instanz initiiert. Dabei hängt es von der Aktion ab, ob diese unmittelbar vor dem Aktualisieren oder gleich danach ausgeführt wird.

Im Beispiel “Benutzer- und Gruppenverwaltung“ können auf der Oberfläche gleichzeitig Benutzerdaten (der Name, das Passwort oder die zugeordnete Gruppe) geändert werden und das Löschen des entsprechenden Benutzers durch das Setzen des jeweiligen Heckchens initiiert werden. Die Interaktion “Benutzer löschen“ wird ausgeführt, bevor Benutzerdaten aktualisiert werden, da die Aktualisierung sich mit der Entscheidung für das Löschen erübrigt hat. Man kann auch sagen, dass das Löschen höhere Priorität als das Aktualisieren hat.

Das Anlegen eines neuen Benutzers erfolgt im Gegenteil zum Löschen vor dem Aktualisieren: Die entsprechenden Strukturen wie zum Beispiel eine Instanz der Klasse Benutzer, Name und Passwort werden zuerst angelegt und erst dann mit den übergebenen Werten ausgefühlt.

## 4.5 Synchronisation zwischen Dienstgeber und Ontologie

Ein Dienstgeber hält die Daten bereit. Die Ontologie definiert eine Struktur, die bereitgehaltene Daten unabhängig von der Art deren Datenhaltung aufnehmen kann. Der Entwickler soll nach den durch die Ontologie definierten Regeln eine Schnittstelle zwischen dem Dienstgeber und der Ontologie fertig stellen, die diese beiden Komponenten miteinander verbindet. Dabei soll die Verbindung in beide Richtungen funktionieren.

Um die Daten vom Dienstgeber zur Ontologie zu übergeben, sollen folgende Schritte gemacht werden:

- Die Daten müssen klassifiziert werden. Für jeden Datentyp muss eine Klasse benannt werden
- Relationen, die Zusammenhänge zwischen den einzelnen Klassen abbilden, müssen definiert werden
- Die vorgesehenen Interaktionen müssen definiert und können annotiert werden
- Für konkrete Datenausprägungen müssen Instanzen einer entsprechenden Klasse angelegt werden. Diese müssen mit Dateninhalten gefüllt werden
- Die einzelnen Instanzen müssen miteinander verlinkt werden

Nach jedem dieser Schritte können beim Bedarf die Standardwerte für die Annotationen der jeweiligen Ontologie-Komponente geändert werden.

Die Ontologie nimmt vom Dienstgeber die Daten an, leitet diese zur Anwendungsschnittstelle und benachrichtigt den Dienstgeber über die durch einen Anwender eventuell vorgenommen Änderungen, die dieser seinerseits übernimmt. Dabei muss er die Aufgabe der Umwandlung der Datenformate übernehmen, da nativen Datentypen durch die Ontologie nicht unterstützt werden und sämtliche Daten bei ihr in einer Textform vorliegen. Bevor dies geschieht, müssen die neuen Daten durch den Dienstgeber auf ihre Richtigkeit überprüft werden (Validität). Wenn die Überprüfung ergibt, dass die Daten richtig sind, werden sie bei der ursprünglichen Datenausprägung des Dienstgebers aktualisiert. Soll sich dagegen feststellen, dass die übergebenen neuen Daten in einem falschen Format vorliegen, wird die Aktualisierung verweigert. Dabei muss die Ontologie mittels einer entsprechenden Meldung darüber benachrichtigt werden. Diese Meldung ist eine Text-Meldung, aus der ein Anwender nachvollziehen kann, was er falsch eingegeben hat. Die Ontologie ihrerseits ist bereit, eine Fehlermeldung aufzunehmen und, wenn eine vom Dienstgeber übergeben wird, die entsprechenden fehlgeschlagenen Änderungen zurückzusetzen und die übergebene Meldung zur Visualisierung der Anwendungsschnittstelle weiterzuleiten.

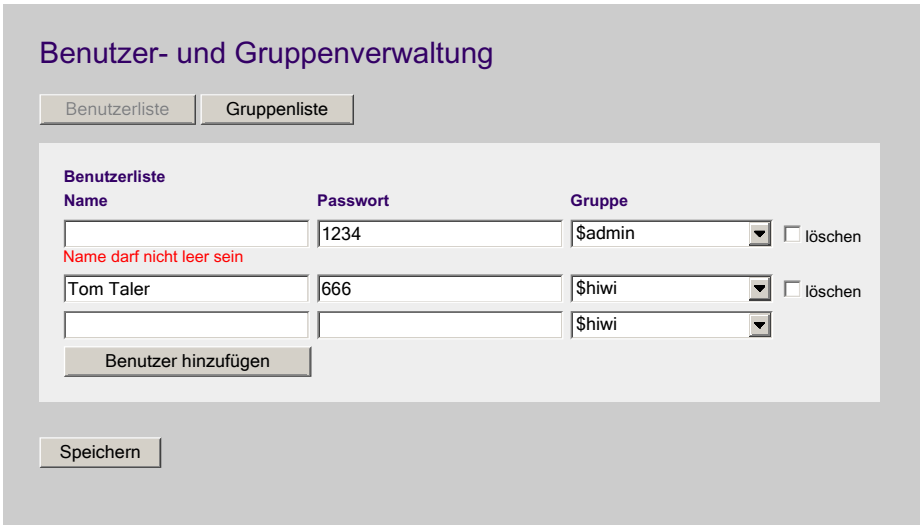


*Abbildung 4.9:* Weiterleitung der Fehlermeldungen

Die vom WebWidgets-System automatisch generierte Weboberfläche lässt es zu, mehrere Instanzen auf einen Schlag zu aktualisieren. Die Validierung der Daten ist dabei Instanz-bezogen: Die Aktualisierung wird nur bei den Instanzen nicht durchgeführt, bei denen die Daten falsch erfasst wurden. Unabhängig davon werden alle anderen Instanzen, bei denen die Inhalte in einem richtigen Format eingegeben wurden, aktualisiert. Die Fehlermeldung wird auch an die entsprechende Instanz weitergeleitet, so dass diese auch Teil der Inhalte der jeweiligen Instanz wird und bei der erneuten Visualisierung zusammen mit den restlichen Inhalten steht (Die Meldung "Name darf nicht leer sein" aus der Abbildung 4.9). Dadurch kann ein Anwender sofort sehen, an welchen Stellen die Eingaben falsch waren, und diese entsprechend korrigieren. Pro Instanz kann nur eine Fehlermeldung gesetzt werden.

Um das Qualitätskriterium 85 zu erfüllen, können nicht nur die Fehlermeldungen, sondern auch andere Mitteilungen des Dienstgebers an einer Ontologie-Komponente übergeben werden. Dabei gibt es auch eine Meldung pro eine Ontologie-Komponente. Diese Meldungen werden wie die Fehlermeldungen durch die Ontologie aufgenommen und an die Visualisierung weitergeleitet (Die Meldung "Alle Änderungen wurden vollständig vorgenommen" aus der Abbildung 4.10 ).

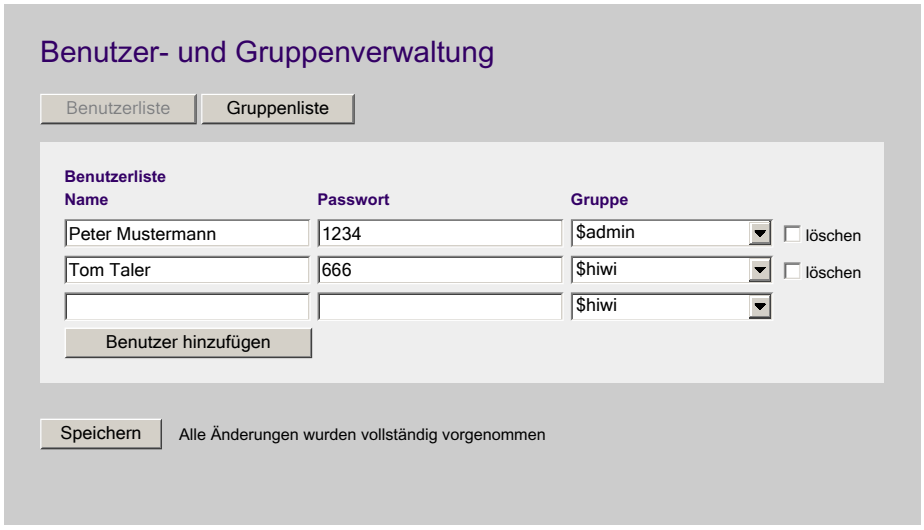


*Abbildung 4.10:* Weiterleitung der Meldungen

## 4.6 Internationalisierung

Wird eine Unterstützung der Mehrsprachigkeit benötigt, so wird diese bereits auf der Ontologie-Ebene realisiert, da die Ontologie alle für eine Benutzeroberfläche notwendigen Informationen enthält. Dies erfolgt, indem der jeweilige Dienstgeber für jede Sprache eine andere Ontologie zusammenstellt.

## 4.7 Flexibilität

### 4.7.1 Verwendung von Stylesheets

Cascading Style Sheets (CSS) ist schon seit vielen Jahren ein Teil der Webtechnologie. Mit Stylesheets können Gestaltungsaspekte wie Farben, Schriftarten, Schriftgrößen etc. näher spezifiziert werden. Diese Aspekte werden in Form von Formateigenschaften einzelner HTML-Elemente je nach Wunsch beschrieben und mit einer Webseite als unmittelbare Ergänzung zu HTML verbunden. Dies kann auf insgesamt drei unterschiedliche Weisen geschehen: CSS kann direkt in HTML-Befehle eingebunden, im Kopf einer Webseite angeordnet oder als separate Datei erstellt werden.

Die unmittelbare Einbindung von CSS in HTML-Befehle schränkt dessen Anwendungsmöglichkeiten stark ein, da es sich jeweils nur auf den entsprechenden Befehl bezieht. Für jede weitere Anwendung an anderer Stelle muss es kopiert und neu eingefügt werden. Dadurch wird der HTML-Code unübersichtlich und unnötig lang. Die Anordnung von CSS im Kopf einer Webseite hingegen hat den Vorteil, dass die Stylesheets-Definitionen zentral angeben werden und in beliebig vielen HTML-Elementen innerhalb einer Seite verwendet werden können. Die Ausführung als separate Datei hat zusätzlich den Vorteil, dass CSS in beliebig vielen verschiedenen Seiten angewandt werden kann.

Das WebWidgets-System lässt die Stylesheet-Technik nicht außen vor und bietet die Möglichkeit, diese einzusetzen. Dafür können Stylesheets-Klassen definiert und deren Namen entsprechenden Ontologie-Komponenten zugewiesen werden. Die Ontologie-Komponenten werden durch das WebWidgets-System mit Hilfe von HTML-Elementen visualisiert. Ist eine Stylesheets-Klasse einer Ontologie-Komponente zugewiesen, kümmert sich das WebWidgets-System gleichzeitig darum, dass diese auch dem entsprechenden HTML-Element zugeordnet wird. Ein

Entwickler muss sich dann nur darum kümmern, dass die Stylesheets-Definitionen auf einem der oben erwähnten Wege korrekt eingebunden werden.

### 4.7.2 Benutzerdefinierte Auswahl einer Visualisierungsmöglichkeit

In Abschnitt 4.3.3 wurde die Auswahl einer am besten geeigneten Visualisierungsmöglichkeit der Ontologie-Komponenten beschrieben. Eine Visualisierungsmöglichkeit wird mittels eines Renderers realisiert. Hat sich ein Entwickler für eine bestimmte Visualisierung entschieden, so kann er den durch die WebWidgets-System bereitgestellten Auswahlmechanismus für eine beliebige Zahl an Ontologie-Komponenten ausschalten. Die Auswahl wird dann vom Entwickler selbst übernommen. Er teilt dazu der entsprechenden Ontologie-Komponente den Namen des gewünschten Renderers mit. Anhand dieses Namens übergibt dann das WebWidgets-System die Visualisierung der entsprechenden Ontologie-Komponente direkt an den durch den Entwickler benannten Renderer. Die Ontologie kann hierdurch unter Umständen schneller visualisiert werden, da auf den Vorgang der Analyse durch die WebWidgets-System verzichtet werden kann. Da es durch die Deaktivierung der Analysefunktion des WebWidgets-Systems nicht zu einer Überprüfung aller optimalen Möglichkeiten kommt, kann hierdurch eventuell nicht das höchstmögliche Qualitätsniveau erreicht werden.

### 4.7.3 Benutzerdefinierte Interaktionen

Zusätzlich zu den vom WebWidgets-System bereitgestellten Interaktionen, soweit diese nicht ausreichend sind, kann ein Entwickler eigene anwendungsspezifische Interaktionen realisieren und in das WebWidgets-System einbinden. Diese Interaktionen unterliegen den gleichen Festlegungen wie alle anderen Interaktionen (Siehe Abschnitt 4.2.6) Wie die verschiedenen Renderer können auch die benutzerdefinierten Interaktionen zur allgemeinen Verwendung veröffentlicht werden.

# 5 Implementierung

Die Implementierung des Prototyps wurde mit der Programmiersprache Java umgesetzt. Der Visualisierungsvorgang wurde zudem mit Hilfe von JSF implementiert.

## 5.1 Ontologie

Die Ontologie besteht aus dem Wurzelelement, in dem anwendungsspezifische Daten abgebildet werden, aus den durch Relationen miteinander verbundenen Klassen sowie aus den mittels Links miteinander verbundenen Instanzen. Weiterhin sind die Interaktionen ebenfalls auf der Ebene der Ontologie festgelegt. Das vereinfachte Klassendiagramm der Klassen, welche Ontologie-Strukturen implementieren, ist in der Abbildung 5.1 dargestellt.

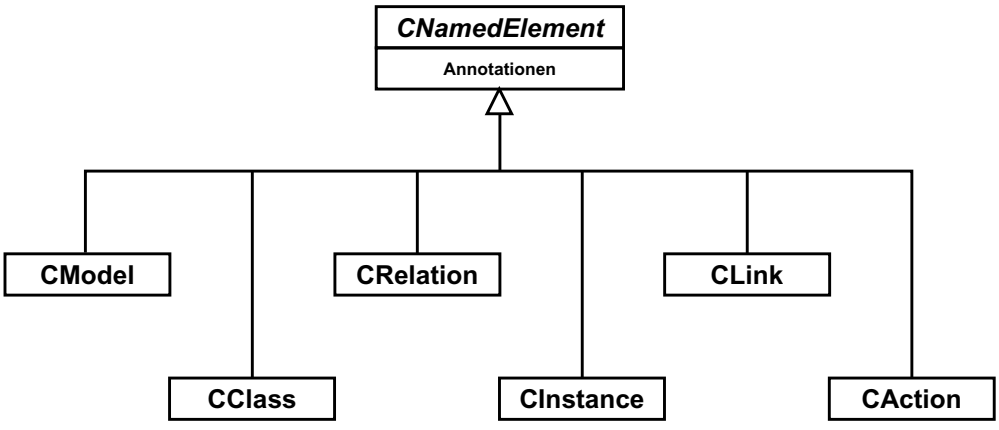


*Abbildung 5.1:* Ontologie-Klassendiagramm

### CNamedElement

Alle Annotationen, die in verschiedenen Ontologie-Komponenten verwendet werden, sind an einer zentralen Stelle abgelegt, und zwar als Variablen der Klasse *CNamedElement*. Alle weiter unten folgenden Klassen, die Ontologie-Komponenten implementieren, erben von der Klasse *CNamedElement* diese Eigenschaften. Das WebWidgets-System kann bei der Visualisierung bestimmte Annotationen im Bezug auf die jeweilige Ontologie-Komponenten interpretieren.

### CModel

Die Klasse *CModel* implementiert das Wurzelelement der Ontologie. In dieser Klasse werden die Referenzen der Instanzen erster Ebene des Ontologie-Baumes in Form einer Menge eingebunden.

### CClass

Die Klasse *CClass* bildet eine Ontologie-Klasse ab. Die Baumstruktur, die Ontologie-Klassen bilden, ist dadurch realisiert, dass jede Ontologie-Klasse zwar viele ausgehende Relationen haben kann, aber nur durch eine eingehende Relation mit einer Ontologie-Klasse der oberen Ebene verbunden ist. Die eingehende Relation und die Liste ausgehender Relationen werden hier mittels entsprechenden Methoden gepflegt. Damit kann der jeweilige Ontologie-Klassen-Baum zusammengestellt und in diesem navigiert werden. Durch die Navigationsmechanismen in dem jeweiligen Ontologie-Klassen-Baum ist der Zugriff auf jede einzelne Ontologie-Klasse gewährleistet.

In jeder Ontologie-Klasse ist festgelegt, ob die Werte ihrer Instanzen geändert werden dürfen, ob die jeweilige Ontologie-Klasse eine neue Instanz aufnehmen darf und ob eine oder mehrere der bereits vorhandenen Instanzen gelöscht werden dürfen.

Jede Ontologie-Klasse kennt deren Instanzen. Dazu werden diese in einer Liste gepflegt. Dazu werden entsprechende Methoden bereitgestellt, mit denen eine neue Instanz bzw. eine vorhandene Instanz der jeweiligen Ontologie-Klasse angelegt bzw. gelöscht wird, sobald dies erlaubt ist.

Einer Ontologie-Klasse können verschiedene Ontologie-Interaktionen zugeordnet werden. Die Klasse *CClass* hält für diese eine Liste bereit.

### CRelation

Die Klasse *CRelation* bildet eine Ontologie-Relation ab, mit der zwei Ontologie-Klassen verbunden werden können. Jede Ontologie-Relation kennt die beiden mit deren Hilfe verbundenen Ontologie-Klassen. Weiterhin legt sie Verbindungsregeln fest, nach denen die einzelnen Ontologie-Instanzen der entsprechenden Ontologie-Klassen verbunden werden. Diese Regeln definieren die maximal und minimal mögliche Anzahl der Beziehungen auf den beiden Seiten.

### CInstance

Die Klasse *CInstance* bildet eine Ontologie-Instanz ab. Jede Ontologie-Instanz kennt die entsprechende Ontologie-Klasse, zu der diese zugeordnet ist.

Jede Ontologie-Instanz pflegt den eingehenden Link und eine Liste ausgehender Links. Ähnlich wie beim Klassen und Relationen ist auch hier die Ontologie-Instanzen-Baumstruktur dadurch gewährleistet: Jede Instanz kann zwar viele ausgehenden Links haben, ist aber nur durch einen eingehenden Link mit der Instanz der oberen Ebene verbunden.

Jede Ontologie-Instanz hat einen Wert. Dieser Wert darf durch den Anwender geändert werden, sobald dies die zugeordnete Klasse zulässt.

Einer Ontologie-Instanz können verschiedene Ontologie-Interaktionen zugeordnet werden. Die Klasse *CInstance* hält für diese eine Liste bereit.

### CLink

Die Klasse *CLink* bildet einen Ontologie-Link ab, mit dem zwei Ontologie-Instanzen verbunden werden. Diese Verbindung ist einer Relation zugeordnet. Jeder Link hat die Referenz auf das Objekt der Relation, zu der er zugeordnet ist.

Jeder Link pflegt auch die Referenzen auf beide mit dessen Hilfe verbundenen Ontologie-Instanzen. Dabei wird zwischen der Ursprung-Instanz (source-Instanz) und Ziel-Instanz (target-Instanz) unterschieden. Ein Link kann geändert werden, indem ihm eine neue target-Instanz zugewiesen wird.

#### CAction

Die Klasse *CAction* bildet die Ontologie-Interaktionen schematisch ab. Damit werden die Regeln festgelegt, welche jede Ontologie-Interaktion befolgen soll. Die Realisierung einer konkreten Ontologie-Interaktion soll in einer Klasse implementiert werden, die von der Klasse *CAction* erbt.

Eine Ontologie-Interaktion kann einer Ontologie-Komponente zugeordnet werden. Dafür pflegt die Klasse *CAction* eine Referenz auf das Objekt der entsprechenden Ontologie-Komponente.

Für die auszuführende Funktionalität stellt die Klasse *CAction* die Methode *processAction()* bereit.

## 5.2 Visualisierung der Ontologie

### 5.2.1 Grundkonzept

Die im Konzept für den Visualisierungsvorgang vorgesehenen Grundkomponenten wurden mit folgenden JSF-Komponenten realisiert:

- *WApplicationComponent* setzt die Applikations-Komponente um
- *WClassComponent* setzt die Klassen-Komponente um
- *WRelationComponent* setzt die Relations-Komponente um
- *WInstanzComponent* setzt die Instanz-Komponente um
- *WLinkComponent* setzt die Link-Komponente um

In der Abbildung 5.2 ist die Implementierung des Visualisierungsvorgangs darstellt.

Im Folgenden werden die einzelnen Schritte des Visualisierungsvorgangs beschrieben. Die Aufzählung entspricht der Nummerierung in der Abbildung 5.2. Sollte bei einem dieser Schritte ein Fehler vorgekommen, so wird dieser direkt in der jeweiligen JSF-Seite ausgegeben. Der Visualisierungsvorgang bricht dabei ab (gestrichelter Pfeil in der Abbildung):

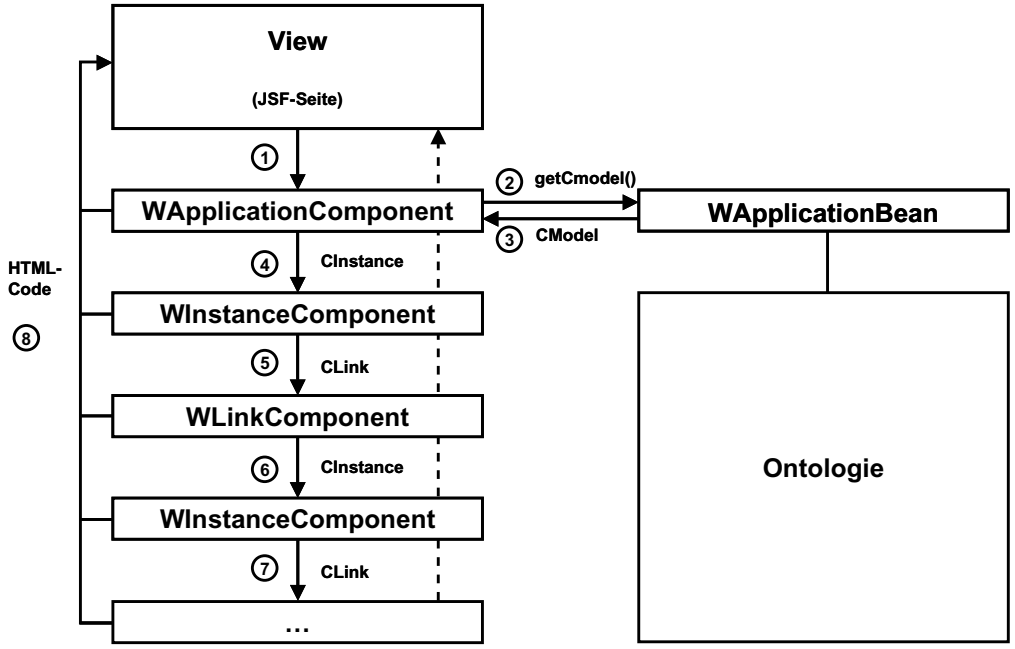


*Abbildung 5.2:* Grundkonzept der Visualisierung

1. Der Visualisierungsvorgang wird von einer JSF-Seite angestoßen. In dieser ist die Applikations-Komponente mit Hilfe der Klasse *WApplicationTag* eingebunden

2. Der Datenaustausch zwischen der Ontologie und der Visualisierung wird durch die JSF Managed Bean Technologie unterstützt. Die Klasse *WApplicationBean* hält die Referenz für das Ontologie-Wurzelelement bereit (Instanz der Klasse *CModel*). Die Applikations-Komponente fordert diese Referenz an

3. Bekommt die Applikations-Komponente die Referenz des Wurzelelements, so kann sie mit dessen Visualisierung beginnen. Mit der Referenz des Wurzelelements bekommt die Applikations-Komponente nicht nur den Zugriff auf dieses, sondern auch auf den ganzen Ontologie-Baum

4. Nach der erfolgten Visualisierung des Wurzelelements übergibt die Applikations-Komponente die Visualisierung an die für die Ontologie-Instanzen

zuständigen Instanz-Komponenten. Dies erfolgt in einem Zyklus über die Ontologie-Instanzen, die mit dem Wurzelelement verbunden sind: Für jede Ontologie-Instanz wird eine Instanz-Komponente bereitgestellt, welche die Referenz auf die entsprechende Ontologie-Instanz erhält. Die jeweilige Instanz-Komponente visualisiert die entsprechende Ontologie-Instanz

5. Nachdem die Ontologie-Instanzen der ersten Ebene visualisiert sind, wird die Visualisierung des Ontologie-Baumes an deren Verbindungen zu den Instanzen zweiter Ebene weitergeleitet. Dies Erfolgt ebenfalls in einem Zyklus: Für alle Verbindungen der jeweiligen Instanzen ersten Ebene wird eine Link-Komponente bereitgestellt. Die jeweilige Link-Komponente visualisiert den entsprechenden Ontologie-Link

6. Jetzt sind die Links visualisiert, deren Ursprungs-Instanzen die Ontologie-Instanzen der ersten Ebene sind. Die Ziel-Instanzen dieser Links sind die Ontologie-Instanzen der zweiten Ebene. Der jeweilige Link übergibt die Visualisierung seiner Ziel-Instanz wiederum an eine Instanz-Komponente, die zu diesem Zweck neu angelegt wird

7. Sind die Instanzen zweiter Ebene visualisiert, so wird die Visualisierung entsprechend der Schritte 5 und 6 an die Ontologie-Links und anschließend an die Ontologie-Instanzen der unteren Ebenen weitergeleitet. Dies geschieht so lange, bis der ganze Ontologie-Baum auf dem Visualisierungs-Baum abgebildet ist

8. Das Ergebnis der Visualisierung, die in den obigen Schritten erfolgt, ist ein durch die entsprechenden Ontologie-Komponenten zusammengestellter HTML-Code, der deren Inhalte abbildet. Dieser HTML-Code wird schrittweise der JSF-Seite hinzugefügt

Die Zusammenstellung des HTML-Codes geschieht in Abhängigkeit der unterschiedlichen Werte verschiedener Annotationen. Dies geschieht unter Anwendung der Fallunterscheidung.

#### Sortierung

Um die Sortierung zu ermöglichen, implementiert die Klasse *CNamedElement* und somit alle Ontologie-Komponenten das Interface *Comparable*. In der Methode *compareTo()* wird ein zum Sortieren notwendiger Vergleich zweier Ontologie-Komponenten nach deren Zeitpunkt des Hinzufügens zum Ontologie-Baum durchgeführt.

### 5.2.2 Erweitertes Konzept

Das Grundkonzept der Visualisierung unterstützt bisher noch keine Vielzahl verschiedener Visualisierungsmöglichkeiten. Zu diesem Zweck wird es durch eine Renderer-Bibliothek und einen UI-Server erweitert.

Einzelne Renderer dieser Bibliothek sollen eine Visualisierungsmöglichkeit einer Ontologie-Komponente umsetzen. Sie erben alle Eigenschaften und Methoden der JSF-Klasse *javax.faces.render.Renderer* und implementieren das vom WebWidgets-System bereitgestellte Interface *WRenderer*.

Der UI-Server übernimmt die Vorauswahl der optimalen Visualisierungsmöglichkeit und wird durch die Klasse *UIServer* implementiert, welche die entsprechenden Methoden liefert.

Die Abbildung 5.3 beschreibt diesen Vorgang:

1. Jede der fünf Grundkomponenten ruft beim Beginn der Visualisierung der Ontologie-Komponente, für die sie zuständig ist, die Methode *getRendererType()* von der Klasse *UIServer* auf. Diese Methode erhält im Zuge dessen die Referenz der entsprechenden Ontologie-Komponente

2. Der UI-Server reicht diese Referenz an jeden Renderer seiner Liste weiter. Dies geschieht durch den Aufruf der durch jeden Renderer implementierten Methode *getQualityCoefficient()*. Dadurch wird bei jedem Renderer abgefragt, ob er die Visualisierung der jeweiligen Ontologie-Komponente übernehmen kann

3. In der Methode *getQualityCoefficient()* ist die Analyse einer Ontologie-Komponente implementiert. Als Ergebnis dieser Analyse erhält der UI-Server von jedem Renderer eine Zahl – den Qualitätskoeffizienten

4. Sollte ein Renderer seine Eignung für die Visualisierung ausschließen, so gibt dieser dem UI-Server den Wert -1 als Qualitätskoeffizient an. Unter den übrigen Renderern wählt der UI-Server denjenigen mit dem höchsten Qualitätskoeffizient aus. Sollten mehrere Renderer einen identischen Wert zurückgegeben haben, so wird derjenige verwendet, der als erster in die UI-Server-Liste eingetragen wurde. Wurde ein Renderer ausgewählt, gibt der UI-Server dessen Namen an die Grundkomponente zurück. Wurde hingegen kein Renderer ausgewählt, übermittelt der UI-Server den Wert Null

5. Erhält die jeweilige Grundkomponente vom UI-Server den Namen eines Renderers, übergibt sie die Visualisierung an diesen

6. Erhält sie einen Null-Wert, übernimmt die Grundkomponente die Visualisierung selbst

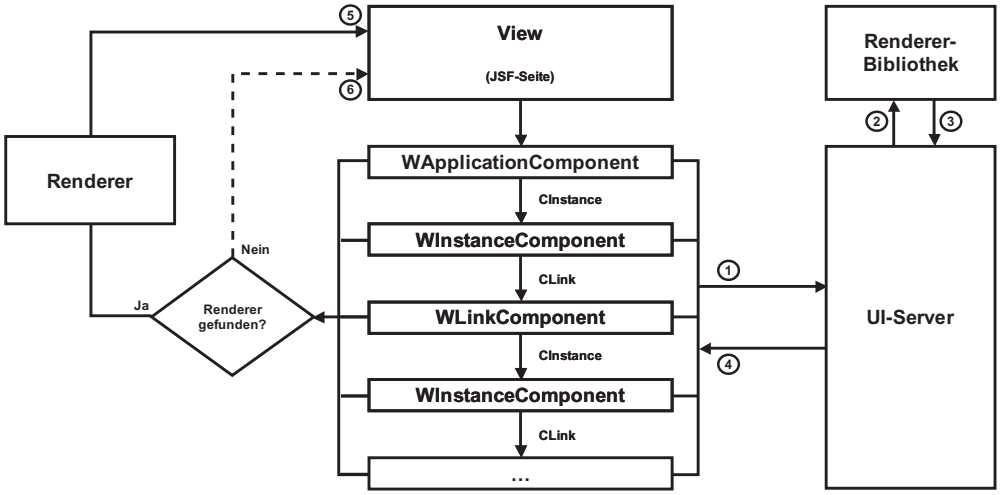


*Abbildung 5.3:* Auswahl der am besten geeigneten Visualisierung

## 5.3 Synchronisation zwischen Ontologie und Visualisierung

Die Synchronisation zwischen Ontologie und Visualisierung erfolgt mittels der Applikations-Komponente (*WApplicationComponent*) und dem Ontologie-Wurzelelement (*CModel*). Die Klasse *WApplicationComponent* implementiert dafür das Interface *javax.faces.component.ValueHolder*. Die Klassen *CModel*, *CLink* und *CInstance* stellen ihrerseits die Methode *update()* bereit.

Wie die Applikations-Komponente eine Referenz der jeweiligen Instanz der Klasse *CModel* erhält, ist in Abschnitt 5.2.1 bereits erläutert. Damit ist der Weg von der Ontologie zur Visualisierung beschrieben.

Die Synchronisation in die andere Richtung erfolgt innerhalb der Aktualisierungsphase. Im Zuge der Aktualisierungsphase (siehe Abschnitt 2.4.4, Phase *Update Model Values*) wird die Methode *update()* aufgerufen. Sie erhält das Request-Map.

In der Methode *update()* wird die Aktualisierung des gesamten Ontologie-Baumes gestartet. Dabei wird in einem Zyklus jeweils die Methode *update()* der einzelnen Ontologie-Instanzen und Ontologie-Links aufgerufen und das Request-Map an diese weitergeleitet.

Die Identifikationsbezeichnungen einzelner Ontologie-Komponenten sind gleich mit denjenigen von HTML-Elementen, welche die Inhalte der jeweiligen Ontologie-Komponenten abbilden. Im Request-Map sind Identifikationsbezeichnungen von HTML-Elementen und neue Werte für jede einzelne Ontologie-Komponente paarweise enthalten. Somit finden die Ontologie-Komponenten ihre Identifikationsbezeichnungen im Request-Map und entnehmen diesem ihren neuen Wert.

## 5.4 Synchronisation zwischen Dienstgeber und Ontologie

Der Dienstgeber ist durch die Klasse *Backend* implementiert. Die Klasse *CModel* enthält die Referenz auf eine Instanz dieser Klasse. Die Klasse *Backend* implementiert zwei Methoden – *init()* und *update()*.

In der Methode *init()* werden die durch den Dienstgeber bereitgestellten Daten initialisiert und mit Annotationen ausgestattet. Durch Aufruf dieser Methode im

Konstruktor der Klasse *CModel* werden diese Daten in die Ontologie eingebunden.

Die Methode *update()* wird in der Aktualisierungsphase von einer Ontologie-Komponente aufgerufen. Dabei wird die Referenz der Instanz der entsprechenden Ontologie-Komponente übergeben. Im Zuge dessen wird der Dienstgeber über eine notwendige Aktualisierung der Inhalte dieser Ontologie-Komponente innerhalb seiner eigenen Datenhaltung unterrichtet.

In der Methode *update()* soll der jeweilige Entwickler die für diese Aktualisierung notwendige Funktionalität implementieren. Die Methode *update()* liefert an die entsprechende Ontologie-Komponente einen textuellen Wert zurück:

- Wurde die Aktualisierung erfolgreich durchgeführt, gibt sie einen leeren String zurück
- Im Falle eines Fehlers erhält die Ontologie-Komponente von der Methode *update()* eine exakte Fehlerbeschreibung. Daraufhin kann die Ontologie-Komponente ihre Aktualisierung zurücksetzen und die gelieferte Fehlerbeschreibung speichern. Diese wird bei der Visualisierung dem Anwender angezeigt.

## 5.5 Flexibilität

### 5.5.1 Verwendung von Stylesheets

Die in Abschnitt 4.7.1 beschriebene Möglichkeit für den Einsatz von Cascading Style Sheets, ist in der Klasse *CNamedElement* implementiert. Dafür stellt die Klasse *CNamedElement* neben den Annotationen eine Variable bereit. Diese Variable kann von allen Ontologie-Komponenten verwendet werden, um den Namen einer Stylesheets-Klasse zu setzen. Wird dieser Name gesetzt, so wird er bei der Visualisierung dem jeweiligen HTML-Element zugeordnet, das die Inhalte der entsprechenden Ontologie-Komponente abbildet.

### 5.5.2 Benutzerdefinierte Auswahl einer Visualisierungsmöglichkeit

Die in Abschnitt 4.7.2 beschriebene Möglichkeit für die benutzerdefinierte Auswahl einer Visualisierungsmöglichkeit wird ebenfalls in der Klasse *CNamedElement* implementiert. Dafür stellt die Klasse *CNamedElement* eine Variable bereit,

die von allen Ontologie-Komponenten verwendet werden kann, um den Namen eines bestimmten Renderer zu setzen. Wird dieser Name gesetzt, so übergibt die Visualisierungs-Grundkomponente die Darstellung der jeweiligen Ontologie-Komponente an den entsprechenden Renderer. Der UI-Server wird dabei zur Auswahl eines am besten geeigneten Renderer nicht aufgefordert.

# 6 Evaluation

## 6.1 Fähigkeiten des Prototyps

Zum Prüfen des Konzepts wurde ein Prototyp erstellt. Der Prototyp wurde anhand des Anwendungsbeispiels “Benutzer- und Gruppenverwaltung“ evaluiert.

Mit dem Prototyp konnten verschiedene Visualisierungen des Beispieles “Benutzer- und Gruppenverwaltung“ automatisch generiert werden. Diese basieren auf den Annotationen sowie auf den Zugriffmöglichkeiten des UI-Servers auf die implementierten Renderer. Die genauen Definitionen der hier verwendeten Annotationen sind in Abschnitt 4.2 aufgelistet.

Der Ontologie-Baum der Beispielanwendung ist in der Abbildung 6.1 dargestellt.

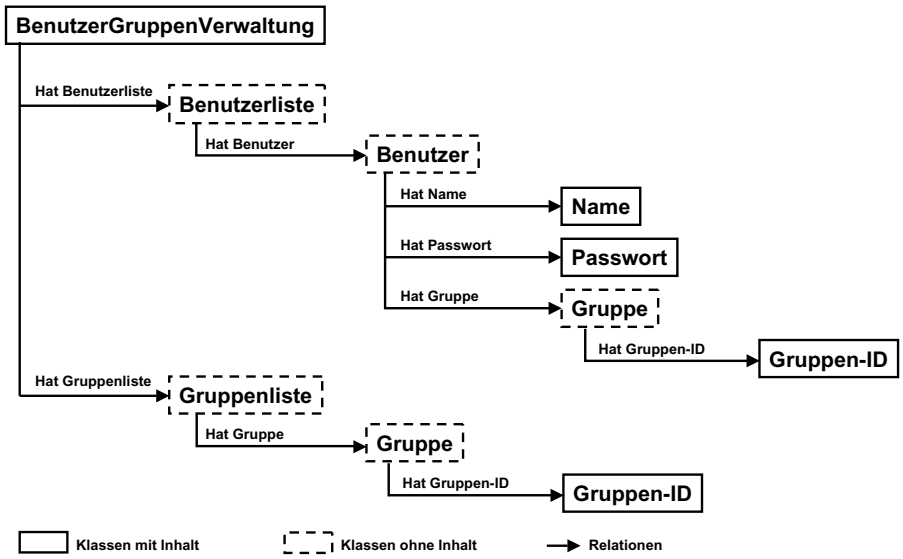


*Abbildung 6.1:* Klassen-Baum des Beispiels “Benutzer- und Gruppenverwaltung“

### Erste Visualisierung

Hier wurden noch keine Renderer umgesetzt. Allein die Grundkomponenten erzeugen die Visualisierung unter Verwendung der Annotationen 4, 5, 6 und 9. Die Standardwerte wurden folgendermaßen angepasst:

Für die Klassen “Name“, “Passwort“ und “Gruppen-ID“ wurde eingestellt, dass die Werte ihrer Instanzen geändert werden dürfen. Für die Klasse “Benutzer“ und “Gruppen“ wurde eingestellt, dass ihre bereits angelegten Instanzen gelöscht werden und neu angelegt werden dürfen. Für die Relation “Benutzer hat Gruppe“ wurde festgelegt, dass sie eine Auswahlmöglichkeit darstellt.

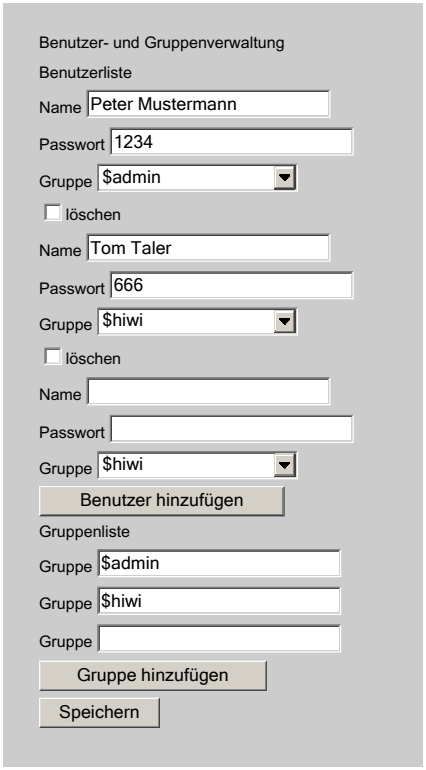


*Abbildung 6.2:* Benutzer- und Gruppenverwaltung

Ergebnis: Die Visualisierung wurde durch die Grundkomponenten selbst erzeugt (Siehe Abbildung 6.2).

### Zweite Visualisierung

Im nächsten Schritt wurde der “Tabellen“-Renderer dem UI-Server bekannt gemacht. Dieser Renderer interpretiert zusätzlich die Annotation 10. Der Standard “von links nach rechts“ blieb bestehen. Für die Relation “ Benutzerliste hat Benutzer“ wurde der Standard “von links nach rechts“ auf “von oben nach unten“ umgestellt.


*Abbildung 6.3:* Benutzer- und Gruppenverwaltung als Tabelle

Ergebnis: Die Visualisierung wurde vom “Tabellen“-Renderer vorgenommen (Siehe Abbildung 6.3).

### Dritte Visualisierung

Im Dritten Schritt wurde ein “Kartei“-Renderer UI-Server zusätzlich bekannt gemacht. Im Falle einer geringeren Anzahl an Instanzen erster Ebene (kleiner als 7) weist der “Kartei“-Renderer einen höheren Qualitätskoeffizienten auf. Daher wurde er durch das System für die Visualisierung ausgewählt.

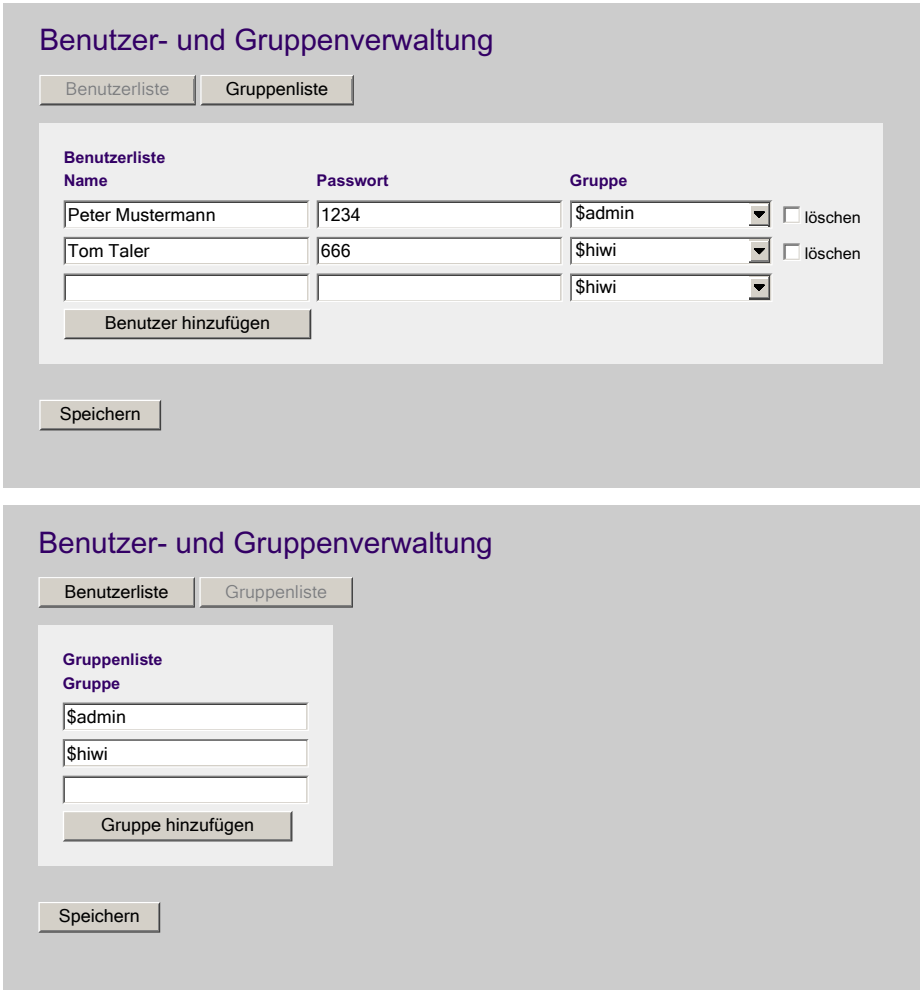


*Abbildung 6.4:* Benutzer- und Gruppenverwaltung als Kartei

Ergebnis: Die Visualisierung wurde vom “Kartei“-Renderer vorgenommen (Siehe Abbildung 6.4).

## 6.2 Gegenüberstellung der Entwicklungsprozesse

| Entwicklung ohne WebWidgets-System: | Entwicklung mit WebWidgets-System: |
|---|---|
| 1. Voruntersuchung durchführen<br>2. Daten analysieren<br>3. Konzept erstellen<br>4. Gesamtlayout entwickeln<br>5. Designkomponenten anfertigen<br>6. Inhalte erstellen und anpassen<br>7. Komponenten zusammenfügen<br>8. Anwendung testen und korrigieren<br>9. Datenfluss zum Dienstgeber muss gewährleistet werden | 1. Daten analysieren<br>2. Ontologie erstellen<br>3. Ergebnis sichten und ergänzen<br>4. Datenfluss zum Dienstgeber muss gewährleistet werden |
| Aktualisierungen erfordern zum Teil aufwändige Eingriffe in die bestehende Implementierung | Erweiterungen können jederzeit hinzugefügt werden |
| Ergebnis – eine manuell erstellte Visualisierung, eventuell in hoher Qualität | Ergebnis – eine automatisch erstellte Visualisierung in gewährleisteter hoher Qualität |

## 6.3 Vorteile

- Der Entwicklungsaufwand ist vergleichsweise gering und reduziert sich weiter nach kurzer Einarbeitungszeit

- Das Konzept sieht vor, dass eine Visualisierung entsprechend der Qualitätskriterien realisiert wird. Dadurch sind Entwickler, die das System erweitern, angehalten, sich mit der Wichtigkeit von Qualitätskriterien auseinanderzusetzen. Somit sollte Qualität gewährleistet sein

- Durch die von den Annotationen in das System eingebundene Liste mit Qualitätskriterien werden dem Entwickler zum Teil neue Aspekte in Bezug auf seine Entwicklung mitgeteilt

- Muss der Entwickler bestimmte zweckgebundene Komponenten eigens realisieren, stehen diese auch allen nachfolgenden Entwicklungen für eine erneute Verwendung zur Verfügung
- Verschiedene unabhängige Entwickler bilden durch die Anwendung des Systems eine Synergie. Sie profitieren voneinander durch die gemeinsame Nutzung des Systems

# 7 Zusammenfassung und Ausblick

Die Motivation dieser Arbeit ist es, einen Lösungsansatz für folgende Problematiken zu finden:

- Die Qualität der Web-Oberflächen wird meist nicht gebührend berücksichtigt
- Es gibt Ansätze für die Qualitätskriterien der Web-Anwendungen, aber keine Standards
- Die Entwicklung einer Web-Anwendung ist derzeit sehr aufwändig
- Die innerhalb eines Teams entstehenden Missverständnisse führen oft zu hohen Entwicklungskosten. Wird an Entwicklungskosten gespart, führt dies unter Umständen zu einer fragwürdigen Benutzbarkeit der Anwendung
- Es gibt nur wenige Werkzeuge, die das Erstellen einer Web-Anwendung im gewünschten Maße erleichtern würden. Die Entwicklung qualitativ hochwertiger Anwendungen ist mit hohem Zeit- und Kostenaufwand verbunden
- Es fehlt an fertigen Lösungen wiederkehrender Aufgaben

Das Ziel dieser Arbeit bestand darin, ein System zu entwickeln, das ausgehend von einer Ontologie automatisch eine qualitativ hochwertige Web-Anwendungsschnittstelle generiert. Die Ontologie soll dabei gleichermaßen Daten und semantische Informationen über die Daten enthalten. Dem Entwickler sollen bei der Realisierung von Web-Anwendungen Arbeitsschritte erleichtert bzw. ganz abgenommen werden. Dies geschieht durch die automatische Generierung von Anwendungsschnittstellen.

Die automatisch erzeugte Web-Anwendungsschnittstelle soll nicht nur “brauchbar“ sondern “gut benutzbar“ sein. Der hohe Aufwand, der mit der Vielfältigkeit und Komplexität der Schnittstelle Internet und Browser zusammenhängt, soll

verringert werden.

Es wurden verschiedene Qualitätsaspekte in Bezug auf Web-Anwendungen gesichtet. Darauf hin wurde eine Liste mit klar definierten Qualitätskriterien zusammengestellt. Diese Liste wurde durch konzeptuelle Überlegungen gegenübergestellt. Es wurde erörtert, wo und wie einzelne Qualitätskriterien unter dem Aspekt der automatischen Erstellung einer Web-Anwendungsschnittstelle angewendet werden können.

Das System hat die Aufgabe, Web-Oberflächen automatisch zu generieren (Siehe Abbildung 7.1). Wichtigster Aspekt ist die Einbeziehung von Qualitätskriterien. Der Ausgangspunkt für das Konzept ist eine Ontologie (hier: semantisch beschriebenes Datenmodell), welche ein einheitliches Schema für die Daten bereitstellt. Die Ontologie erlaubt das Beschreiben dieser Daten mittels Zusatzinformationen (Annotationen). Die automatisch generierte Schnittstelle muss Daten visualisieren, Interaktionen im Kontext darstellen, Eingaben und Änderungen speichern und stets auf dem aktuellen Stand sein. Er bildet eine Schnittstelle zwischen der Ontologie des Systems und dem Dienstgeber. Dessen Daten werden vom Entwickler in die Ontologie eingefügt und können nach einer Qualitätsanalyse automatisch in einer bestimmten Qualität visualisiert werden. Durch die Erweiterungen kann der Entwickler die Zweckmäßigkeit der Visualisierung der Daten erhöhen.

Das Konzept hat sich anhand des implementierten Prototyps als funktionstüchtig erwiesen. Die wichtigsten Konzeptansätze wurden umgesetzt. Dieser Prototyp kann die Daten aus der Ontologie visualisieren. Die Liste der Annotationen wurde in begrenztem Umfang erstellt. Anhand dieser geringen Anzahl von Annotationen konnte die Funktionsweise hinreichend geprüft werden. Die durch den Anwender durchgeführte Eingaben bzw. Änderungen wurden in die Ontologie zurückgeleitet. Die Weiterleitung der Informationen an den Dienstgeber konnte ebenfalls umgesetzt werden. Der Mechanismus für die durch den Dienstgeber möglichen Rückmeldungen an die Ontologie wurde implementiert und anhand von Simulationen getestet. Die Weiterleitung dieser Rückmeldungen von der Ontologie zur Visualisierung wurde ebenfalls umgesetzt.

Durch den Einsatz des Systems wird dem Entwickler die mühsame und fehlerbehaftete Zusammenstellung der Visualisierung erspart. Er muss sich lediglich mit der Ontologie auseinandersetzen und deren Schnittstelle zum jeweiligen Dienstgeber verkörpern. Die Ontologie ist transparent, gut verständlich und einfach

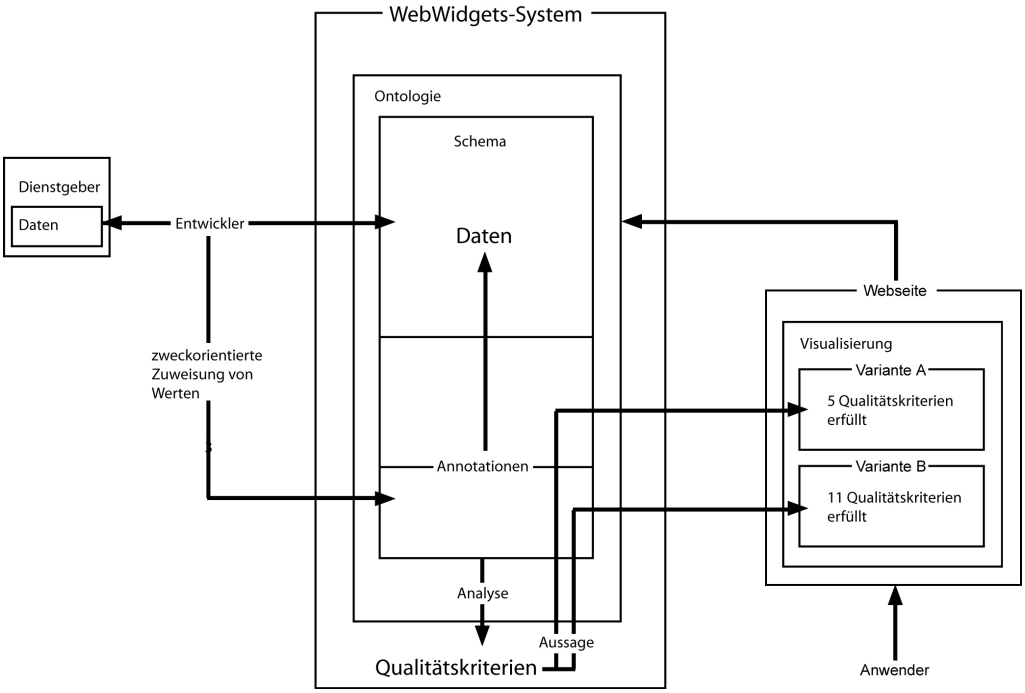


*Abbildung 7.1:* Architektur des Gesamtsystems

anzuwenden: Die funktionalen Vorteile der Ontologie sind in vollem Umfang gewährleistet. Dadurch wird dem Entwickler eine deutliche Erleichterung der Handhabung ermöglicht. Sollte sich dieser Ansatz durchsetzen, so würden Bedienung und Funktionalität von Web-Anwendungen einheitlicher und dadurch wesentlich benutzungsfreundlicher. Der mit dem Erstellen einer Ontologie verbundene Aufwand könnte minimiert werden, in dem auch dieser Erstellungsprozess automatisiert wird. Versuche dazu existieren bereits – [VOSS03]. Dem Anwender könnten mehrere gleichwertige Visualisierungen in Form einer Liste zur Auswahl angeboten werden. Er Anwender kann dadurch aus verschiedenen Möglichkeiten diejenige auswählen, die am ehesten seinen Bedürfnissen entspricht. Auf Grund der Liste von Qualitätskriterien wäre ein automatisches Analysetool denkbar, das anhand dieser Liste die Qualität bereits bestehender Web-Anwendungen beurteilen kann. Daraus resultierend könnten automatisch erzeugte Verbesserungsvorschläge angebracht werden.

# Literaturverzeichnis


[Bos04] Andy Bosch. *Java Server Faces*. Addison-Wesley, 2004.

[Dyn05] August 2005. SAP AG, Online: http://help.sap.com/.

[Jsf05] August 2005. Sun Microsystems, Inc., The JavaServer Faces Technology Tutorial, Online: http://java.sun.com/j2ee/javaserverfaces/.

[Kru02] Steve Krug. *Don't make me think*. Verl. Moderne Industrie, 2002.

[Nie00] Jakob Nielsen. *Jakob Nielsen's Web Design - Erfolg des Einfachen*. Markt + Technik Verl., 2000.

[PBB+02] Alexander Pokahr, Lars Braubach, Andreas Bartelt, Daniel Moldt, and Winfried Lamersdorf. Vesuf, eine modellbasierte user interface entwicklungsumgebung für das ubiquitous computing, 2002.

[Sal05] August 2005. CSS GmbH, Vitalisierendes Controlling, Online: http://www.css.de/index.php?id=348.

[Shn98] Ben Shneiderman. *Designing the user interface*. Addison-Wesley, 3, ed. edition, 1998.

[Thi01] Frank Thissen. *Screen-Design-Handbuch*. Springer, 2., überarb. und erw. aufl. edition, 2001.

[UG96] Mike Uschold and Michael Grüninger. Ontologies: principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996. Online: http://citeseer.ist.psu.edu/uschold96ontologie.html.

[VOSS03] Raphael Volz, Daniel Oberle, Steffen Staab, and Rudi Studer. Ontolift prototype, 2003. Online: http://wonderweb.semanticweb.org/.

[wik05] August 2005. Online: http://de.wikipedia.org/.

# Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, den 31. August 2005