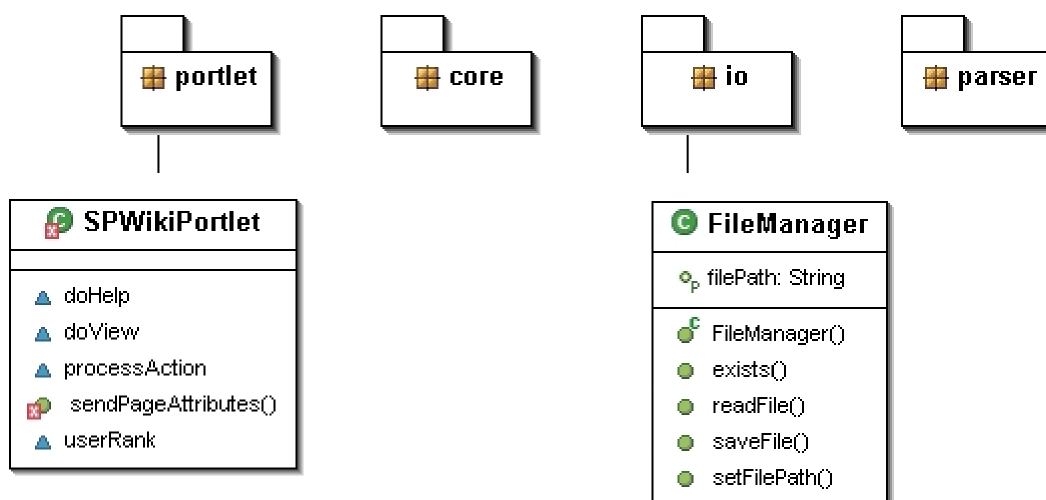
Entwurfsbeschreibung zum SmartPortalWiki

Inhaltsverzeichnis

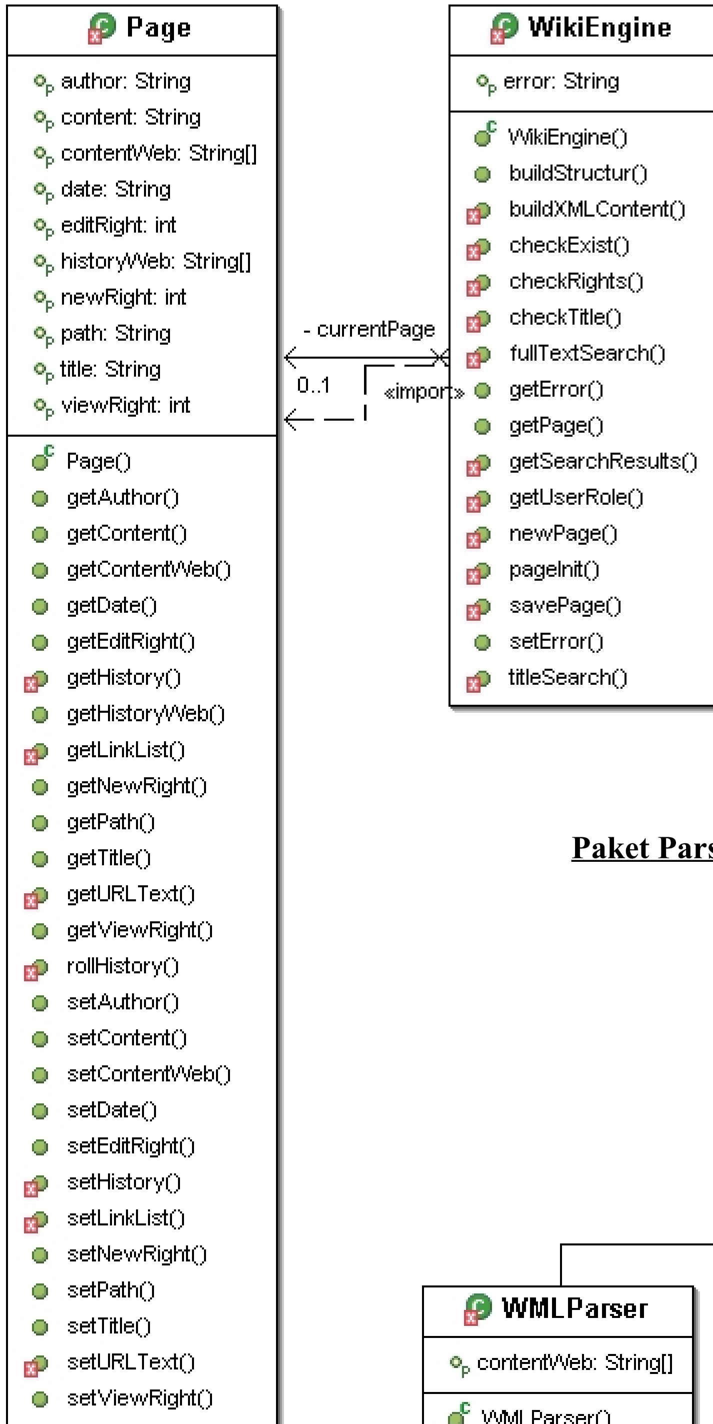
Allgemeines
Zu entwickeln ist ein Wiki-System nach Portletstandard JSR168. Es soll zum Informationsaustausch und der Kommunikation zwischen Studenten, Dozenten und Korrektoren an einem Lehrstuhl einer Fakultät dienen.

Produktübersicht
Das Produkt erfüllt entsprechend obiger Definition natürlich die Portlet-Spezifikationen. Als wichtigste Funktionalitäten sind das Ansehen von Pages, das Erstellen neuer Pages sowie das Editieren vorhandener zu nennen. Außerdem wird Wert gelegt auf die Integration eines im elate-Portal integrierten Rollenmanagements.

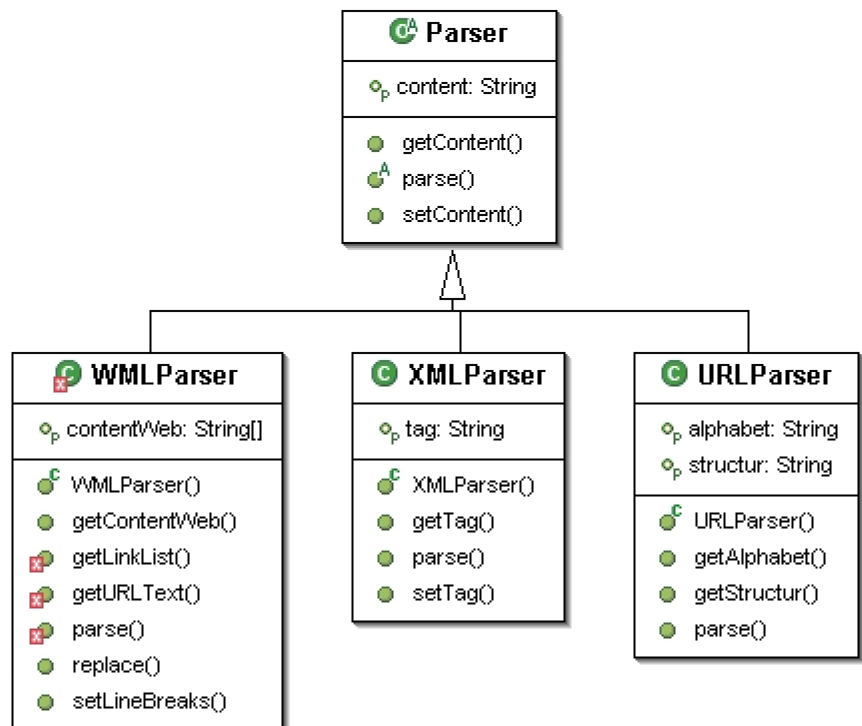
Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem
Bei der Anforderungsanalyse konnten wir 4 relativ unabhängige Funktionseinheiten bestimmen. Daraus haben wir folglich auch die 4 Grundpakete deriviert. Zum Einen sind dies die verschiedenen Parserklassen, die aus verschiedenen Content-Eingaben String-Ausgaben in dem von uns gewünschten Format generieren.
Um Lese- und Schreiboperationen auf das Dateisystem durchzuführen damit geänderte oder erstellte Inhalte angelegt werden können, wird ein entsprechendes Paket benötigt, das gewünschte Funktionalitäten implementiert.
Innerhalb des Paketes core befindet sich die wichtigste Verbindung zwischen diesen bisher relativ unabhängigen Paketen. Alle werden nämlich alle benötigt, um die im core enthaltene Page als Repräsentation unserer Programmlogik innerhalb der WikiEngine zu generieren. Ausserdem ist hier auch die Suchfunktionalität eingefügt.
Das Paket portlet schließlich enthält unsere eigentliche Portletimplementierung, die die Funktionalität unseres Produktes in die standardisierte Portlet-Struktur mit ihren vorgegebenen Methoden einbindet.
Für die Darstellung unserer Inhalte haben wir uns gegen die Verwendung von Velocity und stattdessen für JavaServerPages entschieden. Unsere Vorbehalte gegenüber Velocity sind v. A. darin begründet, dass Velocity standardmässig nicht in Jetspeed implementiert ist. Innerhalb des elate Portals existiert zwar eine entsprechende Anbindung, im Hinblick auf die Portierbarkeit wäre es jedoch nicht ratsam, diese in unserem Projekt zu verwenden.
Innerhalb des Produktes sind bis dato 4 verschiedene JSP`s als Templates zur Darstellung geplant. Im Einzelnen sind dies:
View, Edit, New Page, History.



Paket Core:



Paket Parser:



Grundsätzliche Struktur- und Entwurfsprinzipien für die einzelnen Pakete

Als 1.Punkt wird hier unser Klassendiagramm vorangestellt, Inhalte und Methoden werden in der darauf folgenden Klassenbeschreibung spezifiziert.

Pakete/Klassen:

spwiki.parser

- XMLParser

  Der Parser liest den Content eines bestimmten Tags einer XML-Datei aus und liefert diesen als String zurück

  Begründung:

  Der XMLParser ist unabdingbar, da die Speicherung in XML explizit gefordert ist und somit auch das Lesen und Parsen entsprechender Datenbestände eine Grundfunktionalität des Produktes darstellt. Aufgrund des Umfangs und der mangelnden Beziehungen zu anderen Funktionen empfiehlt es sich, diesen Parser als Klasse zu realisieren. Er wird als Instanz innerhalb der Engine integriert.

- WMLParser

  Der WikiMarkupLanguage Parser wird auf eingegebene Strings angewandt und übersetzt diese in HTML Text mit entsprechenden Formatierungen, welche dann später in der entsprechenden .jsp Datei enthalten sind. Insbesondere HTML Tags sowie die Zeichen "<" und ">" sollen durch entsprechende Symbole ohne syntaktische Doppeldeutigkeit ersetzt werden. Effektiv sollen dann nur die entsprechenden Elemente der WikiMarkupLanguage in die entsprechenden HTML Tags übersetzt werden.

  Begründung:

  Der WikiMarkUpLanguage Parser stellt genau wie die anderen Parser einen eindeutig abgrenzbaren Anwendungsbereich dar, der innerhalb der Klasse Engine als eigene Funktionseinheit existieren sollte.

- URLParser

  Der URLParser ermittelt absolute Pfade aus relativen Pfaden unserer Struktur

  Begründung:

  Der URLParser ist unabhängig von anderen Klassen (außer der generischen Parser Klasse) und stellt Funktionalitäten bereit, die immer gebraucht werden, jedoch unabhängig von Pages oder der Pagerepräsentation zu sehen sind. Ausserdem stellt er relativ komplexe Funktionen bereit. Deshalb empfiehlt sich die Implementierung als eigene Klasse.

- Parser (abstract)

  Begründung:

  In unserem Projekt werden verschiedene Parser verwendet. Alle Parser erben von der abstrakten Klasse, da alle Parser ähnliche Funktionalitäten besitzen (aus Content-Eingaben Strings mit ausgelesenen Informationen erstellen), diese jedoch unterschiedlich realisieren.

spwiki.io

- FileManager

  FileManager realisiert alle Funktionen zur Bearbeitung aller Dateien. Durch die Art unserer Datenhaltung hängt diese Klasse sehr eng mit dem XMLParser zusammen, genauer existiert er sogar nur als Instanz in jenem.

  Begründung:

Der FileManager wird in unterschiedlichem Kontext als Instanz benötigt, eine Modellierug als Klasse ist folglich unumgänglich

spwiki.core

Page

Zur Präsentation einer Seite oder auch zur Darstellung von Fehlermeldungen wird ein Page Objekt erstellt. Alle Lese- und Schreiboperationen auf eine Entität werden auf diesem Objekt durchgeführt. Fehlermeldungen (wenn beispielsweise ein angefragter Link nicht existiert) werden auch als Page Objekt realisiert, dessen Inhalt jedoch nicht aus einer XML Datei ausgelesen wird.

Begründung:

Um ein realitätsnahes und lesbares Klassenmodell zu erhalten empfiehlt sich eine Einbindung als eigene Klasse. Außerdem kann so deutlicher zwischen Pages als Datenhaltungs- und Strukturobjekten und Pagemanipulationen als Methoden auf Pages unterschieden werden. Dies sollte die Klarheit des Entwurfs stärken.

WikiEngine

Die Wiki Engine stellt den Kern unseres Modells dar, in ihr laufen alle wichtgen Prozesse ab, sie bildet den Container für unsere Programmlogik. Sie enthält alle Parser als Klassen, ausserdem die aktuelle Page als Objekt.

Alle auf Pages angewendeten Methoden befinden sich hier und auch die im Pflichtenheft umrissenen Geschäftsprozesse sind als integrierte Methoden enthalten.

Begründung:

Um effektiv neue Pages erstellen zu können und eine komfortable Arbeit mit diesen Objekten zu ermöglichen dient die WikiEngine als Container in dem alle Handlungen direkt oder indirekt ausgeführt werden.

spwiki.portlet

SPWikiPortlet

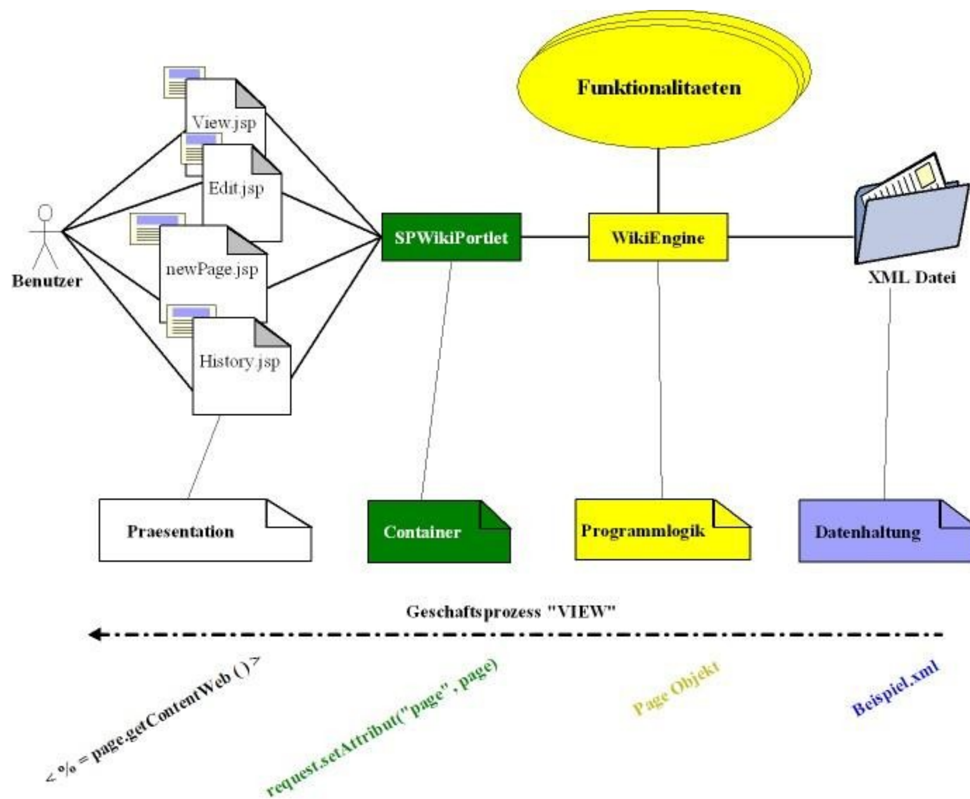
SPWikiPortlet enthält als Attribut die Engine über welche alle Darstellungen nach Außen hin (unter Verwendung der unterschiedlichen Parser) realisiert werden. Übergeben werden diese zur Darstellung innerhalb der verschiedenen JSP's durch die doView() Methode.

Eine geänderte Darstellung setzt geänderte Inhalte voraus. Diese Änderungen innerhalb der Daten/des Modells werden durch die processAction() Methode verarbeitet.

Begründung:

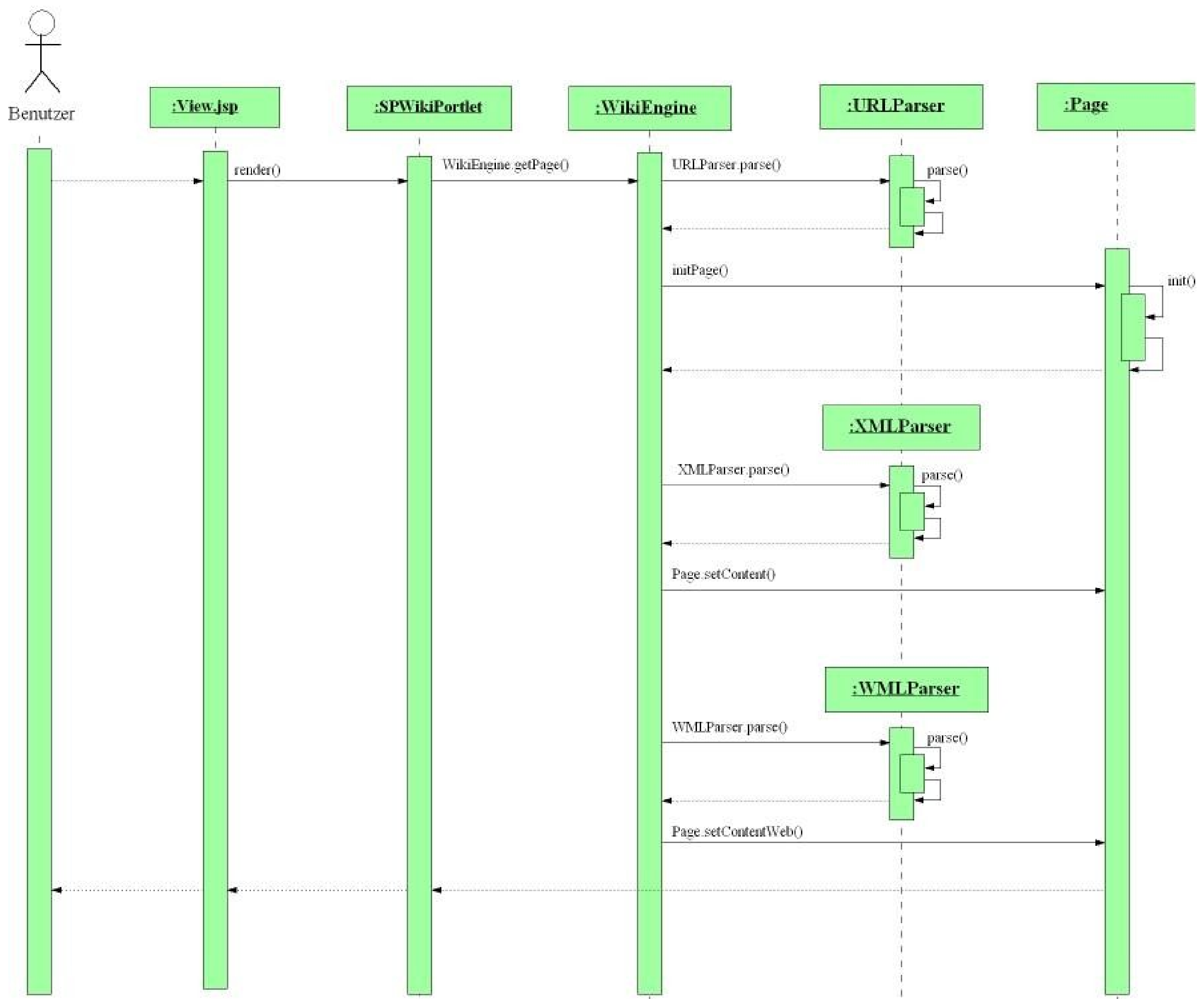
Dass die Implementierung eines Portlets natürlich auch eine Klasse entsprechenden Aufbaus enthalten muss, ist selbstverständlich. Sie bindet die vom Portletstandard vorgegebenen Methoden ein.

Um diese statische Beschreibung unseres Projektes abzuschliessen und um die Abhängigkeit unserer dynamischen Modelle von eben jener Struktur zu verdeutlichen, haben wir entsprechende Zusammenhänge in der folgenden Übersicht dargestellt.



Erklärungen zur Funktionsweise dieser Struktur:
Bereits weiter oben haben wir auf den Zusammenhang von dynamischen und statischem Modell hingewiesen. Verdeutlicht haben wir diese Beziehung durch Darstellung der an dem Geschäftsprozess "View Page" beteiligten Komponenten. Wir wollen diese Übersicht über den entsprechenden Prozess verfeinern und vertiefen, indem wir jenen nochmals als Sequenzdiagramm darstellen.

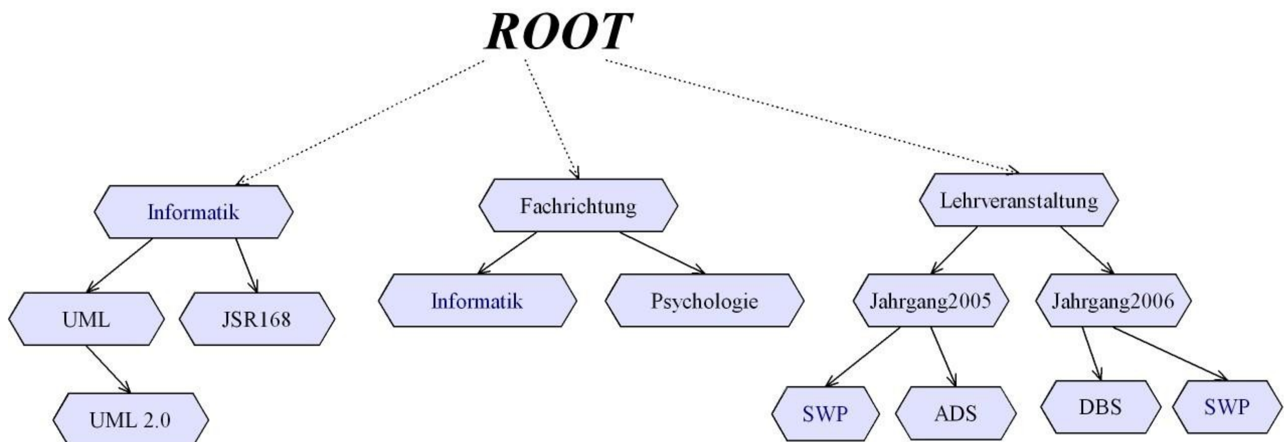
Sequenz Diagramm: Page View



Auch die anderen Geschäftsprozesse "Edit" und "New Page" verwenden nur leicht abgeänderte Versionen mit ähnlichem Ablauf. Auf die grafische Ausführung dieser haben wir somit verständlicherweise verzichtet.
Eine beliebige Seite wird zur Ansicht vom Benutzer ausgewählt. Die View.jsp sendet also eine renderRequest() die im Portlet als Parameter übergeben wird. Aus dieser Request kann eine eindeutige Zuordnung zu einer Page innerhalb unserer Datenhaltung abgeleitet werden. Die methode getPage() der Engine wird aufgerufen um die entsprechende Page aus der Datenhaltung abzurufen. Die entsprechenden Parser werden nun auf den Quelltext der gefundenen Page angewandt, um diese für die Darstellung als HTML innerhalb der JSP aufzubereiten. Schließlich wird die formatierte Page als Page Objekt zurückgegeben um deren Anzeige zu realisieren.

Der letzte interessante Punkt auf den wir hier eingehen wollen ist die Form unserer Datenhaltung. Pages sollen als Baumstruktur gespeichert werden. Das heißt also, daß man aus dem Pfad und Namen (die nach unserer Planug eine Page eindeutig identifizieren – Vgl. Dateisystem) einer gespeicherten Seite Strukturinformationen gewinnen kann. Wir können daraus die Position der Seite innerhalb des Baumes eindeutig erkennen. Zum Speichern und Laden dieser Seiten wird der FileManager verwendet. Speichern einer Seite bedeutet, daß entweder eine Seite editiert wurde (der Speicherpfad ist eindeutig vorbestimmt) oder aber eine neue erstellt wurde. Falls der 2. Fall eintritt (New Page) dann entscheidet die aktuelle Position innerhalb des Baumes – also die Seite, von der aus New Page aufgerufen wurde– über den Speicherpfad. Eine neue Seite wird immer als

Unterpunkt der aktuellen erstellt. Damit lässt sich jede mögliche Verzweigung des Baumes realisieren. Dabei wird jeder neue Ebene der Verzweigung als neuer Ordner im Dateisystem abgebildet. Zur Veranschaulichung des Sachverhaltes haben wir eine solche Baumstruktur als Beispiel grafisch dargestellt.



Besonders hinzuweisen ist an dieser Stelle auch noch darauf, dass der Name einer Page eben nicht hinreichend ist, um sie eindeutig zu identifizieren. So existieren im Beispiel 2 Informatik und 2 SWT Pages, die jedoch nicht gleiche Entitäten bezeichnen. Eine Entität innerhalb der Struktur wird erst eindeutig durch den Pagenamen in Verbindung mit dem Pagepfad bestimmt.