Studienarbeit “WikiDoc”

Sommersemester 2006

WikiDoc

Robert Wolf

rwolf@inf.fu-berlin.de

Betreuer: Christopher Oezbek

16. Oktober 2006

Zusammenfassung

In dieser Arbeit wurde das Dokumentationwerkzeug JavaDoc um Wiki-Funktionalitäten erweitert. Das Ziel ist es, verteilten Entwicklerteams eine bessere Möglichkeit zur Bearbeitung der Dokumentation in die Hand zu geben. Die Barriere für die Bereitstellung von Dokumentation soll gesenkt werden. Besonders quelloffene Projekte wie in der Open-Source- und Free-Software-Bewegung könnten davon profitieren.

Inhaltsverzeichnis

1 Einleitung

Die Aufgabe dieser Studienarbeit besteht darin, das JavaDoc-Dokumentationswerkzeug um Wiki-Funktionalität zu erweitert. Die JavaDoc-HTML-Seiten sollen dabei von den Benutzern nicht nur gelesen, sondern auch verändert werden können. Das Ziel ist es, die Dokumentation von Programmtexten – insbesondere in Open-Source-Projekten – zu verbessern. Den Nutzern des Programmcodes soll die Möglichkeit gegeben werden, die JavaDoc-Kommentare um eigene Bemerkungen zu ergänzen oder ganz umzuschreiben und so anderen Nutzern die Arbeit mit den Java-Klasse zu erleichtern. Langfristig soll dieses Vorgehen zu qualitativ besserer Dokumentation führen. Das Wiki soll es den Entwicklern darüber hinaus erlauben, die veränderten JavaDoc-Kommentare automatisch in den eigenen Quellcode zu übernehmen, und so den Entwicklern die Dokumentationsarbeit erleichtern.

1.1 Softwaredokumentation

Softwareentwicklung ist eine schwierige und komplexe Aufgabe, welche heutzutage nur selten von einem Programmierer allein bewältigt werden kann. So arbeiten in der modernen Softwareentwicklung viele Programmiere in verschiedenen Projektteams an der Fertigstellung eines Softwareproduktes. Diese Aufteilung in einzelne Projektteams findet in der Open-Source-Softwareentwicklung ihren Höhepunkt, wo die einzelnen Entwickler häufig nicht nur räumlich getrennt sind, sondern auch in keinem persönlichen Kontakt zueinander stehen. Diese Aufteilung macht eine Dokumentation der Arbeit der einzelnen Programmierer unumgänglich, um eine sinnvolle Zusammenarbeit überhaupt erst zu ermöglichen.

Die Dokumentation der Software findet allerdings nicht nur auf der Ebene der Entwickler statt, sondern es lassen sich folgende Kategorien der Softwaredokumentation unterscheiden:

- **Technisch Dokumentation** Die technische Dokumentation wird hauptsächlich von Programmieren geschrieben und verwendet, um den Quellcode und die Funktionsweise eines Programms näher zu erläutern. Dies ist notwendig, da der Programmcode alleine zum Nachvollziehen der Funktionsweise unzureichend ist und der Quellcode eines Programms auch nicht immer zur Verfügung steht. Mithilfe der technischen Dokumentationen werden zum Beispiel Erklärungen zur Verwendung von Variablen, Erläuterungen zur Arbeitsweise eines Algorithmus oder Anweisung zum Verwenden von Funktionen bereitgestellt. Die Dokumentation ist ein wichtiges Werkzeug zum Verstehen eines Programms und nur mit ihrer Hilfe ist es sinnvoll möglich Programmkomponenten oder Bibliotheken von anderen Entwicklern zu verwenden. [Law06]

- **Architektur- und Design-Dokumentation:** Architektur- und Design-Dokumentation dient in erster Linie dazu, sich einen Überblick über die Anwendung zu ver-

schaffen und die Arbeitsweise besser zu verstehen. Der Fokus liegt dabei nicht wie bei der technischen Dokumentation auf der konkreten Implementierung, sondern auf der Analyse des gegebenen Problems. Oft wird erst durch die Dokumentation der Architektur einsichtig, warum eine Anwendung so umgesetzt wurde, wie sie in der technischen Dokumentation beschrieben wurde.

- **Endbenutzer-Dokumentation** Diese Art der Dokumentation richtet sich direkt an die Benutzer der Software und soll ihnen die Arbeit mit der Software ermöglichen. Dabei werden die notwendigen Arbeitsschritte und Einstellungen des Programms erklärt, nicht jedoch der technische Hintergrund. Beispielsweise werden Handbücher, Anleitungen und Hilfeseiten im Programm zu dieser Art der Dokumentation gezählt.

- **Werbung und Vermarktung** Bei dieser Art der Dokumentation steht die Werbung für die Software im Vordergrund. Es wird versucht die Vorzüge der Anwendung herauszustellen und beim Endkunden Kaufinteresse zu erwecken. Die Dokumentation der Arbeitsabläufe und des Aufbaus des Programms erfolgt dabei nur, insofern sie für die Bewerbung des Produktes erforderlich ist.

1.2 JavaDoc

JavaDoc ist ein Software-Dokumentationswerkzeug zur Erzeugung einer Beschreibung der API[1] eines Programms. Es hilft somit beim Erzeugen einer technischen Dokumentation. JavaDoc ist in der Lage spezielle, direkt im Programmcode geschriebene Kommentare auszuwerten und so die Dokumentation der API zu erzeugen. Mithilfe verschiedener Doclets können verschiedene Ausgabeformate erstellt werden. So kann JavaDoc die Dokumentationen als HTML-Seiten oder als PDF-Dokument erzeugen. Mit entsprechenden Doclets sind auch andere Ausgabeformate möglich. JavaDoc wird mit dem Java Development Kit[2] installiert und wird von Sun selbst zur Dokumentation der Java-API genutzt. Aufgrund dieser Tatsachen hat sich JavaDoc als Standard-Dokumentationswerkzeug der Programmiersprache Java etablieren. Durch die Lokalität von Code und Daten ist eine mit JavaDoc-Annotationen versehende Quellcodedatei für den Entwickler außerdem auch ohne die erzeugte JavaDoc-Ausgabe besser zu verstehen und folgt somit den von D. E. Knuth postulierten Ansatz des "Literate Programming". [Knu84]

1.3 Wiki

Wikis bezeichnen Webseiten, die von den Benutzern nicht nur gelesen, sondern auch online verändert werden können. Alle Veränderungen werden dabei vom System gespeichert und können bei Bedarf wiederhergestellt werden. Nach diesem Prinzip funk-

[1] Application Programming Interface, Programmschnittstelle
[2] http://java.sun.com/javase/

tioniert auch das wohl bekannteste Online-Lexikon Wikipedia[3]. Das Lexikon entwickelte sich in kurzer Zeit zu den am meisten genutzten Internet-Seiten. Das Expertenwissen der vielen Teilnehmer ermöglicht es hochwertige Artikel zur Verfügung zu stellen, die zum Teil mit denen etablierter Lexika konkurrieren können. [Gil05]

1.4 WikiDoc

WikiDoc soll die Vorzüge eines Wikis mit denen des Dokumentationswerkzeuges JavaDoc verbinden. Hintergrund dieser Idee ist die Erkenntnis, dass die Dokumentation von Softwareprojekten häufig als unzureichend zu bewerten ist. Durch die Möglichkeit die JavaDoc-Kommentare durch Benutzer verändern zu lassen, soll die Dokumentation der Softwareprojekte verbessert werden.

Der Kreis derjenigen, die an der Dokumentation der Software beteiligt sind, vergrößert sich. Aufgrund dessen wird einerseits die Dokumentation nicht ausschließlich durch den Programmierer geschrieben, so dass neben der technischen Sicht auch eine anwendungsorientierte Sicht in die Dokumentation mit einfließen kann. Andererseits besteht die Hoffnung das eine große Anzahl von Beteiligten zu einer besseren Qualität der Dokumentation führt. Das Beispiel Wikipedia zeigt wie qualitativ hochwertige Texte durch die Mitarbeit viele Benutzer entstehen können. Oder wie es Eric S. Raymond formulierte: „Given enough eyeballs, all bugs are shallow." [Ray99]

1.5 Anforderungen

Folgende Anforderungen werden an das Programm gestellt:

- **Wiki Fähigkeiten:** Die JavaDoc Kommentare sollen über ein Wiki Interface editierbar sein. Desweitern sollte es möglich sein die Änderungen wieder in den Quellcode zurückzuschreiben.
- **Laden des Quellcodes:** Das System sollte das Wiki aus den JavaDoc Kommentare der Quellcode Dateien aufbauen köpnnen. Da sich das System in normalen Entwicklungsprozess integrieren sollte, mussten das Wiki die Quellen auch aus CVS und Subversion Repositories laden könne.
- **Synchronisation des Quellcodes:** Das System sollte in der Lage sein das Wiki mit den Java Dateien abzugleichen. Dabei sollten Änderungen am Quellcode sinnvoll in das Wiki integriert werden und Änderungen am Wiki in den Quellcode zurückgeschrieben werden können.
- **JavaDoc „Look and Feel":** Das Wiki sollte sich in das gewohnte Layout der JavaDoc HTML Seiten integrieren. Die Entwickler und Benutzer sollten in der gewohnten Umgebung arbeiten können.

[3] http://wikipedia.org

- **Robustheit:** Das System sollte stabil laufen und Fehler sinnvoll behandeln könne. Besonders soll Konkurierende Zugriffe auf das
- **Benutzerverwaltung:** Gerade Änderungen am Quellcode wie das Zurückschreiben der JavaDoc Kommentare benötigen eine integrierte Benutzerverwaltung, um den Zugriff auf bestimmte Wiki Funktion zu reglementieren.
- **Logging:** Gerade in größeren Projekten sollten die Veränderungen des Wikis protokolliert und auf die Benutzer zurückführbar sein.
- **Performance:** Das System sollte Performant sein. Die Interaktion mit dem Web Interface sollte nicht spürbar langsammer als der Aufrauf einer normalen Webseite sein. Im Falle einer längeren Wartezeit sollte der Nutzer über den Fortschritt informiert werden. Dewsweiteren sollten Änderungen am Wiki für andere Nutzer sofort sichtbar sein. Außerdem sollte das System 20 gleichzeitige Lesezugriffe und fünf gleichzeitige Schreibzugriffe unterstützen. Als Zielvorgabe sollte das System in eine VM mit 128MB RAM laufen.
- **Installationsanforderungen:** Um den weiten Einsatz vor allem in OpenSource Projekten zu erleichtern, sollten keine Installationshürden aufgebaut werden. Das Programm sollte sich einfach installieren lassen und möglichst wenig andere Komponenten benötigen.
- **Optionale Anforderungen:** Unter diesem Punkt sind Ideen zusammengefasst die nicht unbedingt umgesetzt werden mussten, die aber die Arbeit mit dem Wiki vereinfacht hätten und die sich zum Teil auch in anderen Wiki Seiten bewährt haben. Ein Forum bzw. eine Mailingliste zum Beispiel könnte bei Streitigkeiten zu einem Eintrag weiterhelfen. Ein RSS-Feed könnte Nutzer über neue Beiträge und Änderungen informieren.

2 Analyse und Entwurf

2.1 Programmiersprache und Frameworks

Das Projekt ist in Java unter Verwendung der Spezifikation Java 2 Enterprise Edition(J2EE) realisiert. Die Spezifikation stellt eine gute Softwarearchitektur für die Entwicklung von Webanwendungen bereit. Die Spezifikation wird von verschiedenen Anbietern implementiert. Das Projekt setzt nur die Verwendung eines Servlet Container wie z.B. den Apache Tomcat vorraus. Durch die Verwendung von Java sind außerdem alle für das Programm notwendigen Bibliotheken unter eine Open Source Lizenz verfügbar.

2.2 Inbetriebnahme

Damit durch die Installation und Konfiguration des Programms für den Benutzer keine Hüden aufgebaut werden, nutzt WikiDoc für die Installation des Programmes nur das Deployment-Konzept der Java Enterprise Edition. Alle benötigten Bibliotheken werden dabei von dem Programm mitinstalliert, d.h. es bestehen keine zusätzlichen Abhängigkeiten. Insbesondere wird auf den Einsatz von Datenbanken verzichtet. Der Benutzer muss für die Inbetriebnahme keine Konfiguration vornehmen. Das Programm ist in der Standardkonfiguration lauffähig. Für individuelle Bedürfnisse kann das Programm dennoch über Konfigurationsdateien angepasst werden.

2.3 Datenhaltung

Um den Einsatz von Datenbanken zu vermeiden, werden die für das Wiki notwendigen Informationen direkt im Dateisystem gespeichert. Da es sich bei den JavaDoc-Kommentaren um hierarchisch strukturierte Daten handelt, sind XML-Dateien zur Datenhaltung besonders geeignet. Ferner existieren für die Verarbeitung von XML-Dateien zahlreiche Werkzeuge. WikiDoc verwendet das Werkzeug XMLBeans[4] der Apache Foundation, um auf die Informationen in den XML-Dateien in Java zuzugreifen. Dazu erzeugt XMLBeans aus einem XML-Schema Java-Klassen, die vom Zugriff auf die Informationden in den XML-Dateien abstrahieren.

2.4 Extrahieren der JavaDoc-Kommentare

Bevor die Kommentare in WikiDoc bearbeitet werden können, müssen die Struktur einer jeden Klasse analysiert und vorhandene JavaDoc-Kommentare extrahiert werden. Dabei ist es zum einen möglich, die Informationen direkt aus den Java-Quellcodedateien und zum anderen aus den JavaDoc-HTML-Seiten zu gewinnen.

[4] http://xmlbeans.apache.org

2.4.1 Extrahieren aus den Java-Quellcodedateien

Eine Java-Quellcodedatei mit allen Kommentaren kann als Syntaxbaum interpretiert werden, welcher mithilfe eines Abstract-Syntax-Tree-Parsers (AST-Parser) erstellt werden kann. Der AST-Parser arbeitet somit direkt auf den Quellen der Kommentare und stellt dem Programm auf diese Art und Weise die Informationen über Schnittstellen zur Verfügung. Um dem Benutzer das von JavaDoc gewohnte Aussehen der Dokumentation anzubieten, ist es notwendig, aus den JavaDoc-Kommentaren im Sytnax-Baum JavaDoc-HTML-Seiten zu generieren. Dies ist problematisch da zum Erstellen einer vollständigen Dokumentation neben den Beschreibungsseiten der jeweiligen Klassen viele verschiedene Übersichts- und Indexseiten erzeugt werden müssen.

Modifikationen, die durch WikiDoc vorgenommen werden, können direkt im Syntax-Baum erfolgen. Aus dem Syntax-Baum kann jederzeit wieder eine Java-Quellcodedatei erstellt werden. Damit ist es mithilfe dieses Verfahrens möglich, sowohl Kommentare aus den Dateien zu lesen, als auch wieder zurück in Java-Quellcodedateien zu schreiben.

Für das Projekt eignet sich beipielsweise der AST-Parser der Eclipse-Entwicklungsumgebung[5]. Er ist frei verfügbar, hat den Vorteil einer stabilen Codebasis und unterliegt einer ständigen Weiterentwicklung. Er ist jedoch intregraler Bestandteil von Eclipse. Im Rahmen dieser Studienarbeit wurde der Aufwand, diesen AST-Parser seperat einzusetzen, als sehr aufwändig eingeschätzt.

Weniger mächtig, aber bei jedem Java Development Kit installiert, ist das JavaDoc-Werkzeug von Sun[6]. Es handelt sich dabei um einen AST-Parser, der jedoch keinen vollständigen Syntaxbaum erzeugt, sondern nur die für die Erzeugung einer Dokumentation notwendigen Bestandteile. Aufgrund dieser Einschränkung ist es nicht möglich die modifizierten Kommentaren aus dem Wiki unmittelbar in eine Java-Quellcodedatei zu übertragen.

2.4.2 Extrahieren aus den JavaDoc-HTML-Seiten

Eine andere Möglichkeit der Informationsgewinnung ist die Nutzung der von einem JavaDoc-Werkzeug genierten HTML-Seiten. Dazu müssen die aus den JavaDoc-Kommentaren erzeugten HTML-Seiten geparst und die für das Wiki notwendigen Informationen extrahiert werden. Zusätzlich müssen die Kommentare den ursprünglichen Stellen in den HTML-Seiten zugeordnet werden, um im Wiki die Dokumentation im ursprünglichen Design aber mit den geänderte Kommentaren darzustellen. Bei diesem Verfahren entfällt das Erzeugen der HTML-Seiten, diese müssen nur punktuell modifiziert werden. Insbesondere werden alle Index- und Übersichtsseiten von dem JavaDoc-

[5] http://www.eclipse.org
[6] http://java.sun.com/j2se/javadoc/

Werkeug erzeugt und müssen nicht separat generiert werden.

Durch die zusätzliche Abstraktionschicht der HTML-Seiten vom Java-Quellcode ist das Programm in der Lage, Änderungen am Java Syntax insofern zu kompensieren, als das sich die Grundstruktur der HTML-Seiten nicht verändert hat. Bei geringen Änderungen an der Struktur bleibt die Funktionalität des Programmes im bisherigen Umfang erhalten bzw. eine fehlerhafte Interpretation der Seiten bleibt lokal begrenzt. Das Programm zeigt nicht interpretierte Bereiche der HTML-Seiten unverändert an. Daher ist dieses Verfahren relativ robust gegenüber Veränderungen. Beispielsweise hatte der Wechsel der Java-Versionen von Java 1.4 zu Java 5.0 nur geringfügige Änderungen an den HTML-Seiten zur Folge.

Ein Zurückschreiben der Kommentare in die Java-Quellcodedateien ist mit diesem Verfahren noch umständlicher als die Verwendung des JavaDoc-Werkzeuges als AST-Parser, da beispielsweise die Signaturen der Methoden in den HTML-Seiten in keiner kanonischen Form vorliegen.

Mit dem Java Development Kit wird ein Standard-Doclet geliefert, mit welchen HTML-Seiten erzeugt werden können. Die mit diesem Doclet erzeugten HTML-Seiten sind die am meisten verwendeten Java-API-Dokumentationen und somit den meisten Benutzern vertraut. Das Doclet wird mit jeder Java-Version weiterentwickelt und unterstützt daher stets den aktuellen Sprachumfang. Leider sind die erzeugten HTML-Seiten nicht wohlgeformt; sie genügen nicht der HTML-Spezifikation. Außerdem sind die Informationen in den Seiten dem Layout und nicht der Struktur nach angeordnet. Aus diesem Grund stellt es eine Schwierigkeit dar, diese Seiten zu parsen.

Neben dem Standard-Doclet exitieren eine Vielzahl weiterer frei verfügbarer Doclets, welche ebenfalls HTML-Seiten erzeugen können. Im Gegensatz zum Standarad-Doclet, welches aus lizenzrechtlichen Gründen nicht modifiziert weitergegeben werden darf, kann die Ausgabe dieser Doclets beliebig angepasst werden. Dadurch ist es möglich die HTML-Seiten mit im Browser nicht sichtbaren Markierungen zu erweitern, um Kommentare einfach extrahieren und zuordnen zu können. Leider existierten zu Begin der Studienarbeit keine frei verfügbaren Doclets, welche zum einen das Original-Layout des Standard-Doclets angemessen imitieren konnten und zum anderen den aktuellen Sprachstandard Java 5.0 beherrschten.

Neben der Verwendung verschiedener Doclets für das JavaDoc-Werkzeug exitiert die Möglichkeit andere Werkzeuge zur Erzeugung der Java-API-Dokumentation inklusive der Markierungen zu verwenden. Wie bei den freien Doclets bestehen jedoch auch hier Defizite im Bezug auf das Layout und den unterstützten Sprachumfang.

2.4.3 Fazit

Jedes der untersuchten Verfahren hat seine Stärken und Schwächen. Die AST-Parser bieten die größte Flexibilät, sind jedoch am aufwendigsten umzusetzen. Insbesondere erlaubt dieses Verfahren das Zurückschreiben in die Java-Quellcodedateien. Der Aufwand für Einarbeitung und die Erzeugung der HTML-Seiten erschien jedoch recht hoch und im Rahmen der Studienarbeit nicht umsetzbar. Das Parsen der mit dem Standard-Doclet erzeugten HTML-Seiten hingegen erlaubt die Nutzung des Orginal-Layouts sowie des aktuellen Sprachumfanges und ist trotz der schlecht strukturierten HTML-Seiten prinzipiell umsetzbar. Dies zeigte ein im Rahmen der Entwurfsanalyse entwickelter Prototyp, welcher mit wenig Aufwand bereits weite Teile der HTML-Seiten analysieren und mit Markierungen versehen konnte. Das Zurückschreiben in die Java-Quellcodedateien ist mit diesem Verfahren nicht möglich.

2.5 Annotieren der JavaDoc-HTML-Seiten

Da die Ausgabe des Standard-Doclet nicht modifiziert und mit semantischens Markierungen versehen werden kann, müssen die Informationen direkt aus den schlecht strukturierten HTML-Seiten gewonnen werden. Um nicht bei jedem Aufruf die JavaDoc-Seite erneut aufwendig interpretieren zu müssen, werden diese in einem Vorverarbeitungschritt mit semantischens Markierungen erweitert.

Bei einem modularen Aufbau ist dieser Vorverarbeitungschritt notwendigerweise nicht integraler Bestandteil des Programmes und muss daher nicht in Java realisiert sein. Die HTML Dateien entsprechen im wesentlichen XML-Dateien, sodas die Markierungen mithilfe von XML-Wekrzeugen am einfachsten zu realisieren sind. Mit der Extensible Stylesheet Language (XSL) steht eine funktionale Programmiersprache zur Transformation von XML-Dokuemten zur Verfügung. Mit XSL werden Regeln zur Überführung von XML-Dokumenten in andere Dokumente definiert. Mithilfe dieser Regeln können die von JavaDoc erzeugten HTML-Seiten in nahezu identische mit semantischen Annotationen versehene HTML-Seiten transformiert werden.

Da jedoch von JavaDoc keine gültigen XML-Dokumente erzeugt werden können, müssen die HTML-Seiten vor dem Anwenden der XSL-Transformation zunächst in valides XHTML konvertiert werden, so das sie der XML-Spezifikation genügen.

2.6 Zurückschreiben der Kommentare

Um die im Wiki modifizierten Kommentare mit den Java-Quellcodedateien zusammenzuführen, existieren mehrere Verfahren: Generierung neuer Java-Quellcodedateien aus Abstract Syntax Trees oder Suchen der ursprünglichen Kommentare und Ersetzen durch diejenigen aus dem Wiki.

Ein Abstract Syntax Tree enthält alle zum Erzeugen einer Java-Quellcodedatei notwendigen Informationen, die durch einen Tiefendurchlauf extrahiert und in eine neue Java-Quellcodedatei geschrieben werden können. Bei diesem Verfahren werden die Informentare aus dem Wiki vor dem zurückschreiben im AST ersetzt.

Mithilfe der im Wiki gespeicherten Informationen ist es durch eine Suchen-und-Ersetzen-Strategie ebenfalls möglich, die ursprünglichen Kommentare direkt in den Java-Quellcodedateien durch die im Wiki geänderten Kommentaren zu ersetzen. Jedem im Wiki gespeicherten Eintrag ist eindeutig eine Java-Quellcodedatei sowie das dazugehörige Element – wie Attribut, Konstruktor, Methode etc. – zugeordnet. Nach diesem Element kann gezielt in der Java-Quellcodedatei gesucht werden und der darüber liegende JavaDoc-Kommentar gezielt ersetzt werden.

2.7 Projektverwaltung

Zum anlegen eines Projektes sind mehrere, aufeinander aufbauende Arbeitsschritte notwendig: In einem ersten Schritt müssen die Java-Quellcodedateien geladen werden, z.B. aus den verbreiteten CVS und Subversion Respositories; in einem zweiten Schritt werden aus den geladen Java-Quellcodedateien die JavaDoc-HTML-Seiten erzeugt; diese werden in einem dritten Schritt zu XHTML-Seiten konvertiert; im vierten Schritt werden XHTML-Seiten mithilfe der XSL-Transformation mit den Markierungen erweitert; und im letzten Schritt werden die markierten Kommentare in die Datenstruktur des Wikis übertragen.

Für die meißten dieser Arbeitsschritte ist das Entwicklungswerkzeug ANT[7] geeignet. Bei ANT handelt es sich um ein Werkzeug zum automatisierten Erzeugen von Programmen. Es ist vergleichbar mit dem in der UNIX-Welt verbreiteten Entwicklungswerkzeug `make`. Zum Erzeugen eines Programms werden so genannte Targets aufgerufen welche in einen ANT Skript definiert wurden. Die Targets fassen dabei einzelne Arbeitsschritte zusammen, die zum Erzeugen eines Programms notwendig sind. Dabei können Abhängigkeiten zwischen einzelnen Targets definiert werden. ANT löst diese Abhängigkeiten auf in dem es die Targets in der entsprechenden Reihenfolge aufruft.

Die Targets enthalten Aufrufe von so genannten ANT-Tasks, welche einzelne, unteilbare Arbeitsschritte darstellen. ANT stellt eine Reihe von vordefinierten Tasks zur Verfügung. Mithilfe dieser Standard-ANT-Tasks können die ersten vier der oben genannten Arbeitsschritte vollständig mit ANT realisiert werden. Auch der letzte, fehlende Arbeitsschritt kann leicht als ANT-Task implementiert werden.

Weiter Aufgaben der Projektverwaltung sind das Aktualisieren und Löschen von Projekten. Auch für diese Arbeitsschritte ist ANT gut geeignet.

[7] http://ant.apache.org/

Außerdem läßt sich die Projektverwaltung durch den Einsatz von ANT gut in bestehende Build-Prozesse integrieren. So ist insbesondere möglich eigene Targets zu schreiben, welche die Projekte auf eine vollkommen andere Art und Weise erstellen als ursprünglich im Programm implementiert.

2.8 Benutzerauthentisierung

Durch die Nutzung des Java Enterprise Frameworks stehen ausgereifte Formen der Benutzerauthentifizierung zur Verfügung. Das Programm nutzt die Möglichkeiten der Benutzerauthentifizierung standardmäßig nur zum Schutz der Projektverwaltung. Nur vorher festgelegte Benutzer sind in der Lage Wiki-Projekte anzulegen, zu aktualisieren oder zu löschen.

Eine weitergehende Form der Benutzerauthentifizierung wurde im Rahmen der Studienarbeit nicht realisiert. Jeder Benutzer mit Zugriff auf die Webseiten des Wikis kann Wiki-Einträge editieren. Der Zugriff auf die Webseiten kann aber vom Administrator des Wikis mithilfe der Mechanismen der JEE2 beliebig eingeschränkt werden.

2.9 Logging

Bei der Protokollausgabe des Wikis können zwei Arten unterschieden werden.

- **Debuging**
  Um die Arbeitsweise des Programms besser nachvollziehen zu können und zum Auffinden von Fehlern sollten wichtige Arbeitsschritte in einer Log-Datei geschrieben werden können. Art und Umfang der Ausgabe sollte dabei vom Administrator des Wikis beliebig konfigurierbar sein.

- **Sicherheit**
  Zusätzlich sollten Veränderungen des Wikis protokolliert werden. Die Änderungen des Wikis sollten damit zurückverfolgbar sein. Die Protollierung sollte u.a. die IP-Adresse und wenn vorhanden den registrierten Benutzer umfassen.

Um diese Anforderungen zu Erfüllen sollte das Logging Framework Log4j[8] verwendet werden. Die Ausgabe ist fast beliebig konfigurierbar und verschiede Protokollarten können separat behandelt werden. So ist es zum Beispiel möglich die Protokollinformationen über die Arbeitsweise des Wikis zu verwerfen, aber die protokollierten Zugriffe dauerhaft auf der Festplatte zu speichern.

[8] http://logging.apache.org/log4j/docs/

3 Implementierung

Die Umsetzung der in der Entwurfsphase getroffenen Design-Entscheidungen wird im Folgenden anhand von typischen Arbeitsabläufen des Programms erläutert.

3.1 Projektverwaltung

WikiDoc stellt über ein Web-Frontend eine Konfigurationsoberfläche für die Verwaltung von Projekten zur Verfügung. Jedes Projekt umfasst ein einzelnes in sich abgeschlossenes Java-Projekt und enthält somit eine Sammlung von Java Klassen und den dazugehörigen JavaDoc-HTML-Seiten. Über die Konfigurationsoberfläche können mehrere Projekte verwaltet werden: Sie können angelegt, aktualisiert und gelöscht werden.

Die Konfigurationsoberfläche bildet dabei lediglich eine grafische Oberfläche für das

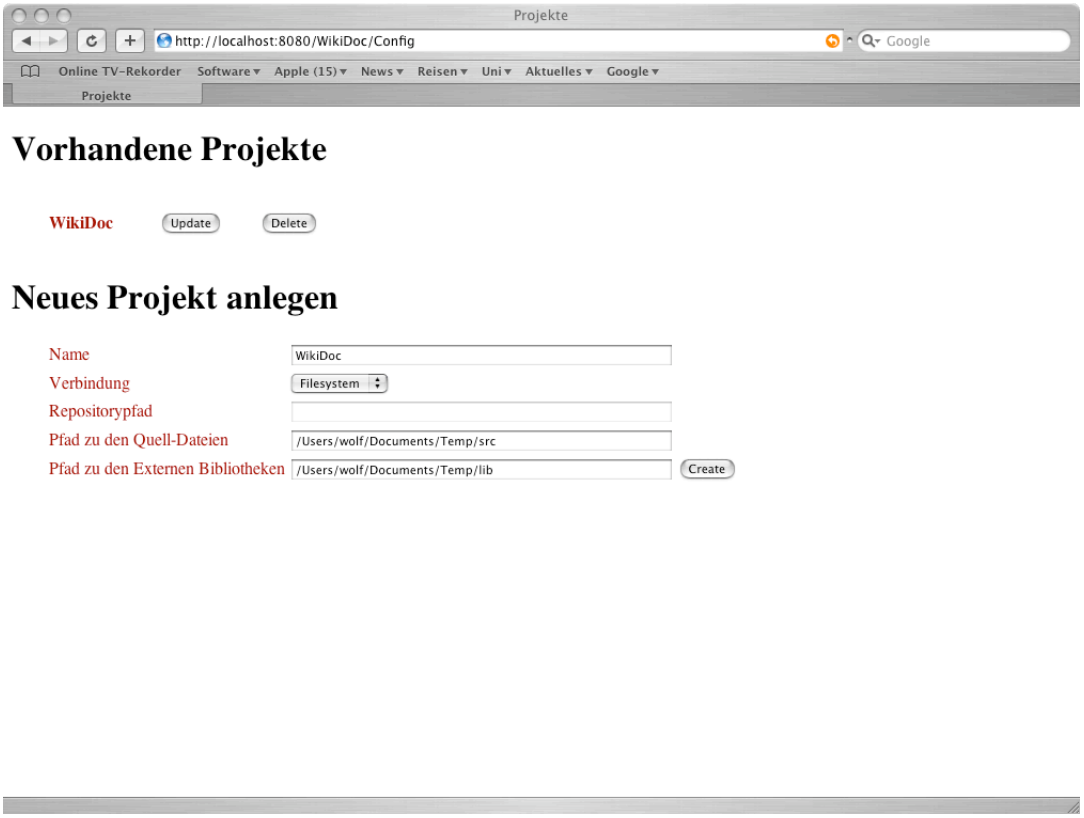

Abbildung 1: Konfigurationsoberfläche

ANT-Skript. Die vom Benutzer aufgerufenen Aktionen werden mit den entsprechenden Parametern an das ANT-Sript übergeben. Dazu werden vom Programm die entsprechenden ANT-Targets aufgerufen und im ANT-Skript die entsprechenden Parameter

gesetzt.

3.2 Erstellen eines Projektes

Über ein Webformular werden die zum Erstellen eines Projektes notwendigen Parameter vom Benutzer abgefragt und an das Programm übergeben. Der Benutzer gibt dazu im Formular dem Projekt einen eindeutigen Namen, über welches das Projekt identifiziert wird.

Im nächsten Schritt gibt er die Methode an, mit welcher die Java-Quellcodedateien geladen werden: Es besteht die Möglichkeit die Dateien über das lokal Dateisystem des Host-Rechner des Wikis oder über das im Internet verbreitete Versionsverwaltungsprogramm CVS zu laden. Im Rahmen der Studienarbeit wurden keine weiteren Methoden bereitgestellt. Aufgrund des modularen Aufbaus des darunter liegenden ANT-Skriptes sind jedoch auch andere Methoden zum Laden der Java-Quellcodedateien wie z.B. aus Subversion leicht zu implementieren.

Wird als Methode CVS verwendet, muss der Benutzer das zu verwendende CVS-Repository und den Pfad der zu ladenen Dateien innerhalb des Repositories angeben. Zusätzlich kann der Benutzer auch einen weiteren Pfad innerhalb des Repositories definieren, aus welchen abhängige Bibliotheken geladen werden.

Wird hingegen als Methode das Laden über das Dateisystem verwendet, gibt der Benutzer einen gültigen Pfad innerhalb des Dateisystems des Servers an, aus welchen die Java-Quellcodedateien geladen werden, sowie einen Pfad zu den abhängigen Bibliotheken.

Die mit dem Absenden des Webformulares übergebenen Parameter werden vom Programm ausgewertet und an das ANT-Skript übergeben. Die Parameter werden außerdem für die spätere Verwendung in einer Properties-Datei gespeichert, welche zum Beispiel beim Aktualisieren eines Projektes ausgewertet wird. Beim Erstellen eines Projektes werden außerdem die Statusmeldungen des ANT-Skriptes auf einer Webseite ausgegeben, um den Anwender über den Fortgang des Prozesses und eventuell auftretende Fehler zu informieren.

3.2.1 ANT-Skript

Um ein Projekt zu erstellen. werden von dem ANT-Skript folgende Arbeitsschritte ausgeführt, die mit den ANT-Tasks übereinstimmen:

1. **Vorbereitung**
   In diesem Arbeitsschritt werden die für das Projekt notwendigen Verzeichnisse erstellt.

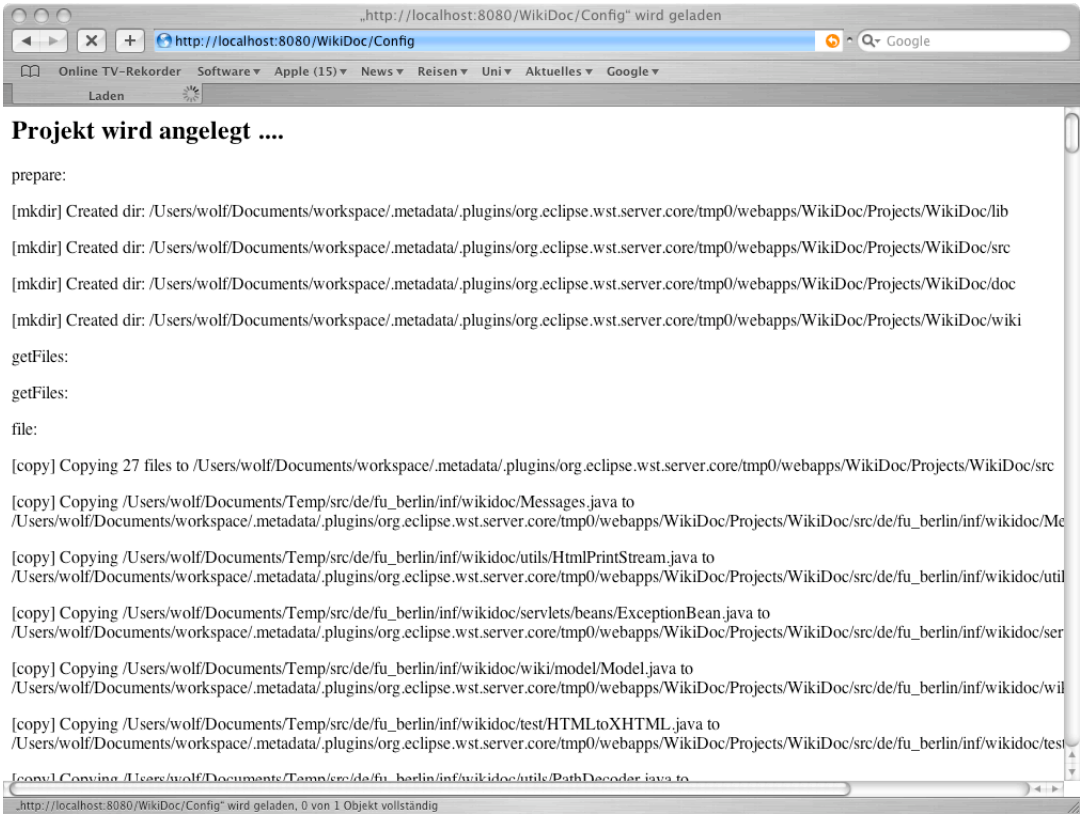

Abbildung 2: Ausgabe des ANT Skriptes beim erstellen eines Projektes

2. **Laden der Quellen**
   Mithilfe der Standard-ANT-Tasks werden je nach ausgewählter Methode die Datei in das vorher erstellte Projektverzeichnis geladen.

3. **Erzeugung der JavaDoc-HTML-Seiten**
   Aus den geladenen Java-Quellcodedateien werden mithilfe des JavaDoc-Generators von Sun die JavaDoc-HTML-Seiten erzeugt.

4. **Konvertierung der JavaDoc-HTML-Seiten nach XHTML**
   Um die Seiten anschließend mit XML-Werkzeugen bearbeiten zu können, werden diese mithilfe des Programmes nekoHTML[9] in XHTML konvertiert.

5. **Anwendung des Stylesheets**
   Nach der Konvertierung wird das XSLT-Stylesheet auf die XHTML-Dateien angewandt, d.h. es werden die notwendigen Markierungen eingefügt. Die so erzeugten Dateien dienen als Grundlage für das neu anzulegende Wiki-Projekt.

[9] http://people.apache.org/~andyc/neko/doc/html/

6. **Extrahieren der Informationen**
   Aus den markierten XHTML-Dateien werden alle JavaDoc-Kommentare extrahiert und mithilfe von XMLBeans in separaten XML-Dateien gespeichert.

3.2.2 XSL-Stylesheet

Ein XSL-Stylesheet besteht aus einer Folge von Templates. Diese fassen Regeln zur Transformation des Quellbaumes zusammen. Sie werden entweder über ihren Namen identifiziert oder über ein frei definierbares Suchmuster. In letzteren Fall werden die Templates nur bei einem zutreffenden Suchmuster aufgerufen. Treffen die Suchmuster mehrerer Templates auf einen Knoten des Quelldokumentes zu, so verwendet der XSLT-Prozessor das Template mit dem spezifischeren Suchmuster.

So hat zum Beispiel das Suchmuster `/HTML/HEAD`, welches auf das HEAD-Element eines HTML-Dokumentes verweist, Vorrang vor dem allgemeineren Suchmuster `node()`, welches auf jeden Knoten des HTML-Dokumentes angewandt werden kann. Würde ein Stylesheet, bestehend aus zwei Templates mit den ebend genannten Suchmustern, auf ein HTML Dokument angewandt, so würde immer das Template mit dem Suchmuster `node()` aufgerufen, außer der Traversierte Knoten im HTML Dokument wäre das HEAD-Element. In diesem Fall würde das Template mit dem spezifischeren Suchmuster `/HTML/HEAD` aufgerufen.

Das XSL-Stylesheet von WikiDoc definiert Templates, um den DOM-Baum des XHTML-Dokumentes zu traversieren und die für das Wiki relevanten JavaDoc-Kommentare zu markieren. Dazu werden alle HTML-Elemente unverändert in das Ergebnisdokument übernommen und die zu markierenden Stellen zusätzlich mit im Browser nicht sichtbaren SPAN-Tags umschlossen. Die eingefügten Tags können anschließend im oben genannten Arbeitsschritt „Parsen der Informationen“ ausgewertet werden. Da das Parsen der HTML-Seiten ausschließlich als XSL-Stylesheet realisiert ist, stellt das Stylesheet einen wichtigen Teil der Implementierung dar und soll im Folgenden anhand von ausgewählten Beispielen erläutert werden.

Standard-Template: Das Standard-Template wird auf jeden Knoten angewandt, der nicht markiert werden soll. Es definiert dazu das allgemeines Suchmuster `node()`, welches auf alle Knoten des Quelldokumentes zutrifft, welche von keinem anderen Template abgedeckt werden. Das Template übernimmt jeden Knoten unverändert mit der `copy`-Anweisung in das Ergebnisdokument, kopiert mit der `copy-of`-Anweisung dessen Attribute und weist über `apply-templates` den XSLT-Prozessor an, rekursiv mit den Kindelementen fortzufahren.

```
<xsl:template match="node()">
  <xsl:copy>
    <xsl:copy-of select="@*" />
```

```
    <xsl:apply-templates select="child::node()" />
  </xsl:copy>
</xsl:template>
```

Neben dem Standard-Template existieren weitere Templates: Ein Teil davon kann nur auf ein einziges Element der JavaDoc-XHTML-Seiten angewandt werden und markiert nur einen einzelnen JavaDoc-Kommentar. Andere Templates markieren komplette Bereiche, z.B. die komplette Beschreibung aller Methoden.

Klassen- und Package-Namen: Die beiden folgenden Templates markieren den Klassen- und Package-Namen einer Java-Klasse. Sie definieren dazu ein sehr spezifisches Suchmuster. Das Template zum Markieren des Klassenames sucht nach der ersten H2-Überschrift im HTML-Dokument.

```
<xsl:template match="HTML/BODY/H2[1]/text()">
  <xsl:element name="SPAN">
    <xsl:attribute name="type">class</xsl:attribute>
    <xsl:attribute name="descr">class_name</xsl:attribute>
    <xsl:copy-of select="normalize-space()" />
  </xsl:element>
</xsl:template>
```

Das Template zum Markieren des Package-Names sucht ebenfalls in der ersten H2-Überschrift, setzt aber zusätzlich voraus, dass der Package-Name von einem FONT-Tag umschlossen ist.

```
<xsl:template match="HTML/BODY/H2[1]/FONT">
  <xsl:copy>
    <xsl:copy-of select="@*" />
      <xsl:element name="SPAN">
      <xsl:attribute name="type">class</xsl:attribute>
      <xsl:attribute name="descr">package_name</xsl:attribute>
      <xsl:value-of select="normalize-space()" />
    </xsl:element>
  </xsl:copy>
</xsl:template>
```

Die Templates kopieren den durch das Suchmuster angegebenen Knoteninhalt mithilfe von `copy` und `copy-of` in das Ergebnisdokument und fügen zugleich mithilfe von `element` und `attribute` die zur Markierung verwendeten SPAN-Tags mit den entsprechenden Attributen hinzu. Über das eingefügte `type`-Attribut des SPAN-Tags werden Informationen des gleichen Types gekennzeichnet. Informationen des gleichen Typs stehen im logischen Zusammenhang zueinander. So werde zum Beispiel alle Informationen die eine Java-Klasse Beschreiben mit dem `type`-Wert `class` gekennzeichnet.

Das ebenfalls hinzugefügte `desc`-Attribut kennzeichnet dagegen eindeutig eine einzelne Information. Auf diese Art und Weise ist es möglich aus dem DOM-Baum gezielt Elemente herauszulesen. So ist es zum Beispiel möglich mithilfe des `type`-Attributes über alle Methoden einer Klasse zu iterieren oder mithilfe des `desc`-Attributes gezielt den Klassennamen herauszulesen.

Die XSL-Transformation durch die beiden Templates soll anhand des folgenden HTML-Fragmentes exemplarisch demonstriert werden.

```
<HTML>
<HEAD>...</HEAD>
<BODY>
...
<H2>Klassenname<FONT>Packagename</FONT></H2>
...
</BODY>
</HTML>
```

Das Ergebnis der Transformation enthält alle Elemente des Ursprungsdokumentes und zusätzlich die für das Wiki notwendigen Erweiterungen.

```
<HTML>
<HEAD>...</HEAD>
<BODY>
...
<H2><SPAN type="class" desc="class_name">Klassenname</SPAN>
<FONT>
<SPAN type="class" desc="package_name">Packagename</SPAN>
</FONT></H2>
...
</BODY>
</HTML>
```

Kurzbeschreibungen: Die Struktur der Kurzberschreibungen von Attributen, Konstruktoren und Methoden unterscheidet sich in den JavaDoc-HTML-Seiten nicht. Aus diesem Grund wird zum Markieren der Informationen dieser drei Beschreibungen ein parametrisiertes Template mit dem Namen `short` angewandt.

```
<xsl:template name="short">
  <xsl:param name="type" />
  ...
</xsl:template>
```

Der Parameter `type` muss beim Aufruf des Templates übergeben werden. Dieser Aufruf erfolgt in einem anderem Template, z.B. das Template zum Markieren der Methoden-Kurzbeschreibungen, welches auf die erste Tabelle nach dem Anker mit dem Namen `method_summery` angewandt wird.

```
<xsl:template
  match="TABLE[preceding-sibling::A[@name='method_summary']][1]">
  <xsl:call-template name="short">
    <xsl:with-param name="type">method</xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Die Templates zum Auswerten der Attribute und Konstrukturen können auf diese Art und Weise das Template `short` wiederverwenden.

Detailbeschreibungen: Auch die Detailbeschreibungen der Attributen, Konstruktoren und Methoden haben eine ähnliche Struktur: Die Beschreibung der Methoden enthält alle mögliche Bestandteile, wie z.B. Erläuterung, Parameter, Rückgabewert und Ausnahmen; Kontruktoren enthalten ebenfalls alle Bestandteile mit Ausnahme des Rückgabewertes; Attribute enthalten nur eine Erläuterung. Zur Auswertung der Detailbeschreibungen wird daher auch ein parametrisiertes Template mit dem Namen `detail` (siehe Anlage) verwendet.

3.2.3 Erstellung der Wiki-XML-Dateien

Nachdem die Kommentare in den JavaDoc-HTML-Dateien durch das XSL-Stylesheet markiert wurden, ist es möglich diese mit dem Programm auszulesen. Dazu werden die HTML-Dateien erneut als DOM-Objekte eingelesen und alle vom XSL-Stylesheet eingefügten SPAN-Tags traversiert.

Die gefundenen JavaDoc-Kommentare werden anschließend mithilfe von XMLBeans in XML-Dateien gespeichert, wobei für jede HTML-Seite eine eigene XML-Datei erzeugt wird. Da in JavaDoc selbst innere Klassen durch eine eigene HTML-Seite repräsentiert werden, wird in jeder XML-Datei die Informationen genau einer Java-Klasse zusammengefasst. Da jeder Kommentar im Wiki modifiziert werden kann und der Zugriff auf früher Versionen möglich sein soll, werden in der XML-Datei zu jedem Kommentar alle Versionen gespeichert. Damit können die XML-Dateien folgende Elemente beinhalten:

Wurzelknoten: Der Wurzelknoten `WikiPage` der XML-Datei speichert alle zu einer Klasse gehörenden Informationen. Der Knoten besitzt aus diesem Grund Attribute für Package- und Klassennamen. Er kann als Kindknoten Einträge für die Klassenbeschreibung (`Description`) und die in der Klasse enthaltenden Methoden (`Methods`), Konstruktoren (`Constructors`) und Attribute (`Fields`) enthalten. Zusätzlich kann die

Klasse über das Attribute `removed` als gelöscht markiert werden. Über das Wiki können zwar keine Klassen gelöscht werden, aber nach der Aktualisierung eines Projektes können im Wiki noch Informationen über gelöschte Klassen enthalten sein, die dann über dieses Attribut markiert werden.

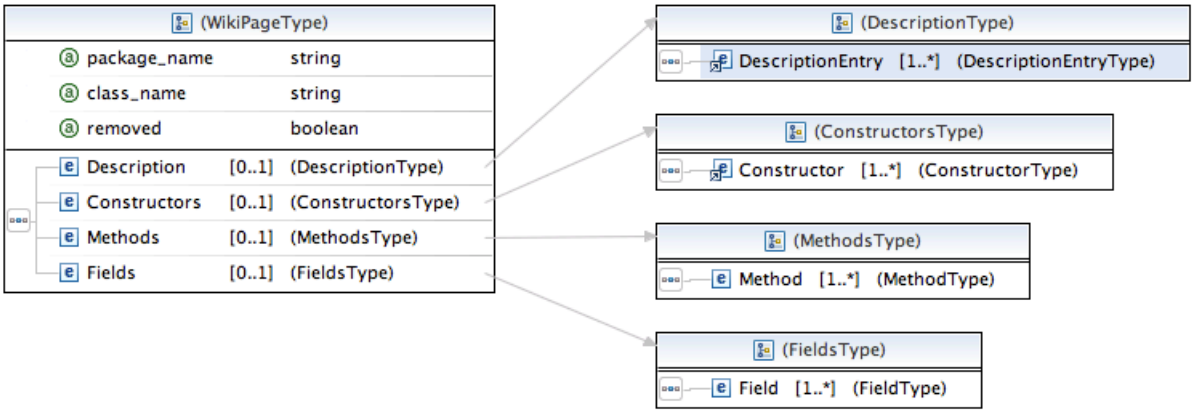

Abbildung 3: Wurzelknoten

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<stor:WikiPage name="Class FooBar" package="test"
    xmlns:stor="store.wiki.wikidoc">
  <stor:Description>...</stor:Description>
  <stor:Constructors>...</stor:Constructors>
  <stor:Methods>...</stor:Methods>
  <stor:Fields>...</stor:Fields>
</stor:WikiPage>
```

Klassenbeschreibungsknoten: In dem Klassenbeschreibungsknoten `Description` werden die verschiedenen Versionen der Klassenbeschreibung gespeichert. Für jede Version wird eine neuer Kindknoten `DescriptionEntry` angelegt. Jedem Versionseintrag wird ein Änderungsdatum, ein Benutzer und ein Kommentar zugeordnet. Zusätzlich wird in dem Attribut `codesync` gespeichert, ob der Eintrag aus dem Java-Quellcodedateien erzeugt wurde.

Beispiel:

```
<stor:Description>
  <stor:DescriptionEntry codesync="true"
```

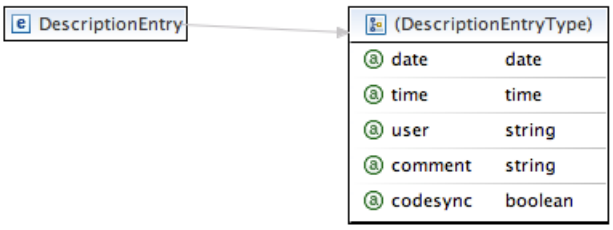

Abbildung 4: Klassenbeschreibungsknoten

```
      date="2006-09-28+02:00" time="15:31:19.627+02:00">
    Beschreibung der Klasse
  </stor:DescriptionEntry>
</stor:Description>
```

Konstruktorknoten: Der `Constructors`-Knoten enthält alle in der Klasse enthaltenden Konstruktoren als `Constructor`-Kindknoten. Die Konstruktoren werden eindeutig über ihre Namen identifiziert, welcher sich aus dem Namen des Konstruktors und der übergebenen Parameter zusammensetzt. Analog zum Wurzelkonten wird über das `removed`-Attribute ein gelöschter Konstruktor markiert. Zu jedem Konstruktor werden in den `ConstructorEntry`-Knoten die verschiedenen Versionen der Kommentare gespeichert.

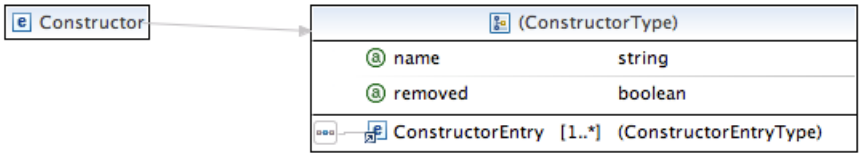

Abbildung 5: Konstruktorknoten

Jeder Versionseintrag beinhaltet die Beschreibung des Konstruktors und die zu dem Konstruktor gehörenden Parameter (`Parameters`) und Ausnahme (`Exceptions`) als Kindknoten. Wie beim `DescriptionEntry`-Knoten werden zu jeder Version ein Änderungsdatum, ein Benutzer, ein Kommentar und das `codesync`-Attribut gespeichert.

Beispiel:

```
<stor:Constructors>
```

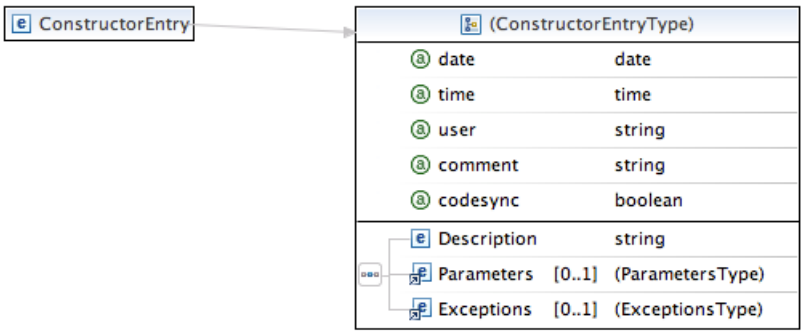

Abbildung 6: Versionseintrag im Konstruktorknoten

```
  <stor:Constructor
      name="FooBar(java.lang.String)" removed="false">
    <stor:ConstructorEntry
        date="2006-09-28+02:00" time="15:46:17.236+02:00"
        codesync="true">
      <stor:Description>
        Beschreibung des Konstruktors.
      </stor:Description>
      <stor:Parameters>...</stor:Parameters>
      <stor:Exceptions>...</stor:Exceptions>
    </stor:ConstructorEntry>
  </stor:Constructor>
</stor:Constructors>
```

Attribut- und Methodenknoten: Der `WikiPage`-Wurzelknoten enthält neben dem `Constructors`-Knoten noch einen `Fields`- und einen `Methods`-Knoten für Attribute und Methoden. Der Aufbau dieser Knoten unterscheidet sich nur geringfügig von dem `Constructors`-Knoten und soll hier nicht weiter erläutert werden.

Parameter- und Ausnahmeknoten: Die `Parameters`- bzw. `Exceptions`-Knoten enthalten alle in einem Konstruktor oder alle in einer Methode enthaltenden Parameter bzw. Ausnahmen als `Parameter`- bzw. `Exception`-Kindknoten. Der Parameter- bzw. der Ausnameknoten speichert den Name und die Beschreibung eines Parameters oder einer Ausnahme. Die Knoten sind Teil eines Versionseintrages eines Konstruktors oder einer Methode. Eine Änderung an einem Paremeter oder einer Ausnahme wird daher als neuer Versionseintrag gespeichert.

Beispiel:

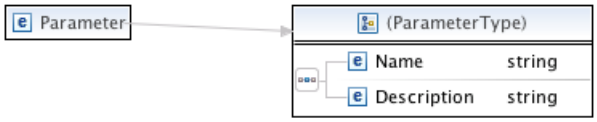

Abbildung 7: Parameterknoten

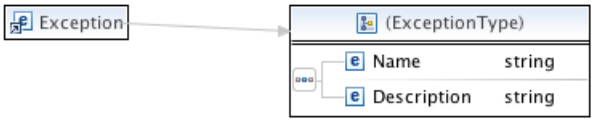

Abbildung 8: Ausnahmeknoten

```
<stor:Parameters>
  <stor:Parameter>
    <stor:Name>&lt;CODE>name&lt;/CODE></stor:Name>
    <stor:Description>
      Beschreibung des Parameters.
    </stor:Description>
  </stor:Parameter>
</stor:Parameters>

<stor:Exceptions>
  <stor:Exception>
    <stor:Name>
      &lt;CODE>java.lang.Exception&lt;/CODE>
    </stor:Name>
    <stor:Description>
      Beschreibung der Ausnahme.
    </stor:Description>
  </stor:Exception>
</stor:Exceptions>
```

3.3 Laden einer Seite

Die Navigation durch die Wiki-Seiten unterscheidet sich für den Benutzer nicht von der Navigation durch die normalen JavaDoc-HTML-Seiten. Die Wiki-Seiten gleichen den JavaDoc-HTML-Seiten im Bezug auf den Inhalt und das Layout. Lediglich die Links zum Bearbeiten eines Eintrages sind bei den Wiki-Seiten hinzugefügt.

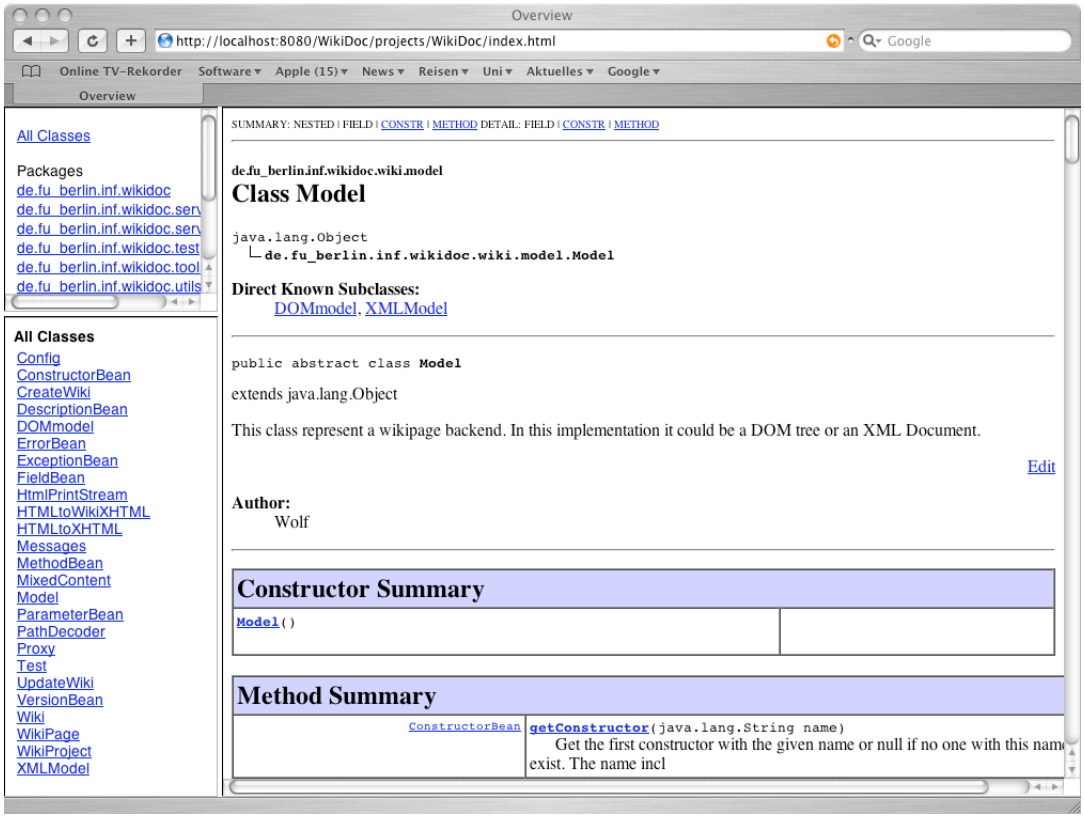

Abbildung 9: WikiDoc-HTML-Seiten

Jede Anfrage des Nutzers erfolgt an ein Proxy-Servlet, welches die Anfrage auswertet und die vom Benutzer angeforderten Dateien aus dem entsprechenden Projektverzeichnis lädt. Dabei werden alle Seiten, welche eine Java-Klasse beschreiben, mit den Informationen aus den Wiki-XML-Dateien kombiniert.

Dazu werden die mit Markierungen versehenen JavaDoc-HTML-Dateien erneut in einen DOM-Baum eingelesen. Im DOM-Baum werden die markierten Kommentare durch die jeweils neuesten Versionen aus den XML-Dateien ersetzt. Existieren in den Wiki-XML-Dateien Kommentare zu Attributen, Konstruktoren oder Methoden, welche in den Java-Quellcodedateien gelöscht wurden, werden die betroffenen Stellen mit einer anderen Schriftfarbe in den DOM-Baum eingefügt. Solche Kommentare können im Wiki nicht mehr bearbeitet, sondern nur noch gelesen und gelöscht werden. Aus diesen modifizierten DOM-Baum wird eine neue JavaDoc-HTML-Seite erzeugt und dem Benutzer präsentiert.

3.4 Bearbeiten einer Seite

Hat der Benutzer in den WikiDoc-HTML-Seiten einen Kommentar zum Bearbeiten ausgewählt, eine Seite auf der der Benutzer die einzelnen Bestandteile, wie z.B. Konstruktor- und Parameterbeschreibung, eines JavaDoc-Kommentares bearbeiten kann. Das Bearbeiten einer Seite wird mit dem Speichern-Befehl abgeschlossen und die Änderungen werden in den Wiki-XML-Dateien als neue Version zu diesen Kommentar hinzugefügt. Alternativ ist es möglich die Bearbeitung einer Seite abzubrechen.

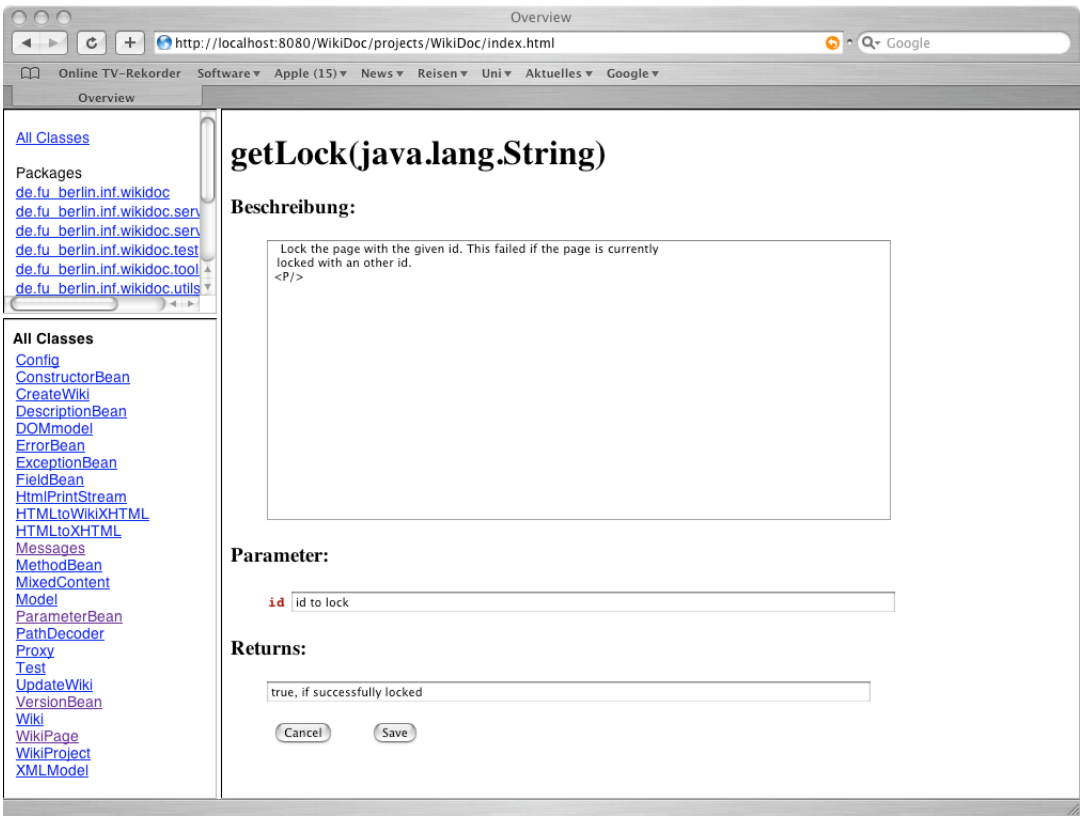

Abbildung 10: WikiDoc Edit-Seiten

Ältere Versionen eines Eintrages können aus einer Liste ausgewählt und durch das erneute Abspeichern des Kommentars wiederhergestellt werden. Dabei kennzeichnen farbige Markierungen in der Auswahlliste Versionen die nicht durch das Wiki erzeugt, sondern aus den Java-Quellcodedateien ausgelesen wurden (siehe `codesync`-Attribute in Abschnitt 3.2.3).

Nur jeweils ein Benutzer kann den Eintrag einer Seite bearbeiten. Mit dem Begin der Bearbeitung wird die Seite für alle anderen Benutzer zum Bearbeiten gesperrt. Diese Sperrung bleibt solange bestehen, wie die Bearbeitung durch Speichern oder Abbrechen

nicht abgeschlossen ist oder die Sperrung nach einer bestimmten Zeit ihre Gültigkeit verliert. Die Sperrung von Seiten soll verhindern, dass mehrere Benutzer zeitgleich eine Seite verändern und damit die Benutzer unwissentlich die Kommentare gegenseitig überschreiben.

3.5 Aktualisierung eines Projektes

Projekte können über die Konfigurationsoberfläche aktualisiert werden. Bei der Aktualisierung werden die Java-Quellcodedateien des Projektes in der aktuellen Fassung geladen und die darin enthaltenen JavaDoc-Kommentare in das Wiki integriert. Dabei werden die gleichen ANT-Tasks wie beim Erstellen des Projektes genutzt.

3.5.1 ANT-Skript

Um ein Projekt zu aktualisieren. werden von dem ANT-Skript folgende Arbeitsschritte ausgeführt, die mit den ANT-Tasks übereinstimmen:

1. **Vorbereitung**
   Die beim Erstellen des Projektes gespeicherten Einstellungen werden aus der Properties-Datei geladen.

2. **Laden der Quellen**
   Mithilfe der Standard-ANT-Tasks werden die Java-Quellcodedateien aktualisiert. Dabei werden sowohl beim Laden aus dem CVS als auch beim Laden über das Dateisystem nur die geänderten Dateien übertragen und im Original gelöschte Dateien aus dem Projektverzeichnis entfernt.

3. **Erzeugung der JavaDoc-HTML-Seiten**
   Der JavaDoc-Generator wird mit den aktualisierten Java-Quellcodedateien aufgerufen. Dabei werden die alten JavaDoc-HTML-Dateien überschrieben. In den Java-Quellcodedateien gelöschte Klassen gehen durch die Aktualisierung nicht verloren. Aus diesem Grund werden geänderte Klassen aktualisiert und gelöschte Klassen bleiben weiterhin im Wiki erhalten.

4. **Konvertierung der JavaDoc-HTML-Seiten nach XHTML**
   Wie beim Erstellen eines Projektes werden die erzeugten JavaDoc-HTML-Seiten werden vor der Weiterverarbeitung mithilfe von NekoHTML in XHTML-Seiten konvertiert.

5. **Anwendung des Stylesheets**
   In den XHTML-Datein werden die relevanten Stellen mithilfe des Stylesheets markiert.

6. **Extrahieren der Informationen**
   Um die aktualisierten Dateien in das Wiki zu integrieren, werden alle markierten XHTML-Seiten untersucht und mit den Wiki-XML-Dateien abgeglichen. Dabei sind folgende Fälle möglich:

   - **Neue Klasse:** Es existiert zu der JavaDoc-XHTML-Datei keine korrespondierende Wiki-XML-Datei. In diesem Fall handelt es sich um eine neue Klasse und die XHTML-Datei kann genau wie beim Erstellen eines Projektes verarbeitet werden.
   - **Aktualisierte Klasse:** Es wird zu JavaDoc-XHTML-Datei eine korrespondierende Wiki-XML-Datei gefunden. In diesem Fall handelt es sich um eine dem Wiki bekannte Klassen und die Wiki-XML-Datei wird aktualisiert.
   - **Gelöschte Klasse:** Es existiert zu einer Wiki-XML-Datei keine korrespondierende JavaDoc-XHTML-Datei. In diesem Fall wurde die Java-Klasse seit der letzten Aktualisierung gelöscht oder umbenannt. Das Wiki ist dadurch nicht in der Lage, die Informationen automatisch abzugleichen und das Programm kann die Klasse lediglich über das Attribut `removed` in der Wiki-XML-Datei als gelöscht markieren. Die Wiki-Seite der Klasse kann dadurch von den Benutzern nicht mehr verändert, sondern nur noch gelöscht werden.

Während der Aktualisierung bleibt das Wiki weiterhin benutzbar, da die Aktualisierung seitenweise erfolgt und durch das Wiki lediglich die gerade aktualisierte Datei für das Bearbeiten gesperrt wird.

3.5.2 Aktualisierung der Wiki-XML-Dateien

In dem Falle das zu einer JavaDoc-XHTML-Datei eine korrespondierende Wiki-XML-Datei existiert, wird die die Wiki-XML-Datei aktualisiert. Das Programm extrahiert dazu die Kommentare aus der markierten JavaDoc-XHTML-Datei und vergleicht diese mit den Einträgen in der Wiki-XML-Datei. Dabei können folgende Fälle auftreten:

- **Neuer Kommentar:** Ein Eintrag, wie z.B. ein neues Attribut oder ein neuer Konstruktor, ist in der JavaDoc-XHTML-Datei enthalten, aber nicht in der Wiki-XML-Datei. Es handelt sich also um einen neuen Eintrag. Dieser Eintrag wird der Wiki-XML-Datei hinzugefügt.

- **Aktualisierter Kommentar:** Ein Eintrag ist der JavaDoc-XHTML-Datei und in der Wiki-XML-Datei enthalten. In diesem Fall wird dem entsprechenden Eintrag eine neue Version in der Wiki-XML-Datei hinzugefügt. Daduch zeigt das Wiki nach der Aktualisierung die geänderten Kommentare so an, wie sie den Java-Quellcodedateien stehen. Die alten Einträge gehen durch die Versionsverwaltung des Wikis jedoch nicht verloren, sie können von den Benutzern wiederhergestellt werden.

- **Gelöschter Kommenter:** Ein Eintrag, der nur in den Wiki-XML-Dateien vorhanden ist, nicht jedoch in den aktualisierten JavaDoc-XHTML-Dateien, wird über das Attribut `removed` in der Wiki-XML-Datei als gelöscht markiert. Ein solcher Eintrag kann von den Benutzern nur betrachtet oder gelöscht, nicht jedoch erneut bearbeitet werden.

Jede Änderung an einem Eintrag in der Wiki-XML-Datei, die durch die Aktualisierung des Wikis verursacht worden ist, wird mit dem `codesync`-Attribut versehen. Damit wird gekennzeichnet, dass es sich nicht um im Wiki vorgenommene Benutzereinträge, sondern um Einträge aus dem Java-Quellcode handelt.

3.6 Löschen eines Projektes

Zum Löschen eines Projektes wird durch das ANT-Skript der Projektordner mit allen enthaltenen Dateien gelöscht. Dadurch dass die Dateien nicht mehr existieren, werden alle Anfragen an das Proxy-Servlet, die auf das alten Projekt verweisen, mit einer Fehlermeldung beendet. Da auch die Projektauflistung der Konfigurationsoberfläche auf den Verzeichnissen basiert, wird das Projekt dort ebenfalls nicht mehr aufgelistet.

3.7 Zurückschreiben der Kommentare

Das Zurückschreiben der Kommentare in die JavaDoc-Quellcodedatein wurde im Rahmen der Studienarbeit nicht mehr realisiert. Durch den in der Entwurfsphase verworfenen AST-Parser fehlte die Zugriffsmöglichkeit auf den Java-Quellcode. Die angedachte Suchen-und-Ersetzen-Strategie scheiterte, da sich die Ausgabe des JavaDoc-Generators stärker als gedacht von den Kommentaren im Quellcode unterscheiden konnte. Zum einen enthält der Original-JavaDoc-Kommentar in der Quellcodedatei Annotationen und Verweise die durch den Generator aufgelöst werden, und zum anderen kann sich das Textlayout der Kommentare stark unterscheiden.

4 Ergebnisse

Die entstandene Anwendung erfüllt die meisten Anforderungskriterien. Es ist möglich das Wiki aus den Java-Quellcodedateien aufzubauen und die JavaDoc-Kommentare zu bearbeiten. Die Installation und die Erstellung von Wiki-Projekten sind einfach und flexibel gestalltet, so dass sich das Wiki gut in bestehende Projekte integriert lässt. JavaDoc-Kommentare, die vom Wiki nicht unterstützt werden, werden dem Benutzer unverändert angezeigt. Durch die Verwendung des JavaDoc-Generators von Sun wird nicht nur die Unterstützung aktueller Sprachelemente gesichert, sondern es bleibt auch das Original-Layout der Webseiten erhalten, inclusive aller Index- und Übersichtsseiten. Weiterhin ist es möglich das Wiki aus den Java-Quellcodedateien zu aktualisieren und damit der Weiterentwicklung des Quellcodes Rechnung zu tragen.

Die Anforderungen zum Zurückschreiben der im Wiki geänderten Kommentare wurden jedoch verfehlt. Mit dem gewählten Ansatz zum Extrahieren der JavaDoc-Informationen ließ sich das Wiki zwar zu Beginn einfacher programmieren, konnten jedoch so nicht alle gestellten Anforderungen erfüllen. Das Extrahieren der Informationen aus den layoutorientierten HTML-Seiten mithilfe eines Stylesheets hat sich als Fehler herausgestellt. Zum einen ist es schwierig die XSL-Transformation zum Verarbeiten nicht strukturierter Informationen zu verwenden und zum anderen ist das Zurückschreiben in den Java-Quellcode nur mit dem AST-Parser-Ansatz sinnvoll realisierbar. Wird jedoch ein AST-Parser für das Zurückschreiben verwendet, lassen sich die Kommentare ebenfalls auf dem selben Wege aus den Java-Quellcodedateien extrahieren. Der Umweg über die HTML-Seiten ist dann obsolet.

Obwohl es momentan nicht möglich ist die geänderte Kommentare in die Java-Quellcodedateien zu übertragen kann das Wiki jedoch auch ohne diese Fähigkeit eingesetzt werden. Alle grundlegenden Wiki-Funktionen sind vorhanden. Entwickler können die Java-API ihrer Projekte in Form des Wikis veröffentlichen und somit, bei einer entsprechenden Benutzerbeteiligung, zumindest ihre öffentlich zugängliche Dokumentation verbessern. Die Entwickler können auch besonders gut dokumentierte Bereiche von Hand zurück in den Quellcode übertragen.

Darüber hinaus ist es auch möglich das Wiki in einer abgewandelten Form einzusetzen. Durch Änderung des zum Erstellen der Wiki-Projekte benutzten Ant-Scriptes kann das Wiki transparent auf jede JavaDoc-Dokumentation gelegt werden, auch ohne direkten Zugriff auf den Quellcode. Das Ant-Script muss dazu die HTML-Seiten lokal Zwischenspeichern und mit den selben Ant-Tasks, wie sie auch zur Erstellung der Wiki-Projekte genutzt werden, verarbeiten. Die so erzeugt Dokumentation kann ebenso wie die der ursprünglichen Projekte im Wiki bearbeitet werden. Auf diese Arte und Weise kann das Wiki über bestehende JavaDoc APIs, wie zum Beispiel der Sun-Java-API gelegt werden. Die Benutzer können dann vom Wiki unterstützt die Kommentare leicht bearbeiten und die bestehende Dokumentation verbessern.

A Installation

Das Programm ist als Webapplikation konzipiert und in jedem Servlet Container lauffähig der die JSP-Spezifikationen 2.0 und die Servlet Spezifikationen 2.4 erfüllt. Außer den Spezifikationen und den Bibliotheken die mit dem Programm installiert werden, müssen keine zusätzlichen Abhängigkeiten erfüllt werden. Im Folgenden soll die Installation am Beispiel des Apache Tomcat Servers beschrieben werden. Dabei wird vorausgesetzt, dass der Server installiert, konfiguriert und gestartet wurde.

1. Die von Tomcat bereitgestellten Webapplikationen können über den Tomcat Manager verwaltet werden. Der Manager wird über einen Link auf der Tomcat Startseite aufgerufen.

2. Das Wiki lässt sich über den Manager von Tomcat leicht installieren. Dazu muss lediglich die WAR Datei des Wikis mithilfe der Funktion `Lokale WAR Datei zur Installation hochladen` in den Webserver geladen werden. Anschließen lässt sich das Wiki wie jede andere Webapplikation verwalten. Das Wiki ist einsatzbereit wenn die Webapplikation gestartet wurde und der Wert `true` in der Spalte `Verfügbar` erscheint.

3. Um Wiki-Projekte erstellen zu können muss das `bin`-Verzeichnis der JDK[10]-Installation im ausführbaren Pfad des Webservers liegen.

4. Werden zusätzlich CVS-Repositories genutzt, muss auf dem Webserver ein CVS-Client installiert werden und dieser ebenfalls zum ausführbaren Pfad des Webservers hinzugefügt werden.

5. Vor der ersten Verwendung muss das Programm Konfiguriert werden. Die Konfigurationsoberfläche des Programms ist über die URL `http://<webserver:port>/WikiDoc/Config` erreichbar.

A.1 Anpassung der Installation

Über die Parameter in den folgenden Dateien kann das Verhalten des Programmes angepasst werden.

- **web.xml**
  Über den `Context`-Parameter `project_path` innerhalb der `web.xml`-Datei lässt sich der Pfad des WikiDoc-Projektverzeichnisses anpassen. In diesem Verzeichnis speichert das Wiki alle Informationen zu den verwalteten Projekten. Ohne Angabe eines separaten Projektverzeichnisses wird ein Verzeichnis innerhalb der Webapplikation verwendet, welches allerdings bei Neuinstallation der Anwendung überschrieben wird. Aus diesem Grund ist es sinnvoll als Projektverzeichnis einen

[10] http://sun.java.com

Pfad außerhalb der Webapplikation festzulegen, wobei der neue Projektordner beim Start der Anwendung existieren und alle Dateien aus dem Projektverzeichnis der Webapplikation enthalten muss.

- **log4j.properties**
  Über die Log4j Konfigurationsdatei lässt sich das Protokollverhalten der Anwendung beeinflussen. Dabei können sowohl die Protokollierten Ereignisse definiert werden, als auch das Ausgabeformat.

B Stylesheet-Auszug

```
<!-- Markiere die Methoden-Kurzzusammenfassung -->
<xsl:template
  match="TABLE[preceding-sibling::A[@name='method_summary']][1]">
  <xsl:call-template name="short">
    <xsl:with-param name="type">method</xsl:with-param>
  </xsl:call-template>
</xsl:template>

<!-- Parametrisiertes Template zum
Markieren der Kurzbeschreibungen-->

<xsl:template name="short">
  <xsl:param name="type" />
  <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:copy-of select="./TR[1]" />

    <xsl:for-each select="TR">
      <!-- Tabellenüberschrift weglassen -->
      <xsl:if test="position()>1">
        <xsl:element name="TR">

          <!-- erste Tabellenspalte komplett übernehmen -->
          <xsl:copy-of select="TD[1]" />
          <xsl:element name="TD">

            <!-- Kopiere in der zweiten Zeile den Codeteil -->
            <xsl:copy-of select="TD[2]/CODE[1]" />

            <xsl:element name="BR" />

            <!--<xsl:call-template name="head">
              <xsl:with-param name="input">
              <xsl:copy-of select="TD/( text()|*)"/>
              </xsl:with-param>
              <xsl:with-param name="marker" select="' '" />
              </xsl:call-template>-->
            <xsl:text>
```

```
                  
          </xsl:text>
          <xsl:element name="SPAN">
            <xsl:attribute name="type" select="$type" />
            <xsl:attribute name="descr">
              short_description
            </xsl:attribute>
            <xsl:attribute name="id"
              select="position()-1" />
            <xsl:for-each
              select="TD[2]/text()|TD[2]/*[position() > 2]">
              <xsl:choose>
                <xsl:when test="position()=1">
                  <xsl:call-template name="tail">
                    <xsl:with-param name="input"
                      select="normalize-space(.)" />
                    <xsl:with-param name="marker"
                      select="' '" />
                  </xsl:call-template>
                </xsl:when>
                <xsl:otherwise>
                  <xsl:copy-of select="." />
                </xsl:otherwise>
              </xsl:choose>
            </xsl:for-each>
          </xsl:element>
        </xsl:element>
      </xsl:element>
    </xsl:if>
  </xsl:for-each>
 </xsl:copy>
</xsl:template>


<!-- Markiere die Konstructoren-Details -->
<xsl:template
match="TABLE[preceding-sibling::A[@name='constructor_detail']][1]">
<xsl:choose>
<xsl:when test="following::A[@name='method_detail']">
  <xsl:call-template name="detail">
    <xsl:with-param name="type">constructor</xsl:with-param>
    <xsl:with-param name="end">method_detail</xsl:with-param>
  </xsl:call-template>
</xsl:when>
<xsl:otherwise>
  <xsl:call-template name="detail">
    <xsl:with-param name="type">constructor</xsl:with-param>
    <xsl:with-param name="end">navbar_bottom</xsl:with-param>
  </xsl:call-template>
</xsl:otherwise>
</xsl:choose>
```

```
</xsl:template>


<!-- Parametrisiertes Template zum Markieren
der Detailbeschreibungen -->

<xsl:template name="detail">
<xsl:param name="type" />
<xsl:param name="end" />

<xsl:copy-of select="." />

<xsl:for-each
    select="following::H3[following-sibling::A[@name=$end]]">

<xsl:sort select="." />
<xsl:variable name="nummer" select="position()" />
<xsl:variable name="name" select="preceding::A[1]/@name" />
<xsl:copy-of select="preceding::A[1]" />

<xsl:copy>
  <xsl:element name="SPAN">
    <xsl:attribute name="type" select="$type" />
    <xsl:attribute name="descr">name</xsl:attribute>
    <xsl:attribute name="id" select="$nummer" />
    <xsl:attribute name="name" select="$name" />
    <xsl:value-of select="." />
  </xsl:element>
</xsl:copy>

<xsl:copy-of select="following-sibling::PRE[1]" />

<xsl:element name="SPAN">
  <xsl:attribute name="type" select="$type" />
  <xsl:attribute name="descr">description</xsl:attribute>
  <xsl:attribute name="id" select="$nummer" />
  <xsl:copy-of
    select="following-sibling::DL[1]/DD/(text()|*[not(element())
        or .!=P/DL])" />

</xsl:element>
<xsl:element name="P" />

<xsl:for-each-group select="following-sibling::DL[1]/DD/P/DL/*"
  group-starting-with="DT">
  <xsl:variable name="was"
      select="substring-before(B/text(),':')" />

  <xsl:for-each select="current-group()">
    <xsl:if test="name()='DT'">
      <xsl:copy-of select="." />
```

```
    </xsl:if>
    <xsl:if test="name()='DD'">
      <xsl:element name="DD">
        <xsl:element name="SPAN">
          <xsl:attribute name="type" select="$type" />
          <xsl:attribute name="descr" select="$was" />
          <xsl:attribute name="id" select="$nummer" />
          <xsl:attribute name="count" select="position()-1" />
          <xsl:copy-of select="(child::*|text())" />
        </xsl:element>
      </xsl:element>
    </xsl:if>
  </xsl:for-each>
</xsl:for-each-group>

<xsl:call-template name="edit">
  <xsl:with-param name="name" select="$name" />
  <xsl:with-param name="type">
    <xsl:value-of select="$type" />
  </xsl:with-param>
  <xsl:with-param name="id" select="$nummer" />
</xsl:call-template>
<xsl:element name="HR" />

</xsl:for-each>
<xsl:element name="SPAN">
<xsl:attribute name="type" select="$type"/>
<xsl:attribute name="descr">removed</xsl:attribute>
<xsl:attribute name="id">-1</xsl:attribute>
</xsl:element>
</xsl:template>
```

Literatur

[Gil05] Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070):900–901, December 2005.

[Knu84] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[Law06] Cormac Lawler. Wikipedia als lerngemeinschaft. In *Open Source Jahrbuch 2006*, Open Source Jahrbuch. Lehmanns Media, Berlin, 2006.

[Ray99] Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.