Roland Tapken

Eine SOAP-Schnittstelle für PmWiki

mit Eclipse-RCP als Beispiel-Client

Bachelorarbeit

Betreut von:
Prof. Dr. Oliver Vornberger
Prof. Dr. Joachim Hertzberg

Universität Osnabrück
FB Mathematik/Informatik
5. Dez. 2006 bis 20. Mrz. 2007

Danksagungen

Für die Hilfe und Unterstützung bei der Entstehung dieser Bachelorarbeit möchte ich mich bedanken bei…

Prof. Dr. Oliver Vornberger, für die Anregung zu dieser Bachelorarbeit und der intensiven Betreuung.

Prof. Dr. Joachim Hertzberg, der sich für diese Arbeit als Zweitprüfer zur Verfügung gestellt hat.

Dr. Ralf Kunze, von dem die Anregung kam, Eclipse als Basis für die Entwicklung der eigenen Software zu verwenden.

Michael Valenta, Softwareentwickler bei IBM, für die Hilfe beim Umgang mit den weniger gut dokumentierten Eclipse-Klassen.

Alice Smirnow, für die moralische Unterstützung und ihrer unendlichen Geduld.

Inhaltsverzeichnis

1 Einleitung

Zunächst ein kleiner Kommentar in eigener Sache: Ich gehe davon aus, dass dem Leser die Begriffe *Wiki* und *XML* vertraut sind, dass er bei *Java* nicht zuerst an Kaffee oder Urlaub denkt und er *PHP* nicht für die neuste Partydroge hält. Andernfalls möchte ich auf die unzähligen, zu einem großen Teil frei verfügbaren Informationsquellen im Internet oder auch auf diverse Fachbücher und -zeitschriften verweisen. Ein guter Ausgangspunkt für Informationen über Wikis und über *media2mult*[1] ist der Vortrag, den ich im Rahmen des Seminars "Web Publishing" im Wintersemester 2005/2006 gehalten habe[2].

Die Entstehung dieser Bachelorarbeit hat eine lange Vorgeschichte. Im Rahmen des Programmierpraktikums zur Vorlesung "Datenbanksysteme" im Sommersemester 2005[3] ergab sich das Problem, eine Plattform für die gemeinsame Kommunikation zu finden. Als Ergebnis wurde die webbasierte Wiki-Software *PmWiki*[4] auf einem eigenen Webserver mit PHP- und Apache-Unterstützung installiert.

Schon damals kam die Idee auf, auch die Projektdokumentation mit diesem System zu erstellen und anschließend als PDF-Datei exportieren zu lassen. Leider fand sich kein geeignetes Programm für diese Aufgabe, so dass der Plan verworfen und auf eine ganz klassisch mit LaTeX erstellte Dokumentation zurückgegriffen wurde.

Aber wir waren offensichtlich nicht die einzigen mit dieser Idee, denn kurze Zeit später erfuhr ich, dass das *Zentrum VirtUOS*[5] der Universität Osnabrück mit *media2mult* genau den selben Ansatz verfolgte - zufälligerweise ebenfalls mit PmWiki als Plattform. Es folgte ein erster Praxisversuch im Rahmen des Seminars zum Thema "Web Publishing", bei dem die Ausarbeitung der Vorträge mithilfe von media2mult erstellt werden sollte. Dabei wurde unter anderem erkannt, dass die Verwaltung von eingebundenen Dateien ("Anhänge") noch sehr aufwendig war - eine lokal geänderte Datei musste erst in mehreren Schritten von Hand in das Wiki übertragen werden.

Es entstand die Idee ein Programm zu schreiben, welches lokale Änderungen an eingebundenen Dateien erkennt und mit dem Wiki synchronisiert. Ich griff dieses Thema für meine Bachelorarbeit auf und erkannte im Rahmen der Vorbesprechungen, dass dieser Ansatz durch eine generische Schnittstelle zu PmWiki für verschiedene Aufgaben erweitert werden könnte. Also habe ich mich entschlossen, unter der Betreuung

[1] http://www.virtuos.uni-osnabrueck.de/Content/Media2mult

[2] http://zentrum.virtuos.uos.de/wiki-webpublishing/?n=Media2mult-Ausarbeitung
und auf der beigelegten CD-ROM im Verzeichnis `documents/`

[3] http://www-lehre.inf.uos.de/dbp/2005/

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

[5] http://www.virtuos.uos.de

durch Prof. Dr. Oliver Vornberger eine SOAP-basierte Erweiterung für PmWiki zu schreiben. Für die Client-Software, die der Demonstration der Möglichkeiten und als Basis weiterer Erweiterungen dienen soll, fiel die Wahl nach Anregung durch Dr. Ralf Kunze auf das in Java geschriebene *Eclipse*-Framework, welches eine umfangreiche Bibliothek zur Erstellung, Modifikation und Verwaltung von Dateien zur Verfügung stellt.

Dieses Dokument ist eine Beschreibung der Vorgehensweise bei der Erstellung dieser Programme und der Funktionsweise der genutzten Software. Die beiliegende CD-ROM enthält darüber hinaus die fertigen Programme, die Quelltexte sowie eine ausführliche API-Dokumentation der Klassen.

Aktuelle Versionen der beigelegten Software befinden sich auf meiner Homepage unter http://www.dau-sicher.de/bachelorarbeit/.

2 PmWiki

2.1 Die Wiki-Idee

Die meisten Leute, die das Wort *Wiki* hören, denken zuerst an die berühmt-berüchtigte Wikipedia[1]. In der Tat handelt es sich bei dem im Jahr 2001 gegründeten Projekt (vgl. [1], Artikel "Wikipedia") , welches sich das Ziel gesetzt hat, das "Wissen der Welt" zu sammeln, um das wohl aktuell größte und erfolgreichste, welches die Idee der gemeinschaftlichen und freien Mitarbeit aufgreift.

Tatsächlich ist das Konzept jedoch viel älter. Schon als *Tim Berner-Lee* Anfang der 90er Jahre das *World Wide Web* entwickelte, betonte er, dass das Editieren der Seiten genauso wichtig sei wie das Betrachten (vgl. [1], Artikel "Tim Berners-Lee"). Die erste erfolgreiche Umsetzung dieser Idee geht auf *Ward Cunningham* zurück, der im Jahr 1995 das noch heute aktive *WikiWikiWeb*[2] gründete (vgl. [1], Artikel "WardsWiki"). Er erlaubte es jedermann die Inhalte seiner Homepage zu modifizieren. Das ganze wurde durch eine einfache Formatierungssprache sowie einer Versionsverwaltung als Schutz vor Vandalismus unterstützt.

Heute gibt es hunderte verschiedener Wiki-Engines, die in den verschiedensten Programmiersprachen geschrieben und oft auf bestimmte Einsatzbereiche spezialisiert sind. Das Grundkonzept ist jedoch immer gleich: Einfache Syntax, schnelle Bearbeitung und eine Versionsgeschichte. Da die Grenze zu *Content Management Systemen* fließend ist, ist bei vielen Programmen keine klare Zuordnung möglich. Die bekannteste Software ist sicherlich *Wikimedia*[3], mit der unter anderem auch die erwähnte Wikipedia betrieben wird. Eine in der Installation und Konfiguration sehr simple Alternative ist das in dieser Arbeit verwendete PmWiki.

2.2 PmWiki - Die Umsetzung von Patrick Michaud

Das von Prof. Patrick Michaud für die Website der Texas A & M University mit PHP4 entwickelte Content-Management-System soll vor allem technisch unbedarfte Nutzer ansprechen und ihnen die Scheu vor dem Modifizieren der Inhalte nehmen. Deshalb wurde die Oberfläche einfach und die Formatierungssprache schlank gehalten (erster Grundsatz: "Favor writers over readers", vgl. [2], Artikel "PmWiki Philosophy").

[1] http://www.wikipedia.org
[2] http://c2.com/cgi/wiki
[3] http://www.wikimedia.org

2.3 Erweiterungen durch Cookbooks

Aufgrund der angestrebten Einfachheit beherrscht die Software in ihrer Grundausstattung nicht viel mehr als die wichtigsten Aufgaben eines Wikis. Sie erlaubt aber über ein recht einfaches Plugin-System (als *Cookbook* bezeichnet) nahezu beliebige, in PHP geschriebene Erweiterungen. Obwohl PmWiki fast vollständig prozedural geschrieben ist, kann man durch die an vielen Stellen verwendete *dynamische Bindung* von Funktionsaufrufen und vielen globalen Variablen das System um eigene Programmlogik erweitern. Begrenzt wird dies nur durch die etwas knappe Dokumentation der internen API und der Eigenart, das Programm bei Fehlern einfach zu beenden. Dennoch stehen bereits hunderte Cookbooks zur Verfügung[4].

Beispiele sind *Make Many Columns*[5] zur Erstellung mehrspaltiger Fließtexte oder *PmWiki AuthUser*[6], welches ein Benutzerverwaltungssystem ergänzt.

2.4 media2mult

Auch das *Cross-Publishing*-Tool media2mult ist ein gutes Beispiel für eine solche Erweiterung. Es wird vom Zentrum VirtUOS an der Universität Osnabrück entwickelt und ergänzt PmWiki um Export-Filter für verschiedene Formate (PDF, Postscript, HTML u.a.). Dem Problem, dynamische Inhalte wie Filme “auf’s Papier” zu bringen, begegnet das Modul durch den Einsatz neuer Formatierungsbefehle (*Markup-Tags*), mit denen zum Beispiel alternative Grafiken für Printmedien definiert werden können.

Dieses Konzept ermöglicht unter anderem auch das verteilte Arbeiten an Publikationen. Mehrere Autoren können problemlos zur selben Zeit verschiedene und, mit leichten Einschränkungen, auch gleiche Seite bearbeiten. Die dafür erforderliche Versions- und Konfliktverwaltung bringt PmWiki bereits mit, so dass sich die Entwickler von media2mult auf ihre Kernaufgabe konzentrieren können. Denkbar sind in Zukunft zum Beispiel zentrale *Publikationsserver* von Universitäten, auf denen verschiedene Autoren an ihren Dokumenten von beliebigen Arbeitsplätzen aus arbeiten können.

Allerdings wird eine permanente Internetverbindung benötigt, und Dateianhänge (zum Beispiel Bilder oder Quelltexte) müssen mit viel Aufwand manuell verwaltet werden. Wünschenswert wäre deshalb eine Möglichkeit, die Seiten und Anhänge des Wikis lokal speichern und bei Gelegenheit wieder zurückkopieren zu können. Dann hätte man die Möglichkeit, seine (zuvor kopierten) Seiten auch während einer Bahnfahrt zu bearbeiten und am Zielort diese Änderungen wieder anderen Autoren zur Verfügung zu stellen.

Dies zu ermöglichen war das Ziel der vorliegenden Arbeit. Hierfür gab es nun im Vorfeld zwei Varianten abzuwägen: Der direkte Zugriff per *HTTP* auf die vorhandenen

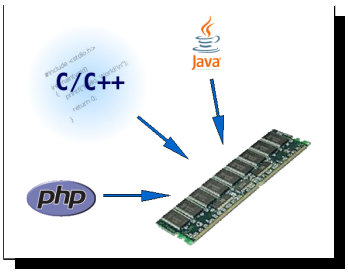
[4] http://www.pmwiki.org/wiki/Cookbook/Cookbook
[5] http://www.pmwiki.org/wiki/Cookbook/MakeManyColumns
[6] http://www.pmwiki.org/wiki/PmWiki/AuthUser

Formulare, also quasi die Simulation eines Webbrowsers mit einem eigenen Client-Programm, oder die Verwendung einer Technologie für entfernte Funktionsaufrufe, sogenannte *Remote Procedure Calls*. Der Vorteil der ersten Variante ist, dass keine Modifikationen an PmWiki notwendig sind. Dies wird allerdings durch den großen Nachteil erkauft, dass bei jeder Änderung des Layouts der Webseite oder bei Aktualisierungen von PmWiki ein neuer Client notwendig wäre. Außerdem bestände ein Großteil der übertragenen Daten aus überflüssigen Layoutinformationen, die vor der Verarbeitung aufwendig gefiltert und entfernt werden müssten. Die RPC-Technologie würde im Extremfall nur ein Update der serverseitigen Implementierung erfordern und wurde deshalb vorgezogen.

3 Verteilte Systeme / Middleware

Einen großer Teil meiner Arbeit beschäftigte ich mich mit Middleware-Technologien und insbesondere dem SOAP-Protokoll. Deshalb folgt nun ein kleines Exkurs in diese Thematik.

3.1 Datenaustausch zwischen Programmen



Wenn zwei Programme miteinander Daten austauschen möchten, dann wird dies in der klassischen Programmierung häufig durch gemeinsam genutzte Bereiche im Arbeitsspeicher realisiert (selbst, wenn man sich als Programmierer in vielen Programmiersprachen nicht explizit darum kümmern braucht). Diese Möglichkeit entfällt jedoch, wenn sich die Programme nicht auf der selben Maschine befinden und über ein Netzwerk kommunizieren müssen.

Heutzutage hat sich hierfür die Verwendung des *UDP*- und des *TCP*-Protokolls durchgesetzt. Die beteiligten Programme bauen untereinander eine Verbindung auf und tauschen durch diesen Kanal die Daten aus. Natürlich müssen beide Programme das selbe Kommunikationsprotokoll beherschen, wie zum Beispiel HTTP für die Übermittlung von Webseiten und Dateien oder *SMTP* für das Versenden von E-Mails.

Middleware-Protokolle bilden eine weitere Abstraktionsschicht, die neben dem Transport auch die Codierung verschiedener Datentypen und den Aufruf von Methoden auf dem Zielrechner regelt. Der Entwickler kann einmal erstellte Bibliotheken für verschiedene Aufgaben verwenden und muss nicht jedesmal ein neues Kommunikationsprotokoll entwerfen. Die beteiligten Systeme müssen nur wissen, welche Methoden mit welchen Paramtern und Rückgabewerten auf dem Zielsystem existieren - also die selben Informationen, die auch für den lokalen Datenaustausch klassischer Programmierung benötigt werden. Beispiele für diese Technik sind das sprach- und plattformunabhängige *Common Object Request Broker Architecture* (CORBA) und das auf Java spezialisierte *Remote Method Invocation* (RMI).

Seit einigen Jahren hat sich XML als plattformunabhängiges Containerformat für die Speicherung und den Austausch von Daten durchgesetzt. Da es einfach zu verarbeiten ist, sind für viele Plattformen fertige Bibliotheken verfügbar. Warum also nicht auch diese Technik verwenden, um Funktionen auf anderen Rechnern aufzurufen und Daten auszutauschen?

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getLocalTime
           soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           xmlns:ns1="urn:SomeNamespace">
   <timezone xsi:type="xsd:string">GMT+01</timezone>
  </ns1:getLocalTime>
 </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.1: SOAP-Anfrage vom Client

3.2 SOAP

Ein solches plattformunabhängiges Protokoll für den entfernten Funktionsaufruf ist *SOAP* (früher ein Akronym für "Simple Object Access Protokoll", heute ein eigenständiger Name). Der durch das *W3C*[1] definierte Standard regelt, wie zwei Systeme mit einer XML-Sprache Daten austauschen können. Das Tägerprotokoll, über das die XML-Daten versendet werden, ist übrigens in neureren Versionen undefiniert. In der Regel werden HTTP und *HTTPS* eingesetzt, aber auch *FTP*, SMTP und andere sind möglich.

3.2.1 Grundlagen

Ein Dienst in einem Netzwerk, der eine SOAP-Schnittstelle anbietet, wird auch als *Webservice* bezeichnet. Ein Programm (*SOAP-Client*), das diesen Dienst nutzen will, muss wissen, welche Methoden mit welchen Argumenten dieser Dienst zur Verfügung stellt. Diese Information wird entweder durch den Programmierer von Hand eingegeben oder aus der Schnittstellenbeschreibung (WSDL, siehe Abschnitt 3.2.6) ermittelt.

Der Client schickt dem Server dann eine Request-Anfrage in XML-Form zu. Das Beispiel in Listing 3.1 zeigt eine solche Nachricht am Beispiel einer Methode namens `getLocalTime`, die eine Zeitzone als String übergeben bekommt und die Zeit als String zurückgeben soll. Die dazugehörige Antwort des Servers könnte dann aussehen wie in Listing 3.2.

Dieses Beispiel zeigt auch den größten Nachteil von SOAP. Durch das Verpacken der eigentlichen Anfrage in XML-Code sowie die Berücksichtigung verschiedener Namespaces (mit deren Hilfe zum Beispiel die Definition eigener Typen möglich ist) wächst die Datenmenge stark an. So liegt das Verhältnis zwischen Auszeichnungsdaten und Nutzdaten im Beispiel bei ca. 25:1. Ist die Menge der übermittelten Nutzdaten größer (in diesem Projekt werden zum Beispiel komplette Dokumente ausgetauscht), verbessert sich das Verhältnis natürlich deutlich. Außerdem kann durch Kompression auf der darüberliegenden Protokollebene, zum Beispiel mit *GZip* bei HTTP, die Datenmen-

[1] http://www.w3.org

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getLocalTimeResponse
          soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:ns1="urn:TimeService">
   <getLocalTimeReturn
          xsi:type=''xsd:string''>2007-03-02 17:45</getLocalTimeReturn>
  </ns1:getLocalTimeResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.2: SOAP-Antwort vom Server

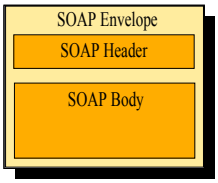
ge aufgrund der XML-typischen Redundanzen erheblich reduziert werden - allerdings auf Kosten der Rechenzeit, die für das Ver- und Entpacken der Informationen benötigt wird. Ob SOAP für die jeweilige Aufgabe das passende Mittel ist, muss also im Einzelfall abgewägt werden.

3.2.2 Datentypen

SOAP definiert bereits eine große Menge an Datentypen, darunter zum Beispiel `xsd:int`, `xsd:float` und `xsd:string`, aber auch komplexere wie `xsd:date`. Wie die jeweiligen Typen in die eigene Programmiersprache umgesetzt werden, bleibt der jeweils verwendeten Bibliothek überlassen.

Zusätzlich lassen sich eigene Typen definieren und verwenden. Zum einen gibt es die aus C bekannten Structs, also eine Zusammenstellung von verschiedenen, bereits definierten Datentypen, zum anderen Arrays, also eine Folge unbestimmter Länge von Werten des gleichen Datentyps. Auch hier gilt, dass es der jeweiligen Bibliothek überlassen bleibt, wie diese Datentypen in der verwendeten Programmiersprache umgesetzt werden.

3.2.3 Aufbau einer SOAP-Nachricht



Eine SOAP-Nachricht ähnelt im Aufbau einem Brief. Sie besteht aus einem Umschlag (*Envelope*), der einen Kopfbereich (*Header*, optional) sowie einen Textkörper (*Body*) enthält. Der Header kann Metainformationen wie zum Beispiel verwendete Verschlüsselungsalgorithmen oder eine Transaktionsnummer beinhalten. Im Body folgen dann Funktionsaufrufe, Fehlermeldungen oder reine Daten.

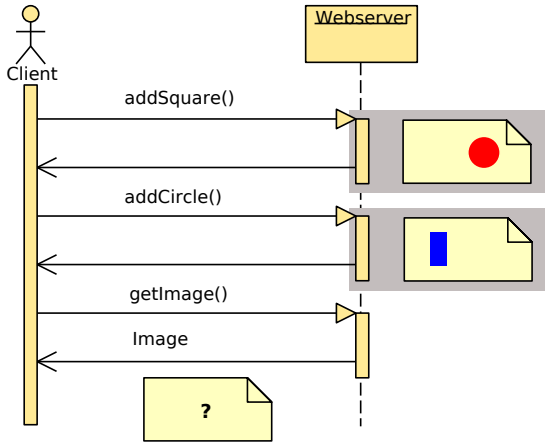


Abbildung 3.1: SOAP ohne Persistenz

3.2.4 Persistenz

SOAP ist ein zustandsloses Protokoll, bei dem für jede Anfrage eine neue Verbindung aufgebaut wird. Dies folgt aus der Unbestimmtheit des Übertragungsprotokolls, denn andernfalls würde die Technik mit zustandslosen Protokollen wie HTTP nicht funktionieren. Oft ist es jedoch nötig, die Daten auf der Serverseite persistent zu halten.

Um das zu veranschaulichen, stellen wir uns einen Webservice vor, dem nacheinander verschiedene Grafikobjekte übergeben werden und der aus diesen Informationen ein Bild erzeugen soll.

Dies ist ohne Persistenz nicht möglich, da bei jedem Aufruf nur die Daten der aktuellen Anfrage vorhanden sind (Abbildung 3.1).

Um das zu Umgehen könnte man bei jeder Anfrage das vorherige Bild mit übergeben und das neue zurückgeben zu lassen. Eleganter ist es jedoch, das unfertige Bild auf dem Server mit einer Session-ID speichern und nur diese ID bei jeder Anfrage zu übergeben (Abbildung 3.2).

Auf diese Weise wird es möglich, eine zustandsbehaftete Verbindung über ein zustandsloses Trägerprotokoll zu simulieren. Die Programmiersprache PHP, die häufig für serverseitige Programme eingesetzt wird, unterstützt dies mit einem eigenen Session-Mechanismus, der komplette Datenstrukturen und Objekte vorübergehend auf die Festplatte schreiben kann.

3.2.5 Übertragung binärer Daten

XML ist für die Übertragung von normalem, von Menschen lesbaren Text ausgelegt, nicht jedoch für das Versenden von binären Daten. Deshalb müssen solche Daten neu codiert werden, zum Beispiel mit dem *Base64*-Algorithmus. Dieses, noch aus der Zeit

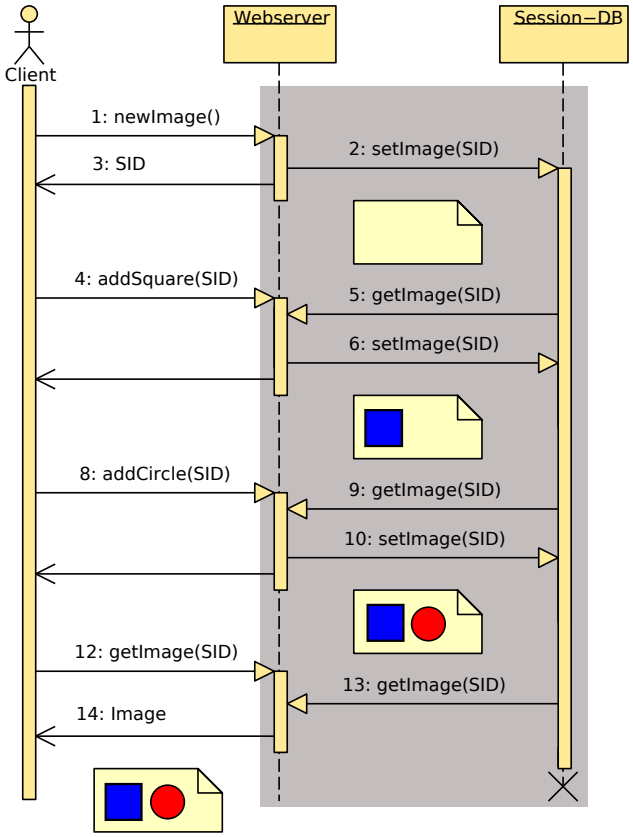


Abbildung 3.2: SOAP mit Persistenz

der 7-Bit-E-Mails stammende Protokoll verpackt binäre Daten in die 64 Standardzeichen A-Z, a-z, 0-9, + und /. Das Format nutzt also pro Zeichen genau 6 Bit, und so bleiben bei jedem Byte 2 Bit ungenutzt. Oder anders ausgedrückt: Die versendeten Datenmengen wachsen um ca. 33% an.

Deshalb werden binäre Daten wie Bilder oder Programme in diesem Projekt nicht als SOAP-Nachricht übertragen. Statt dessen wird bei einer entsprechenden Anfrage eine URL zurückgegeben, über die der Datenaustausch dann auf HTTP-Ebene stattfinden kann. Der zusätzliche Aufwand einer zweiten Anfrage ist aufgrund der eingesparte Datenmengen und der Möglichkeit eine Übertragung zu unterbrechen und später neu aufzunehmen, zu vernachlässigen.

3.2.6 WSDL

Die *Web Services Description Language* (WSDL) ist eine plattform- und sprachunabhängige XML-Spezifikation zur Beschreibung von Netzwerkdiensten wie SOAP. Ähnlich wie eine C-Header-Datei oder ein Java-Interface enthält die WSDL-Datei eine Beschreibung der von einem Webservice angebotenen Methoden und der verwendeten Datenstrukturen.

Aus diesen Informationen kann ein Programmierer oder ein Hilfsprogramm die zur jeweils verwendeten Programmmiersprache passenden Zugriffsfunktionen generieren. Im Idealfall kommt der Programmierer dann mit dem SOAP-Protokoll nicht weiter in Berührung sondern kann völlig transparent auf die erzeugten Daten zugreifen.

Die WSDL-Datei für den in Abschnitt 3.2.1 vorgestellten Zeitdienst könnte zum Beispiel aussehen wie in Listing 3.3.

Mit dieser Datei kann das Hilfsprogramm `wsdl2java` aus dem Apache Axis-Projekt Stellvertreterklassen erzeugen, deren Anwendung in Listing 3.4 demonstriert ist.

3.2.7 SOAP mit PHP (PEAR::SOAP)

Für die weitverbreitete Scriptsprache PHP existieren zwei wichtige Bibliotheken, die das Erstellen von SOAP-Diensten und den Zugriff auf diese erleichtern. Die ältere von beiden entstammt der *PEAR-Softwaresammlung*[2] und ist vollständig in PHP geschrieben. Seit PHP Version 5.0 gibt es auch eine in C geschriebene Bibliothek, die direkt in die API des Interpreters integriert ist.

Für dieses Projekt kam nur die PEAR-Bibliothek in Frage. Zum einen funktioniert sie, genau wie PmWiki selbst, auch noch mit PHP Version 4, zum anderen unterstützt sie die automatische Generierung einer WSDL-Datei aus den gegebenen Methoden.

Auf die Verwendung der Bibliothek für die Erweiterung von PmWiki gehe ich in Kapitel 4 genauer ein.

[2] `http://pear.php.net`

```
<?xml version="1.0"?>
<definitions>
  <message name="getLocalTimeRequest">
    <part name="timezone" type="xsd:string"/>
  </message>
  <message name="getLocalTimeResponse">
    <part name="return" type="xsd:string"/>
  </message>
  <portType name="timeservicePort">
    <operation name="getLocalTime">
      <input message="tns:getLocalTimeRequest"/>
      <output message="tns:getLocalTimeResponse"/>
    </operation>
  </portType>
  <binding name="timeserviceBinding" type="tns:timeservicePort">
    <soap:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getLocalTime">
    </operation>
  </binding>
  <service name="timeserviceService">
    <port name="timeservicePort" binding="tns:timeserviceBinding">
      <soap:address location="http://some.domain/timeservice.php"/>
    </port>
  </service>
</definitions>
```

Listing 3.3: WSDL-Datei für einen Zeitservice (gekürzt)

```
// Zugriff auf den von Axis erzeugte Stellvertreter (Stub)
TimeServicePort port = (new TimeServiceServiceLocator()).getTimeServicePort();
System.out.println(port.getLocalTime("GMT+01"));
```

Listing 3.4: Zugriff auf den Service aus Java mit Apache Axis

3.2.8 SOAP mit Java (Apache Axis)

Für Java gibt es eine vielzahl von Bibliotheken. Die bekannteste ist sicherlich *Axis*[3] aus dem *Apache Project*[4]. Sie bietet die Möglichkeit, vollautomatisch einsetzbare Stellvertreter-Klassen aus einer gegebenen WSDL-Datei zu erzeugen und stellt dadurch eine nicht zu unterschätzende Arbeitserleichterung dar.

[3] http://ws.apache.org/axis/
[4] http://www.apache.org

4 Ein SOAP-Plugin für PmWiki

Trotz der Vielzahl an verfügbaren Erweiterungen für PmWiki existiert bis heute noch kein veröffentlichtes Plugin, welches den Zugriff auf PmWiki als Webservice erlaubt. Der erste Schritt war also, die Plugin-Architektur und Funktionsweise von PmWiki zu erforschen und eine entsprechende Erweiterung zu erstellen. Zwar gibt es wenig Dokumentation zu diesem Thema (frei nach dem Motto "Der beste Quelltext dokumentiert sich selbst"), aber die Quelltexte der vorhandenen Erweiterungen halfen, das System schnell zu verstehen.

4.1 Die Funktionsweise von PmWiki

PmWiki ist prozedural programmiert und verzichtet fast vollständig auf den Einsatz von Klassen. Das ist verständlich, da das Konzept der Objekt-Orientierung in PHP Version 4 nur rudimentär implementiert war und erst mit Version 5 auf ein vernünftig einsetzbares Niveau erweitert worden ist.

Das System ist gruppenbasiert, jede Seite gehört also immer zu einer Gruppe. Die Standardgruppe heißt `Main`, die Standardseite `HomePage`. Der Name der aktuellen Seite wird über den Paramter `n=groupname.pagename` an die URL angehängt. Der zweite wichtige Parameter ist `action`, der den Ablauf steuert. Einige Aktionen kennt PmWiki von Haus aus:

Name	**Bedeutung**
browse	Seite betrachten (Standard)
edit	Seite bearbeiten
source	Quelltext betrachten
attr	Seitenattribute bearbeiten
logout	Abmelden

Daneben gibt es weitere mitgelieferte Aktionen, die aber erst in der Konfigurationsdatei `local/config.php` explizit aktiviert werden müssen, zum Beispiel `upload` für das Anhängen von Dateien. Der Programmcode für diese internen Erweiterungen befindet sich im Unterverzeichnis `source/`.

Was genau eine Aktion ausführen soll, definiert das globale assoziative Array `$HandleAuth`. Hier befinden sich Paare der Form `'action' => 'Funktionsname'`. PmWiki ruft nach der Initialisierung die zur übergebenen Aktion passende Funktion mit dem Seitennamen als Parameter auf.

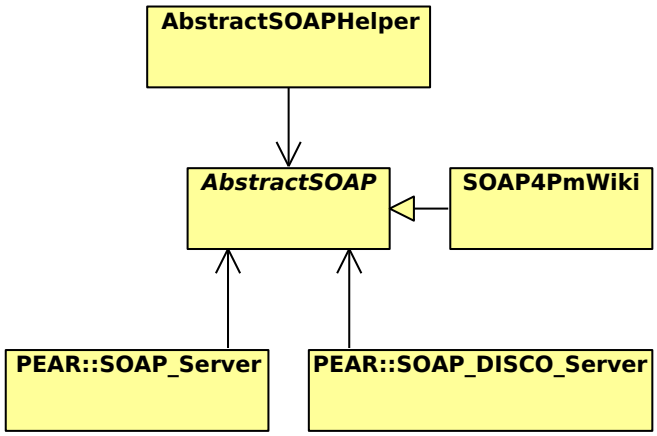


Abbildung 4.1: Klassendiagramm von Soap4PmWiki

Eigene Erweiterungen können im Unterverzeichnis `cookbook/` abgelegt werden, müssen aber in der Konfigurationsdatei explizit eingebunden werden. Hier kann zum Beispiel `$HandleAuth` um eine eigene Aktion erweitert werden.

4.2 soap4pmwiki

Das von mir erstellte Plugin hört auf den Namen *soap4pmwiki* und wird im gleichnamigen Unterverzeichnis im `cookbook`-Verzeichnis abgelegt. Nachdem es über die Konfigurationsdatei eingebunden worden ist, registriert es die Methode `HandleSoap` für eine neue Aktion namens `soap`. Das Plugin wird also immer dann aktiv, wenn PmWiki mit dem URL-Parameter `action=soap` aufgerufen wird.

Nach dem Aufruf der Funktion wird eine neue Instanz der Klasse `Soap4PmWiki` erzeugt, die URL des Wikis übergeben und die weitere Verarbeitung der von `AbstractSOAP` geerbten Funktion `handleRequest()` überlassen.

4.2.1 Erstellen von SOAP-Diensten

Das Erstellen von SOAP-Diensten mithilfe der Klasse `PEAR::SOAP_Server` ist zwar einfach, aber fehleranfällig. Man erstellt eine Klasse mit zwei festgelegten Feldern und übergibt diese einer Instanz der Klasse `SOAP_Server`. Das erste Feld ist ein mehrdimensionales assoziatives Array namens `$__typedef` und für die Definition eigener Datentypen zuständig. Dabei muss auf die strenge Verwendung des passenden Namespaces geachtet werden. Das andere Feld ist ebenfalls ein Array und hört auf den Namen `$__dispatch_map`. Hier werden die nach außen zur Verfügung gestellten Methoden, deren Parameter und Rückgabewerte definiert. Auch hier muss immer auf die Verwendung des passenden Namespaces geachtet werden (Listing 4.1).

Um das zu Vereinfachen habe ich die Klassen `AbstractSOAP` und

```
class TimeService {
    // Deklaration einer Liste von Zeitstempeln
    var $__typedef = array(
        'TimeData' => array(
             'timezone' => 'string',
             'timestamp' => 'long'
         ),
        'ArrayOfTimeDiff' => array(
            array('item' => '{urn:timeservice}TimeData')
        )
    );

    // Einbindung einer Methode
    var $__dispatch_map = array(
        'getWorldTimes' => array(
            'in' => array(
                // No arguments
            ),
            'out' => array(
                'return' => '{urn:timeservice}ArrayOfTimeData'
            )
        )
    );

    function getWorldTimes() {
        // Liefert eine Liste der aktuellen Uhrzeiten auf der Welt zurueck
    }

}
```

Listing 4.1: Verwendung von PEAR::SOAP

`AbstractSOAPHelper` geschrieben, die auch unabhängig von diesem Projekt für andere Webservices verwendet werden können. `AbstractSOAP` kapselt den Zugriff auf die Arrays `$__typedef` und `$__dispatch_map` in Methoden, die den korrekten Aufbau und die Verwendung des richtigen Namensraum sicherstellen (Listing 4.2). Diese Methoden sind:

`registerStruct($name, $fields)`
: Registriert einen neuen Struct-Typen, der aus den im assoziativem Array `$fields` übergebenen Feldern besteht.

`registerArray($name, $type)`
: Registriert ein neues Array mit Werten vom Typ `$type`.

`registerMethod($name, $arguments, $result)`
: Registriert eine neue zu exportierende Methode. Im Array `$arguments` befindet sich eine Liste der benötigten Funktionsparamter, der Typ des Rückgabewertes wird in `$result` definiert. Hat die Funktion keine Paramter oder liefert sie kein Ergebnis zurück dürfen die entsprechenden Argumente leer bleiben.

Auf die Angabe des Namespace darf bei der Verwendung dieser Methoden verzichtet werden, da sie den jeweils verwendeten Namensraum selbstständig ergänzen. Das funktioniert allerdings nur, wenn es keine Konflikte zwischen den durch SOAP defi-

```
class TimeService {

    function TimeService($url) {
        // Namespace bekanntgeben
        parent::AbstractSOAP($url, 'timeservice');

        // Datentype TimeData bekannt geben
        parent::registerType(
            'TimeData',
            array(
                'timezone' => 'string',
                'timestamp' => 'long'
            )
        );

        // Datentyp ArrayOfTypeData registrieren
        parent::registerArray('ArrayOfTimeData', 'TimeData');

        // Methode getWorldTimes veroeffentlichen
        parent::registerMethod('getWorldTimes', '', 'ArrayOfTimeData');

    }

    function getWorldTimes() {
        // Liefert eine Liste der aktuellen Uhrzeiten auf der Welt zurueck
    }

}
```

Listing 4.2: Verwendung von AbstractSOAP

nierten Datentypen und den eigenen gibt. Ansonsten wird ein entsprechender Fehler erzeugt und das Programm beendet.

Eine Klasse, die auf `AbstractSOAP` aufsetzt, sollte von ihr erben und in ihrem Konstruktor die entsprechenden `register`-Methoden aufrufen. Sobald alle vorbereitenden Maßnahmen getroffen worden sind, kann die Methode `AbstractSOAP::handleRequest` aufgerufen werden, die die weitere Verarbeitung übernimmt.

Handelt es sich bei der aktuellen Anfrage um einen *HTTP-POST*-Request, wird mithilfe der Dispatch-Map eine neue Klasse von `AbstractSOAPHelper` abgeleitet, instanziiert und `PEAR::SOAP_Server` zur Verarbeitung übergeben.

Falls es sich um einen *HTTP-GET*-Request handelt, wie er zum Beispiel von einem Webbrowser nach Eingabe einer URL erzeugt wird, sucht `handleRequest` nach einem URL-Parameter namens `wsdl`. Existiert dieser, wird die Klasse `PEAR::SOAP_Disco_Server` bemüht, um aus den in `$__dispatch_map` und `$__typedef` enthaltenen Informationen ein WSDL-Dokument zu erstellen.

Tritt weder der eine noch der andere Fall ein, wird die Methode beendet und das Programm normal fortgeführt.

4.2.1.1 Persistenz mit Sessions

Wie schon in Abschnitt 3.2.4 beschrieben, ist ein Session-Mechanismus nötig, um die Zustandslosigkeit von HTTP und SOAP zu überwinden. PHP stellt bereits die benötigte Funktionalität zur Verfügung. Um sie anwendbar zu machen, muss eine neue SOAP-Methode zur Erzeugung einer solchen Session erstellt und jede vorhandene Methode um einen zusätzlichen Parameter für die entsprechende Session-ID erweitert werden. In den jeweiligen Methoden muss dann diese ID der PHP-Funktion `session_id()` übergeben, die Funktion `session_start()` aufgerufen und alle verwendeten Variablen an der entsprechenden Session registriert werden.

Dies ist sehr umständlich, weswegen einem die Klasse `AbstractSOAPHelper` die komplette Arbeit abnimmt. Das einzige, was der Programmierer noch tun muss, ist das von `AbstractSOAP` geerbte Feld `$__use_session` auf `true` zu setzen.

`AbstractSOAP` übergibt dann nicht das eigentliche Objekt mit den registrierten Funktionen an `SOAP_Server`, sondern kapselt es in einer Instanz von `AbstractSOAPHelper`. Diese Klasse übernimmt die Dispatch-Map des originalen Objekts und erweitert jede Funktion um einen zusätzlichen Parameter für die Session-ID. Außerdem registriert sie eine neue Methode namens `getSessionID` zur Erzeugung einer solchen ID.

Beim Aufruf von `getSessionID` erzeugt `AbstractSOAPHelper` mithilfe der PHP-Funktionen eine neue Session und gibt dessen Session-ID zurück. Außerdem wird das gekapselte Objekt an der Session registriert, so dass alle Felder der Klasse persistent werden.

Sobald eine der veröffentlichten und mit der Session-ID erweiterten SOAP-Methoden aufgerufen wird, läd AbstractSOAPHelper mithilfe der übergeben ID die entsprechende Session, stellt die dort abgelegte Instanz des Originalobjektes wieder her und leitet den Aufruf - ohne die Session-ID - an die entsprechende Methode der Originalklasse weiter.

Dieses Verfahren erlaubt dem Programmierer eines Webservices eine völlig transparente Entwicklung. Nur bei der Verwendung globaler Variablen sollte er sich bewusst sein, dass sich diese zwischen zwei Aufrufen der Klassenmethoden geändert haben könnten, wenn er sie nicht explizit an der aktiven Session registriert.

4.2.1.2 Erweiterung eines SOAP-Dienstes

AbstractSOAP verfügt über eine öffentliche Methode namens `registerForeignMethod`. Damit kann eine nicht zum eigentlichen Objekt gehörende Methode an der SOAP-Klasse registriert werden. Auf diese Weise ist es möglich, die von der Klasse angebotenen Dienste ohne Modifikationen zu erweitern. Das Objekt, dem die fremde Methode gehört, wird mithilfe der Session ebenfalls persistent gemacht.

4.2.2 SOAP-Interface für PmWiki

Das eigentliche SOAP-Interface wird durch die Klasse Soap4PmWiki repräsentiert. Jede wichtige Funktion von PmWiki, wie zum Beispiel die Ausgabe von Seiten, ihren Quelltexten, der Versionsgeschichte und natürlich die Bearbeitung kann durch die entsprechenden Methoden von außen aufgerufen werden. Dabei wird auch die eventuell benötigte Authorisierung geprüft.

Die Klasse nutzt das Session-Feature von AbstractSOAP, daher hat jede Methode in der WSDL-Beschreibung noch einen Parameter für die Session-ID, der in der tatsächlichen Implementierung nicht vorkommt.

Die komplette Klasse wird in der API-Dokumentation beschrieben, die auf der beigelegten CD-ROM im Unterverzeichnis `soap4pmwiki/doc/` zu finden ist. Zusätzlich befindet sich unter `soap4pmwiki/soap4pmwiki.wsdl` die Schnittstellenbeschreibung im WSDL-Format.

4.3 Installation

Die Installation des Plugins in eine bestehende PmWiki-Installation ist denkbar einfach. Der Inhalt der ZIP-Datei (auf der CD-ROM im Verzeichnis `soap4pmwiki`) wird in das Verzeichnis `cookbook` der PmWiki-Installation entpackt. Danach wird die Konfigurationsdatei von PmWiki (normalerweise `local/config.php`) bearbeitet und um die folgende Zeile ergänzt:

```
include_once 'cookbook/soap4pmwiki/soap4pmwiki.php';
```

Schon ist die Installation abgeschlossen. Ob das Plugin funktioniert kann man feststellen, indem man die URL des Wikis im Browser um die Parameter `?action=soap&wsdl=1` erweitert.

Das Plugin Soap4PmWiki benötigt die Bibliothek `PEAR::SOAP`[1] sowie deren Abhängigkeiten im Suchpfad des PHP-Interpreters. Die Installation dieser Pakete ist distributionsabhängig.

4.4 Zusammenarbeit mit media2mult

Für die Zusammenarbeit mit anderen PmWiki-Erweiterungen, insbesondere media2mult, sind zwei Schnittstellen vorgesehen. Zum einen können zusätzliche Methoden mit `AbstractSOAP::registerForeignMethod` an der SOAP-Schnittstelle registriert werden. Denkbar ist etwa eine Methode zum Export einer bestimmten Seite in ein beliebiges Zielformat.

[1] http://pear.php.net/package/SOAP/

Die zweite Schnittstelle ist die Funktion `Soap4PmWiki::addFeature`. Hier können installierte Plugins ihren Namen sowie ihre Versionsnummer bekannt geben, so dass ein Dienst von außen erkennen kann, ob es bestimmte Funktionen des Wikis überhaupt benutzen kann. Hier wäre zum Beispiel ein Editor mit Syntax-Highlighting für die speziellen media2mult-Markup-Tags möglich, welches nur genutzt wird, wenn auf der Zielplattform media2mult installiert ist.

5 Eclipse

5.1 Die Entwicklungsumgebung...

Wenn Programmierer von *Eclipse*[1] reden, dann meinen sie zumeist das *Eclipse SDK*, die ursprünglich von IBM entwickelte Entwicklungsumgebung für Java. Tatsächlich hat das Eclipse SDK in diesem Bereich alle anderen Plattformen hinter sich gelassen. Dies liegt sicher nicht zuletzt auch daran, dass IBM recht früh beschlossen hat, den Quelltext des Programms unter eine offene Lizenz zu stellen und die Gemeinschaft mitentwickeln zu lassen - Entwickler wissen vermutlich am besten, was Entwickler für ihre Arbeit brauchen.

5.2 ...die mehr kann

Doch spätestens seit Version 3.0 ist Eclipse deutlich mehr als nur eine Entwicklungsumgebung. Oder besser gesagt: Deutlich weniger. Denn der Kern des Programms, der *Eclipse-Core*, besteht aus nicht viel mehr als einer Verwaltungsplattform für Plugins. Weitere Komponenten, wie die Oberfläche, die Menüs, das Framework und die Integration des Java Development Toolkits sind nichts anderes als Erweiterungen, die beim Start des Programms geladen werden.

Deshalb ist es zum einen sehr einfach, neue Plugins für das SDK zu schreiben, zum anderen aber auch möglich, völlig eigenständige Programme auf Basis der Eclipse-Plattform zu erstellen. Dadurch kommt man in den Genuss des breiten Pools von Eclipse-Plugins, die bereits viele Grundfunktionalitäten, wie Oberflächenaufbau und Updatemechanismus, abdecken. Darüber hinaus kann man seine Anwendung ebenso einfach um Einstiegspunkte für andere Plugins, sogenannte *Extension Points* (Abschnitt 5.2.1.3), erweitern.

Über die Plugin-Entwicklung für Eclipse kann man ganze Bücher schreiben ([3], [4] und [5]). Deshalb möchte ich dieses Thema nur soweit anschneiden, wie es für das Verständnis der hier entwickelten Erweiterung nötig ist.

[1] http://www.eclipse.org

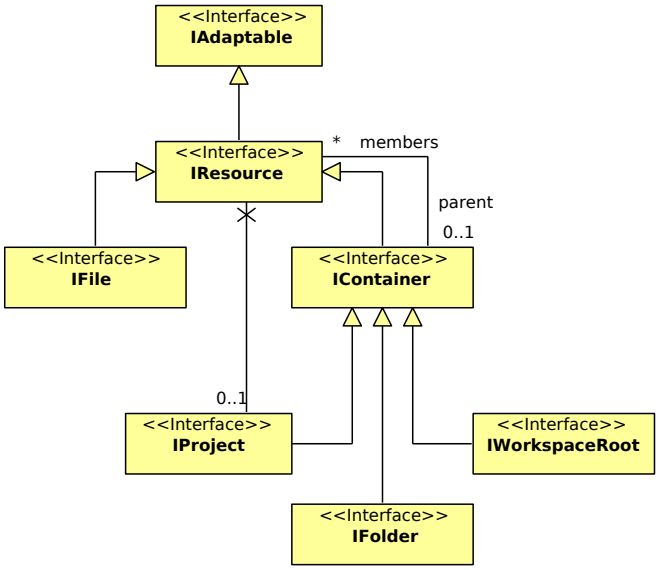


Abbildung 5.1: IResource und dessen Nachfahren

5.2.1 Plugin-Entwicklung

In den Versionen vor Eclipse 3.0 war das Programm in erster Linie eine Java-Entwicklungsumgebung, die sich mithilfe von Plugins erweitern ließ. Während der Entwicklung von Eclipse 3.0 verfolgte man dann den Ansatz “Alles ist ein Plugin” und modularisierte die vorhandenen Bestandteile so sehr, dass der eigentliche Kern nur noch die installierten Plugins läd, ihre Abhängigkeiten auflöst und sich um das Auslösen der Module an den entsprechenden Einstiegspunkten kümmert.

Der Rest wurde in Plugins ausgelagert, die auf Namen wie `org.eclipse.ui` oder `org.eclipse.jdt` hören. Die meisten dieser Plugins, wie zum Beispiel auch das JDT, sind wiederum in ein Core-Plugin für die Programmlogik (Model) und ein UI-Plugin für die grafische Darstellung und Menüs (View) aufgeteilt.

An ein Eclipse-Plugin werden nicht viele Anforderungen gestellt. Im Prinzip muss es nur zwei Dateien zu Verfügung stellen, die `plugin.xml` und ein Manifest (`META-INF/MANIFEST.MF`). Diese Dateien enthalten Informationen über den Namen des Plugins, seine Entwickler, die benötigten Abhängigkeiten (also andere Plugins, die installiert sein müssen), die zur Verfügung gestellten *Extension Points* und *Extensions*, und viele weitere Metadaten.

5.2.1.1 Resourcen

Das Plugin `org.eclipse.core.resources` stellt Klassen zur Verfügung, mit denen auf die Resourcen des *Workspace*, des aktuellen Arbeitsraumes, zugegriffen werden kann. Eine Resource (`IResource`) kann dabei der aktuelle Workspace (`IWorkspaceRoot`), ein Projekt (`IProject`), ein Verzeichnis (`IFolder`) oder eine Da-

tei (`IFile`) sein (Abbildung 5.1).

Den Workspace erhält man mit der Methode `ResourcesPlugin.getWorkspace().getRoot()`, von dort aus kann man bei Bedarf über die entsprechenden Kindobjekte das gewünschte Zielobjekt erreichen.

5.2.1.2 Naturen und Builder

Im Eclipse SDK wird man eine vielzahl von möglichen Projektarten finden. Jedes dieser Projekte wird durch das gleiche Interface `IProject` identifiziert.

Die Unterscheidung der Projektarten findet durch *Projektnaturen* statt. Dies sind Klassen, die das Interface `IProjectNature` implementieren und am Extension Point `org.eclipse.ui.ide.projectNatureImages` registriert worden sind. Sie werden in der Regel während der Erstellung eines neuen Projekts diesem hinzugefügt. Ein Projekt kann grundsätzlich beliebig viele Naturen enthalten, allerdings können die Naturen Abhängigkeiten und Ausschlusskritierien definieren. Die hier verwendete Natur `PmWikiNature` besteht zum Beispiel darauf, die einzige aktive Natur eines Projekts zu sein.

Eine andere Besonderheit, die der SDK-Vergangenheit von Eclipse geschuldet ist, sind die Builders. Diese Klassen werden am Extension Point `org.eclipse.core.resources.builders` angemeldet und bei Änderungen an geöffneten Resourcen aufgerufen, wie beispielsweise nach dem Bearbeiten und Speichern einer Datei. Im JDT-Plugin werden Builder verwendet, um eine Java-Klasse unmittelbar nach einer Modifikation zu kompilieren und eventuelle Fehler im Quelltext anzeigen zu können.

Dieses Projekt verzichtet auf eine eigene Builder-Klasse. Ein sinnvoller Einsatz ist aber dennoch vorstellbar und in Kapitel 7 beschrieben.

5.2.1.3 Extension Points und Extensions

Extension Points, Erweiterungspunkte, sind Stellen im Programmcode, an denen ein anderes Plugin aufgerufen werden kann. Die für eine konkrete Erweiterung benötigten Parameter (zum Beispiel ein Interface, welches von der registrierten Klasse implementiert werden muss), werden in der Datei `plugin.xml` bekannt gegeben.

Ein Plugin, dass einen Extension Point nutzen will, registriert dann eine entsprechende *Extension* in seiner eigenen `plugin.xml`. Die Verknüpfung zwischen Extension Points und zugehörigen Extensions stellt der Eclipse-Kern sicher.

Diese Technik wird für die größten Teil der Plugin-Kommunikation genutzt, zum Beispiel für das Einhängen von Menüpunkten, Assistenten oder eigenen Editoren, aber auch für das Hinzufügen von sogenannten Projektnaturen.

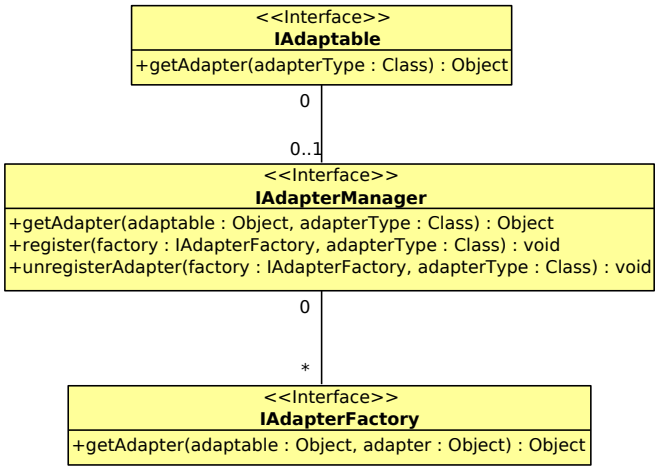


Abbildung 5.2: Adapter-Klassen in Eclipse

5.2.1.4 Adapter und AdapterFactory

Häufig will man Klassen anderer Plugins um eigene Funktionalitäten erweitern, zum Beispiel die `IFile`-Klasse für bestimmte Dateitypen um zusätzliche Methoden ergänzen. Natürlich kann man nicht einfach die Originalklasse bearbeiten (über deren Quelltexte man unter Umständen garnicht verfügt), und auch Vererbung ist keine Alternative, denn was machen dann andere Plugins, die Klasse ebenfalls erweitern möchten?

Die Eclipse-Lösung für dieses Problem ist das *Adapter-Pattern* (Abbildung 5.2). Dieses wird so sehr ausgereizt, dass die entsprechenden Klassen und Interfaces bereits vom Grundplugin `org.eclipse.core` zur Verfügung gestellt werden. Dazu fragt man ein Objekt, welches `IAdaptable` implementiert (zum Beispiel `IFile`) nach der Instanz eines bestimmten Interface oder einer bestimmten Klasse. Innerhalb des selben Plugins wird diese Anfrage häufig direkt von dem entsprechenden Objekt verarbeitet, ansonsten in der Regel an den `AdapterManager` von Eclipse weitergeleitet.

Diese dem *Delegation Pattern* nachempfundene Klasse prüft, ob ein oder mehrere Plugins für den zu adaptierenden Typen am Extension Point `org.eclipse.core.runtime.adapters` eine `IAdapterFactory` registriert haben und leitet die Anfrage nach einem passenden Adapter an alle Factories solange weiter, bis er eine Instanz des Adapters erhält. Dieser Mechanismus erlaubt, eine saubere Trennung zwischen den verschiedenen Plugins zu gewährleisten und dennoch Erweiterungen zu ermöglichen.

Bei dem PmWiki-Client-Plugin wurde der `AdapterManager` genutzt, um die Programmlogik für Projekte, Gruppen und Seiten auf die jeweiligen Instanzen von `IProject`, `IFolder` und `IFile` abzubilden.

5.2.2 Rich Client Plattform (RCP)

Wie erwähnt ist es auch möglich, Eclipse als Plattform für völlig eigenständige Programme zu nutzen und dabei die Vorteile der vorhandenen Bibliotheken wie SWT, JFace oder dem Update-Mechanismus zu nutzen. In dem Fall dient Eclipse als *Rich Client Plattform*. Letztendlich handelt es sich bei einem RCP-Programm um ein oder mehrere Eclipse-Plugins, die mit dem Eclipse-Core und einiger weiterer Abhängigkeiten gebundelt und als eigenständig ausführbares Programm vertrieben werden. Es ist möglich, Eclipse soweit anzupassen (*Branding*), dass das exportierte Programm kaum noch hinweise auf seinen Ursprung hinterlässt, zum Beispiel durch Andern des Programmnamens- und Icons sowie des Splashscreens, durch die Einbindung eigener Menüs und eigener Hilfen und vielem mehr. Das erstellte Programm muss dabei nichtmal im entferntesten etwas mit einer Entwicklungsumgebung (IDE) zu tun haben, auch kaufmännische Büroanwendungen oder ein Grafikprogramm sind denkbar.

Das hier entwickelte Plugin lässt sich optional ebenfalls als eigenständige RCP-Anwendung starten (Abschnitt 6.4). Der Vorteil ist die geringere Startzeit im Gegensatz zur vollständigen Java-IDE, da viele Plugins einfach weggelassen werden können. Tatsächlich verwendet es nur wenig mehr als die IDE-Basis-Bibliothek `org.eclipse.ide.core` und Bibliotheken für die Team-Unterstützung.

6 PmWiki-Client für Eclipse

Ursprünglich war die Entwicklung des Clients als eigenständige Anwendung geplant. Von Dr. Ralf Kunze kam der Tipp, die Anwendung als Plugin für Eclipse zu entwickeln, so dass man vorhandene Resourcen (wie zum Beispiel den Vergleich zwischen verschiedenen Revisionen der gleichen Datei) nutzen könnte.

Nach einigen Recherchen in diesem Bereich entschloss ich mich, dieses Konzept zu übernehmen. Das entwickelte Plugin sowie dessen Quelltexte befindet sich im Unterverzeichnis `pmwikiclient/` auf der beigelegten CD-ROM.

6.1 Nutzung vorhandener Resourcen

6.1.1 Projekte, Verzeichnisse und Dateien

Das Plugin `org.eclipse.core.resources` stellt Klassen für den Umgang mit Projekten, Verzeichnissen und Dateien zur Verfügung. Diese können mithilfe von Adaptern um speziellere Programmlogik erweitert werden (Abschnitt 5.2.1 ff).

6.1.2 Die IDE-Oberfläche

Für PmWiki wurde eine neue Eclipse-Perspektive, das heißt eine neue Darstellung der Programmoberfläche, eingeführt. Sie kann über den Menüpunkt WINDOW » OPEN PERSPECTIVE » PMWIKI aktiviert werden und blendet die Funktionen für die Java-Entwicklung aus. Statt dessen werden neue Buttons und ein neues Menü mit Funktionen für PmWiki angezeigt.

Abbildung 6.1 zeigt den Unterschied zwischen der Java- und PmWiki-Perspektive.

6.1.3 Das Team-Plugin

Die Klassen des Team-Plugins `org.eclipse.team.core` unterstützen die Verwaltung von Dateien, die in verschiedenen Revisionen vorliegen. Dies ist bei diesem Projekt der Fall, denn eine lokal gespeicherte Datei kann im Wiki bereits komplett anders aussehen, ohne dass der Anwender dies weiß. Es ist deshalb nötig, solche Änderungen mit eventuellen lokalen Modifikationen zu verschmelzen und den Anwender gegebenenfalls auf Konflikte hinzuweisen.

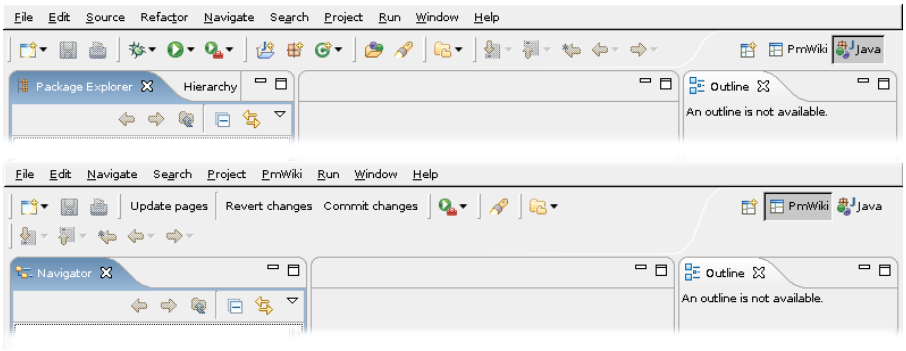


Abbildung 6.1: Java- und PmWiki-Perspektive

Dies wird durch die TEAM-PERSPEKTIVE, die alle Änderungen an den Seiten anzeigt und bearbeiten lässt, realisiert.

6.2 Erzeugung der SOAP-Schnittstelle

6.2.1 Apache Axis

Apache Axis ist eine mächtige Java-Bibliothek für den Zugriff auf SOAP-Dienste (Abschnitt 3.2.8).

6.2.1.1 wsdl2java

Mithilfe des Programms `wsdl2java` kann aus einer WSDL-Datei eine komplette Sammlung von Java-Klassen erzeugt werden, die dem Programmierer den Zugriff auf die SOAP-Schnittstelle abnehmen.

Dieses Tool habe ich auch für die Erzeugung der in im Package `de.rotapken.pmwiki.soap` liegenden Klassen genutzt.

6.2.1.2 Anpassung des erzeugten Codes

Da noch einige spezielle Anpassungen der automatisch generierten Klassen nötig waren, wurde der Aufruf von `wsdl2java` in ein Unix-Shell-Script verpackt. Dieses befindet sich in den Quelltexten des Plugins `de.rotapken.pmwiki` im Unterverzeichnis `tools` unter dem Namen `makewsdl.sh`. Das Programm wird mit der URL der WSDL-Datei, für die es Klassen erzeugen soll, aufgerufen.

Folgende Schritte werden durchgeführt:

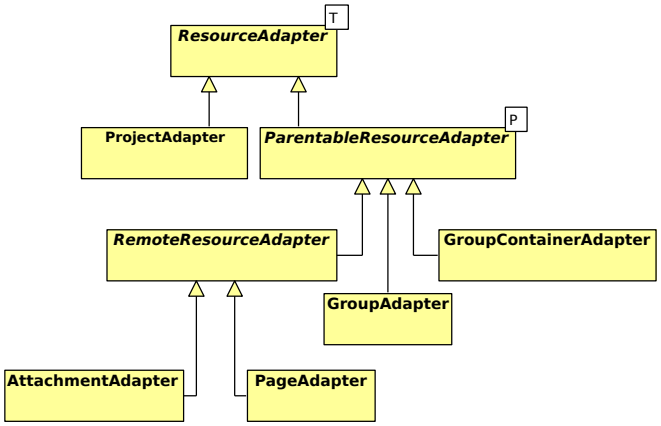


Abbildung 6.2: Vererbungshierarchie des PmWiki-Client-Models

1. Aufruf von `wsdl2java` mit der übergeben URL und dem Package `de.rotapken.pmwiki.soap` als Ausgabeverzeichnis

2. Umbenennung des erzeugten Interface `Soap4PmwikiPort_PortType` in das handlichere `Soap4Pmwiki`

3. Entfernung des Parameters `sessionID` aus den Methodenköpfen und Ersetzung durch ein implizites Session-Management

4. Die Methode `__getEndpointURL()` dem Interface `Soap4Pmwiki` hinzufügen

5. Korrektur der Rückgabe von leeren Arrays (statt `null` ein Array der Größe 0)

6. Anwendung aller weiteren Patches, die im Verzeichnis `tools/wsdl_patches` abgelegt sind.

Für diese Änderungen müssen die Unix-Programme `sed` und `patch` installiert sein. Der Aufruf des Programms sollte aber nur für die Entwicklung des Plugins, nicht für die Anwendung nötig sein.

6.3 Die Model- und Controller-Klassen

Die Abbildung der Seiten und Gruppen des Wikis auf lokale Objekte sowie der Datenaustausch mit dem Wiki ist über die Klassen des Package `de.rotapken.pmwiki.model` realisiert. Instanzen dieser Klassen können über das Adapter-Pattern von `IResource`-Objekten angefordert werden. Bildet eine Resource kein Objekt eines Wikis ab, so wird `null` zurückgegeben.

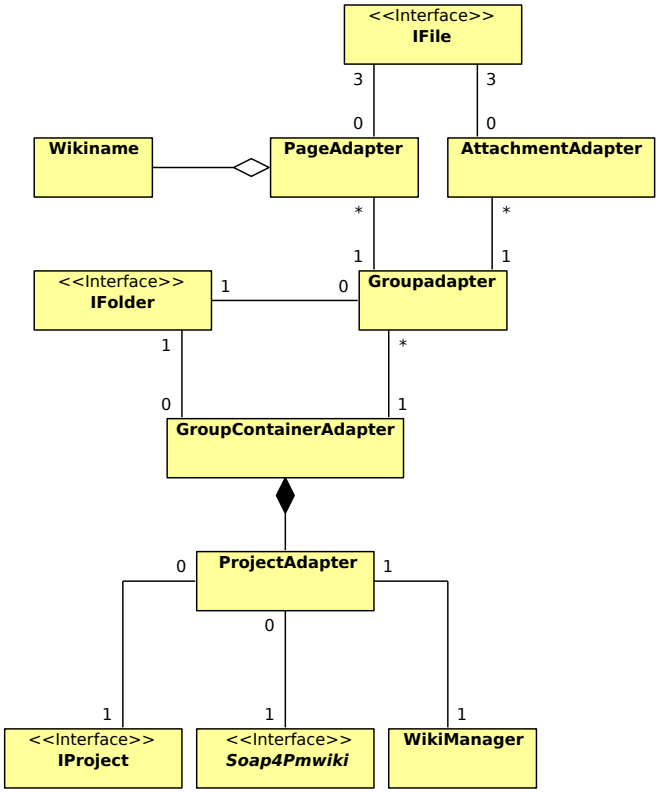


Abbildung 6.3: Assoziationsdiagramm des PmWiki-Client-Models

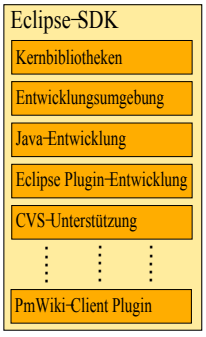


Abbildung 6.4: SDK mit Plugins

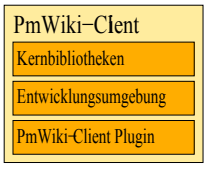


Abbildung 6.5: PmWiki-Client

6.4 Export als RCP-Anwendung

Das bis hierhin erstellte Plugin kann einfach in Eclipse installiert und zusammen mit anderen Plugins genutzt werden.

Das Zielpublikum von media2mult hat aber vermutlich nur eine kleine Schnittmenge mit den Java-Programmierern, die Eclipse benutzen. Für eine Person, die einfach nur media2mult oder PmWiki auf dem lokalen Rechner nutzen möchte, stellt es eine große Hürde dar, Eclipse zu installieren und dann noch ein Plugin einzubinden. Außerdem dürfte durch die große Menge an (nicht genutzer) Java-Funktionen eine Abschreckung durch Reizüberflutung zu erwarten sein.

Deshalb habe ich mit `de.rotapken.pmwiki.rcp` ein weiteres Plugin geschrieben, welches zusätzlich zu den Kernpaketen nur die Basis-Oberfläche der Entwicklungsumgebung und das PmWiki-Client-Plugin einbindet. Diese werden dann in aufgeräumter Form präsentiert und können vom Benutzer als eigenständige Anwendung installiert werden.

Intern besteht es im wesentlichen nur aus Initialisierungsklassen, die den Benutzer nach dem zu verwendeten Workspace fragen und das Layout der Entwicklungsumgebung mit aktivierter PmWiki-Perspektive konfigurieren. Außerdem fügt das Plugin einen eigenen Splash-Screen und eigene Programmicons hinzu.

Kompilierte Versionen für Windows und Linux befinden sich auf der beigelegten CD-ROM im Verzeichnis `pmwikiclient/rcp/`.

6.5 Installation und Anwendung

In den folgenden Abschnitten wird die Installation und Verwendung des PmWiki-Clients sowohl als RCP-Anwendung als auch als Eclipse-Plugin beschrieben. In jedem Fall muss auf dem System eine Java-Laufzeitumgebung (JRE) der Version 1.5 oder höher installiert sein, die man unter `http://java.sun.com` herunterladen kann.

6.5.1 Installation und Start der RCP-Version

Die Installation der RCP-Version des PmWiki-Clients ist einfach. Das Programm kommt in einer Archivdatei, die an einem beliebigen Ort entpackt werden muss (die Version auf der CD-ROM ist bereits entpackt und muss nur noch auf die Festplatte kopiert werden).

Zum Starten der Anwendung wird die entsprechende ausführbare Datei geöffnet. Unter Windows geschieht dies durch einen Doppelklick auf `pmwikiclient.exe`, unter Linux wird die Datei `pmwikiclient` aufgerufen.

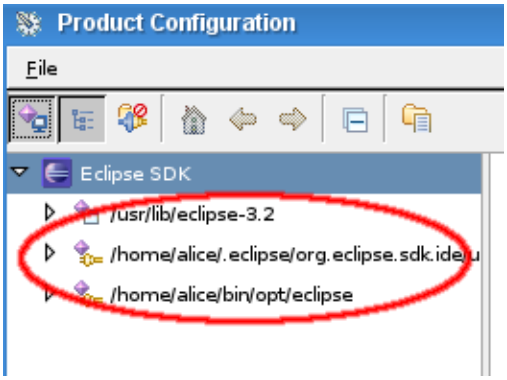


Abbildung 6.6: Installationspfade von Eclipse-Plugins

6.5.2 Installation des Eclipse-Plugins

Zur Installation unter Eclipse muss man den Installationsort der Plugins kennen. Diesen erfährt man, wenn man das Eclipse SDK startet und im Menü die Punkte HELP » SOFTWARE UPDATES » MANAGE CONFIGURATION auswählt.

Auf der linken Seite werden ein oder mehrere Pfade angezeigt (Abbildung 6.6). Einen von diesen öffnet man dann im Dateimanager seines Systems und entpackt die Archivdatei des Plugins an diesem Ort.

Beim nächsten Start von Eclipse sollte im Menü FILE » NEW » OTHER der neue Eintrag PMWIKI zu sehen sein.

6.5.3 Anlegen eines neuen Projekts

Nach der Installation startet man die RCP-Anwendung oder das Eclipse-SDK und ruft über FILE » NEW » PMWIKI » PMWIKI-CLIENT den Dialog zur Erstellung eines neuen PmWiki-Client-Projekts auf (Abbildung 6.7).

In diesem Dialog gibt man einen beliebigen Namen für das Projekt und die URL des Wikis ein, auf das man zugreifen möchte (also das gleiche, was man auch im Webbrowser eingeben würde, um die Seite aufzurufen). Nach der Eingabe der URL wird diese validiert, und, falls gültig, eine Liste der verfügbaren Gruppen angezeigt.

Hier wählt man die zu abonnierenden Gruppen aus und erhält die Gelegenheit, neue Gruppen anzulegen. Die Option CHECKOUT AFTER FINISH sollte aktiviert sein, wenn die Seiten sofort ausgelesen und auf den eigenen Rechner übertragen werden sollen. Ansonsten kann dies aber auch noch später geschehen, indem die jeweilige Gruppe ausgewählt und der Menüpunkt PMWIKI » UPDATE ausgewählt wird.

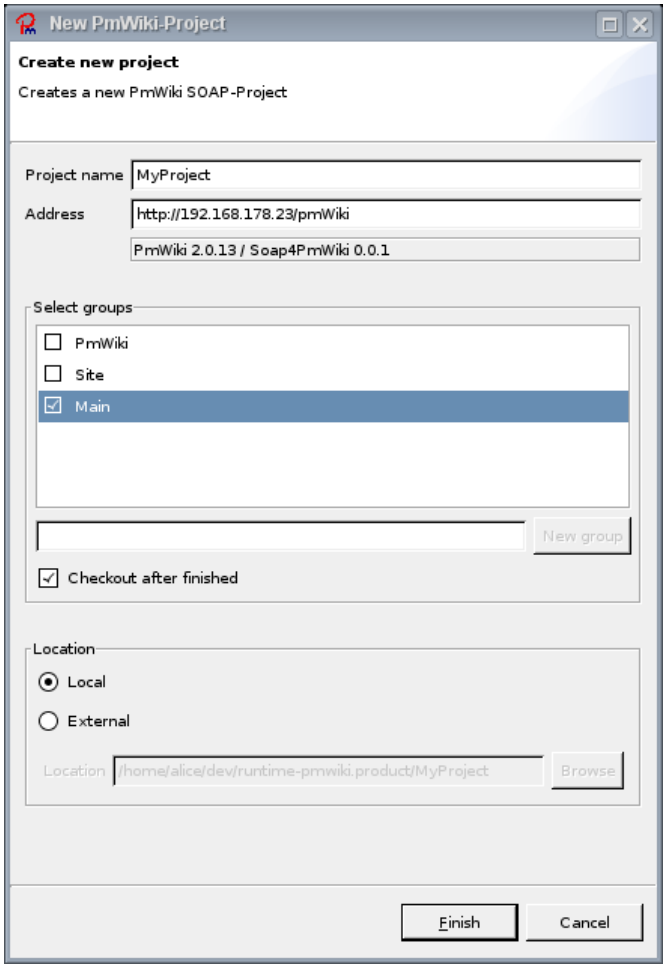


Abbildung 6.7: Neues PmWiki-Client-Projekt anlegen

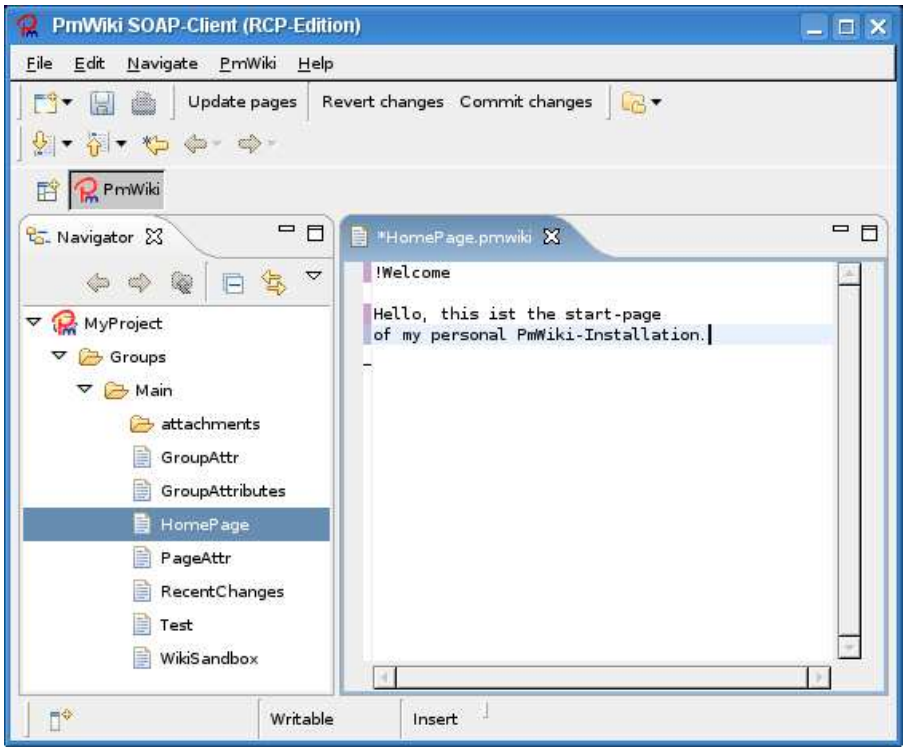


Abbildung 6.8: Quelltext einer Seite

6.5.4 Bearbeitung von Seiten

Die Seiten sind in Gruppen organisiert und werden als Text-Dateien angezeigt. Durch einen Doppelklick kann eine Seite im Editor bearbeitet werden (Abbildung 6.8). Auch das Löschen und Umbenennen von Dateien ist möglich, allerdings wird der neue Name nur lokal verwendet und nicht in das Wiki übertragen.

6.5.5 Aktualisierung der lokalen Dateien

Um Änderungen, die im Wiki vorgenommen worden sind, auf den eigenen Rechner zu übertragen, muss der Menüpunkt PMWIKI » UPDATE ausgewählt werden. Sofern eine Internetverbindung besteht und das Wiki erreichbar ist, werden die geänderten Seiten auf den lokalen Rechner übertragen.

In der TEAM-PERSPEKTIVE, die anschließend geöffnet wird, hat man Gelegenheit, diese Änderungen einzusehen und ggf. zu bearbeiten (Abbildung 6.9). Durch einen Klick auf das PmWiki-Symbol in der Perspektivenauswahl gelangt man zurück in die gewohnte Ansicht.

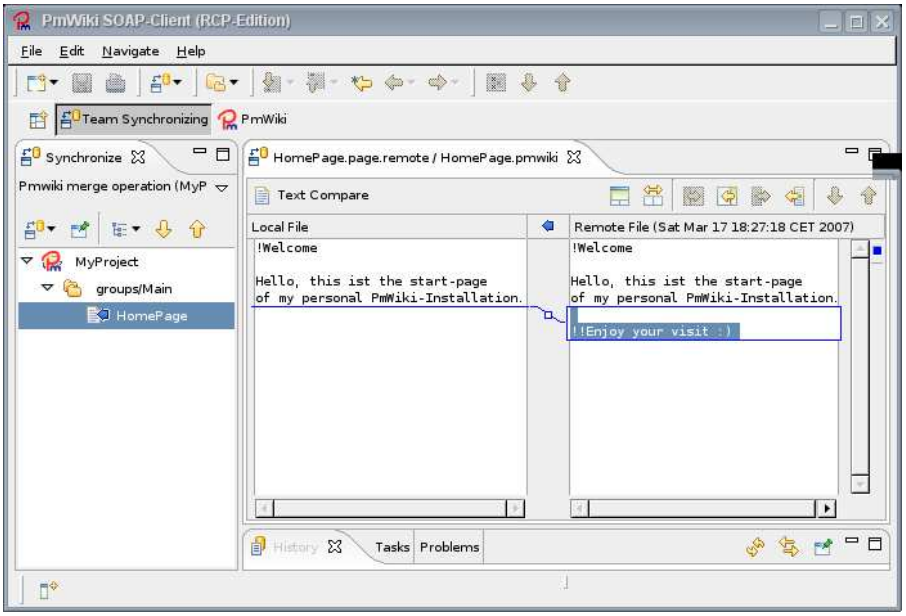


Abbildung 6.9: Aktualisierungsdialog

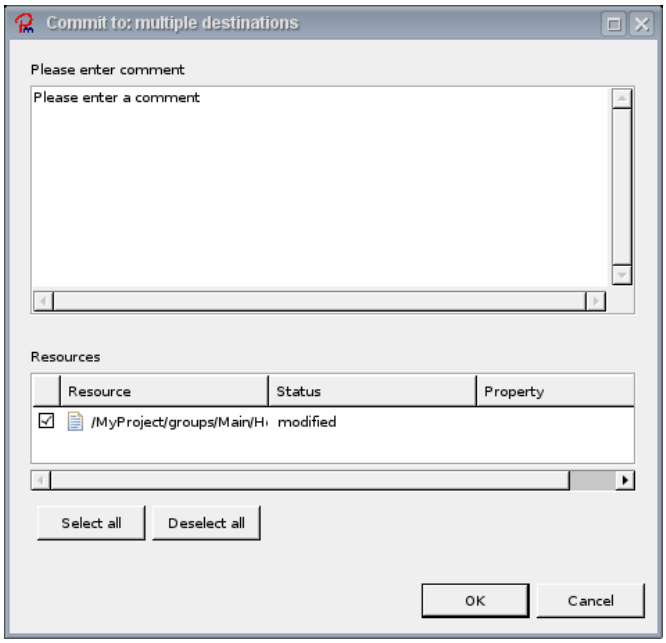


Abbildung 6.10: Änderungen übertragen

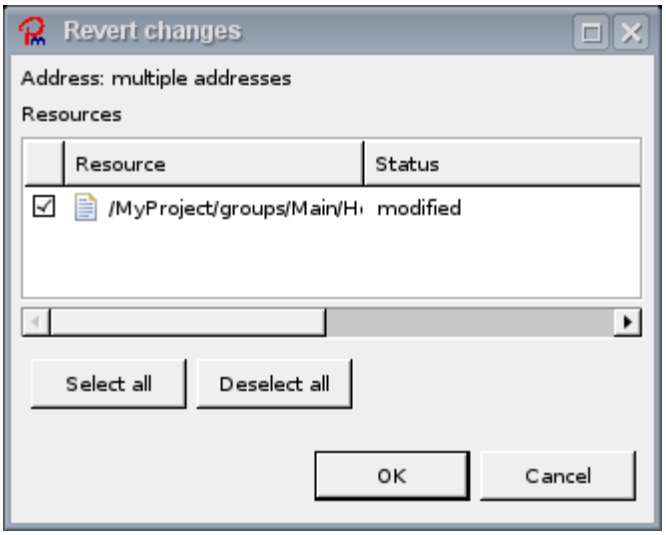


Abbildung 6.11: Änderungen widerrufen

6.5.6 Übertragung der lokalen Änderungen

Der Menüpunkt PmWiki » Commit zeigt die lokal geänderten Dateien an und bietet an, diese in das Wiki zu übertragen. Es ist sinnvoll, einen Kommentar einzugeben, der in der Versionsgeschichte der geänderten Seiten angezeigt wird (Abbildung 6.10).

6.5.7 Änderungen widerrufen

Möchte man eine lokal geänderte Datei wieder auf die Version im Wiki zurücksetzen, sollte man den Menüpunkt PmWiki » Revert bemühen. Wie im Commit-Dialog werden auch hier alle Änderungen angezeigt und können rückgängig gemacht werden (Abbildung 6.11).

6.5.8 Eine Seite im Webbrowser öffnen

Der PmWiki-Client kann eine Seite direkt im Webbrowser öffnen. Dazu muss zuvor **einmalig** ein Webbrowser konfiguriert worden sein. Dies geschieht im Menüpunkt Edit » Preferences » General » Web Browser.

Dann kann eine Seite im Webbrower geöffnet werden, wenn der Menüpunkt PmWiki » Goto PmWiki ausgewählt wird.

6.5.9 Falls mal etwas nicht funktioniert...

Die Menüpunkte funktionieren derzeit nur dann zuverlässig, wenn vorher eine entsprechende Resource (also ein Projekt, eine Gruppe oder eine Seite eines PmWiki-Client-Projekts) ausgewählt worden ist. Erfolgt keine Reaktion, sollte man probieren, den gewünschten Menüeintrag über das Kontextmenü der entsprechenden Datei bzw. des entsprechenden Verzeichnisses aufzurufen.

In jedem Fall ist eine direkte Internetverbindung erforderlich. Der Einsatz von Proxy-Servern ist momentan noch nicht möglich.

7 Epilog

7.1 Probleme während der Eclipse-Entwicklung

Ich möchte nicht verschweigen, dass die Entwicklung von Eclipse-Plugins dieser Art äußerst komplex ist und entgegen der ursprünglichen Planung den größten Anteil der Bearbeitungszeit in Anspruch nahmen. Trotz ca. 70(!) erstellter Klassen, die in das Plugin eingeflossen sind, konnten nur die Grundfunktionen fristgerecht fertiggestellt werden (zum Vergleich: Das PmWiki-Plugin *soap4pmwiki* kommt mit drei neuen Klassen aus). Der nächste Abschnitt stellt Ideen vor, für die die Zeit nicht mehr ausreichte, die aber in absehbarer Zeit nachgereicht werden sollen.

Problematisch war vor allem der hohe Abstraktionsgrad von Eclipse. Die strikte Trennung von Plugins, die darüber hinaus auch noch in öffentliche und private (*internal*) Bereiche aufgeteilt sind, sowie die scharfe Grenze zwischen Oberfläche und Programmlogik erzwingen vom Entwickler ein sauberes Arbeiten. Aus dem selben Grund ist die Lernkurve bei der Eclipse-Entwicklung aber auch äußerst steil.

Ärgerlicherweise kommt hinzu, dass große Teile des internen Aufbaus und der Funktionsweise von Eclipse und seinen Plugins ausschließlich durch die API-Dokumentation erklärt werden und es an brauchbaren Beispielen mangelt. Besonders das Team-Plugin ist hier negativ aufgefallen. Die entsprechenden Klassen konnte ich erst nach Rückfragen in der Eclipse-Newsgroup ([7]) und intensivem Reverse-Engineering der CVS- und Subversion-Plugins nutzen, was natürlich entsprechend Zeit brauchte.

Mein Fazit: Eclipse ist als Plattform für eigene Anwendungen extrem mächtig, erfordert aber eine lange Einarbeitungszeit. Idealerweise sollten solche Anwendungen im Team geschrieben werden, so dass nicht jeder Programmierer von Anfang an die Funktionsweise der gesamtem Plattform erlernen braucht. Eine Aufteilung in einzelne Gruppen für die Oberfläche, das Konfigurationsmenü, dem Hilfesystem und der Programmlogik ist sinnvoll.

7.2 Zukünftige Erweiterungen des Eclipse-Plugins

Eine ganze Reihe ursprünglich geplanter Funktionen hat es nicht mehr bis in das Endprodukt geschafft. Da diese aber teilweise essentiell sind, werde ich an dem Programm weiterarbeiten und diese Funktionen in absehbarer Zeit zur Verfügung stellen.

- Verwaltung von Dateianhängen
- Dialoge und Templates zum Erstellen neuer Gruppen und Seiten
- Builder, der die Vorschau einer Seite beim Speichern erzeugt
- Einstellungsseite zur Änderung der URL des Wikis
- Unterstützung für Proxy-Server
- Unterstützung für passwortgeschützte Seiten
- Eigener Editor mit Syntax-Highlighting
- Verbesserung des Update-Mechanismus bei Konflikten
- Hilfesystem
- Trennung des Plugins in Oberfläche und Programmlogik

7.3 Nutzung des Plugins für andere Wiki-Engines

Theoretisch wäre es denkbar, das Plugin auch für andere Wiki-Systeme (zum Beispiel Wikimedia) zu verwenden. Dies könnte entweder erreicht werden, indem für das System eine kompatible SOAP-Schnittstelle geschaffen wird. Sauberer ist es jedoch, die konkreten Wiki-Funktionen aus dem PmWiki-Client-Plugin auszulagern und über das Adapter-Pattern und eigene Extension-Points einzubinden, so dass man dieses Plugin für andere SOAP-Schnittstellen austauschen kann.

Rechtliches

PmWiki ist ein eingetragenes Markenzeichen von Patrick R. Michaud. Bitte besuchen Sie http://www.pmwiki.org für weitere Informationen.

Eclipse, *Build on Eclipse* und *Eclipse Ready* sind Markenzeichen der *Eclipse Foundation, Inc.*

Java ist ein eingetragenes Markenzeichen von *Sun Microsystems, Inc.*

Sämtliche Rechte an den hier verwendeten Marken verbleiben bei ihren Inhabern.

Lizenzierung

Die Programme *Soap4PmWiki* und *PmWiki-Client für Eclipse*, (c) Roland Tapken, sind freie Software. Sie können sie unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation veröffentlicht, weitergeben und/oder modifizieren, entweder gemäß Version 2 der Lizenz oder (nach Ihrer Wahl) jeder späteren Version.

Die Veröffentlichung dieses Programms erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber **ohne irgendeine Garantie**, sogar ohne die implizite Garantie der Marktreife oder der **Verwendbarkeit für einen bestimmten Zweck**. Details finden Sie in der GNU General Public License.

Sie sollten ein Exemplar der GNU General Public License zusammen mit diesem Programm erhalten haben. Falls nicht, schreiben Sie an die Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110, USA.

Den Originaltext dieser Lizenz finden Sie auf der beigelegten CD-ROM in der Datei `LICENSE.TXT` oder unter http://www.fsf.org/licenses/gpl.html.

Literaturverzeichnis

[1] Wikipedia - The Free Encyclopedia
http://www.wikipedia.org

[2] PmWiki-Homepage und -Dokumentation
http://www.pmwiki.org

[3] Berthold Daum, “Rich-Client-Entwicklung mit Eclipse 3.1”
dpunkt Verlag, 2005

[4] Berthold Daum, “Das Eclipse Codebuch”
dpunkt Verlag, 2006

[5] Kai Brüssau, Oliver Widder, “Eclipse - Die Plattform”
Software & Support-Verlag, 2005

[6] Torsten Langner, “Web Services mit Java”
Markt+Technik-Verlag, 2003

[7] Eclipse.org Newgroups
http://www.eclipse.org/newsgroups/

Abbildungsverzeichnis

Glossar

Branding ..... Anpassung eines Produkts an das eigene Design
Cross-Publishing Veröffentlichung für verschiedene Plattformen
dynamische Bindung Die tatsächlich ausgeführte Funktion steht zur Programmierzeit noch nicht fest
FTP ..... File Transfer Protocol, Übertragung von Dateien
GZip ..... Einfaches Kompressionsprogramm
HTTP ..... HypterText Markup Language
HTTPS ..... Verschlüsseltes HTTP
Java ..... Objektorientierte, plattformunabhängige Programmiersprache von Sun Microsystems, Inc.
PHP ..... Scriptsprache, hauptsächlich in der Webentwicklung eingesetzt
Rich Client Plattform Eine Plattform, die eine für eine vielzahl von Problemen fertige Lösungen anbietet
SMTP ..... Simple Mail Transfair Protocol, Standard zum E-Mail-Versand
TCP ..... Transmission Control Protocol, Netzprotokoll
UDP ..... User Datagram Protocol, verbindungsloses Netzprotokoll
W3C ..... Gremium zur Standardisierung das World Wide Web
Wiki ..... Seitensammlung, die vom Besucher geändert werden kann
XML ..... Extensible Markup Language, eine erweiterbare Auszeichnungssprache

Inhalt der CD-ROM

Dieser Arbeit ist eine CD-ROM mit ergänzenden Inhalten beigelegt.

- `documents/`
  Enthält dieses Dokument im PDF-Format und meinen Vortrag über media2mult im Seminar Webpublishing (WS05/06)
- `eclipse/`
  Enthält das Eclipse-SDK für ausgewählte Plattformen
- `pmwiki/`
  Enthält die aktuelle Version vom PmWiki
- `pmwikiclient/plugin`
  Die Plugin-Version des PmWiki-SOAP-Clients
- `pmwikiclient/rcp`
  Die RCP-Version des Clients für Windows und für Linux
- `pmwikiclient/doc`
  Die Klassendokumentation des Clients
- `soap4pmwiki/`
  Dieses Verzeichnis beinhaltet das SOAP-Plugin für PmWiki
- `soap4pmwiki/doc`
  Hier sind die WSDL und die API-Dokumentation des PmWiki-Plugins zu finden

Erklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbstständig angefertigt habe und keine Hilfsmittel außer denen in der Arbeit angegebenen benutzt habe.

Osnabrück, den 20. März 2007 .....

Roland Tapken