Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Stephan Wagner

Aufgabenorientierte Codegenerierung für webbasierte Benutzeroberflächen

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Stephan Wagner

Aufgabenorientierte Codegenerierung für webbasierte Benutzeroberflächen

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Bernd Owsnicki

Abgegeben am 28. Januar 2004

Stephan Wagner

Thema der Diplomarbeit

Aufgabenorientierte Codegenerierung für webbasierte Benutzeroberflächen

Stichworte

PHP, JSP, ASP, HTML, WWW, Codegenerierung, Eclipse, Java

Kurzzusammenfassung

Diese Arbeit befasst sich mit den Problematiken, welche sich während der Entwicklung von webbasierten Benutzeroberflächen ergeben. Das ist in erster Linie die Vermischung von HTML und Skript-Code (PHP, JSP, ASP und so weiter). Um die Entwicklung solcher Benutzeroberflächen zu erleichtern wird oft die Codegenerierung als Hilfsmittel eingesetzt. Die bisherigen Codegenerierungswerkzeuge haben jedoch einige Schwächen (wie großer Code-Overhead oder eingeschränkte Gestaltungsmöglichkeiten), die in dieser Arbeit erörtert werden. Die Arbeit enthält einen entsprechenden Ansatz für eine verbesserte Codegenerierungslösung. Dieses System arbeitet aufgabenorientiert (Generierung auf Basis konkreter Problemstellungen) und besitzt eine offene Schnittstelle für weitere Lösungsmodule, um immer mehr Problemstellungen abdecken zu können. Es ist als Eclipse-Plug-in realisiert. Die Funktion des Systems wird in einem Szenario überprüft und der Nutzen des Systems gegenüber den bisherigen wird erörtert.

Stephan Wagner

Title of the paper

Task-oriented code generation for web based user interfaces

Keywords

PHP, JSP, ASP, HTML, WWW, Code generation, Eclipse, Java

Abstract

This thesis deals with the problems coming up during development of web based user interfaces. Primarily that's the mix of HTML and script code. In order to ease the development of those user interfaces, code generation is often used, but the previous code generation tools have langours (like large code overhead or limited scope for design) which are discovered in this thesis. The thesis contains an according design of an improved code generation solution. This system works task-oriented (generation based on concrete problems) and owns an open interface for other solution-modules in order to cover more and more problems. It´s realized as a Eclipse-Plug-in. The function of this system is examined on basis of an scenario and the benefit of the system compared to the previous ones is evaluated.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Entwickelt man Web-Anwendungen, also Applikationen die über das WWW[1] mit dem Benutzer kommunizieren, so beschäftigt man sich meist auch mit mehrschichtigen Architekturen (zum Beispiel die 3-Schichten-Architektur). Je nach Betrachtung können sich diese Schichten überschneiden. Gerade auf der Präsentationschicht ist diese Überschneidung bzw. Vermischung stark. Dort gibt es einerseits HTML[2], welches Seiten beschreibt, die der Browser des Benutzers anzeigt und andererseits PHP- (s. [BAS+03]), JSP- (s. [SM03]) oder ASP-Skripte (s. [Cor03a]), die einen bestimmten Anteil der Anwendungslogik („Präsentationslogik“) beinhalten, der serverseitig ausgeführt wird. Beides ist in ein und derselben Datei gespeichert, welche sich auf dem Webserver (also der serverseitigen Präsentationsschicht) befinden. Abbildung 1.1 zeigt die Betrachtung einer 3-Schichten-Architektur aus Hardwaresicht und aus Softwaresicht.

Als Informatiker strebt man meistens nach einer möglichst strikten Trennung zwischen Präsentation und Daten/Anwendungslogik. Diese Trennung ist mit HTML aber nicht ausreichend möglich. Daher wurden bereits andere Ansätze verfolgt, die beispielsweise mit Hilfe von XML die Darstellung von den Daten abzukapseln. Dies geschieht zum Beispiel dadurch, dass definiert wird, wie ein bestimmtes XML-Tag in HTML umgesetzt werden soll. Während des Datenabrufs muss also lediglich eine Datenstruktur in Form von XML erzeugt werden. Ein solcher Ansatz ist XHTML (s. [W3C03a]). Dies hat aber auch den Nachteil, dass das Design einer Seite nicht mehr in der Art und Weise erstellt werden kann, wie es der WebDesigner gewohnt ist. Dieser muss nun statt mit WYSIWYG-Editoren[3] eine

[1] Abkürzung für World Wide Web; Internet-Dienst mit grafischer Bedienoberfläche, der neben der Textübertragung auch die Übertragung von Bildern, Ton- und Videosequenzen ermöglicht. Siehe [Gmb03]

[2] Abkürzung für englisch Hyper-Text Markup Language, eine Dokumentenbeschreibungssprache, mit der Websites im WWW formatiert werden. Die Umsetzung der Darstellung von HTML-Dokumenten übernimmt der Browser. (nähere Infos unter [Mn03a])

[3] Abkürzung für englisch „What You See Is What You Get“. WYSIWYG-Editoren stellen den zu bearbeitenden Inhalt so dar, wie er später (bei Textdokumenten gedruckt, bei Webseiten die Darstellung im Browser) aussieht.

Seite als Ganzes zu entwerfen, das Aussehen kleinerer untergeordneter Elemente definieren. Diese wohlstrukturierte Entwicklung von Internetseiten schränkt den WebDesigner natürlich ein. Er kann seine Ideen nicht mehr frei umsetzen, sondern muss innerhalb der entstehenden Strukturen arbeiten, um später ein gewünschtes Gesamtbild zu erhalten. Deshalb werden Internetseiten auch meistens noch in HTML entworfen. Damit die Entwicklung von Webseiten dennoch erleichtert wird, bedient man sich oft Hilfsmitteln wie der Code-Generierung, um den erforderlichen Skript-Code für die Funktionalität nicht von Hand schreiben zu müssen.

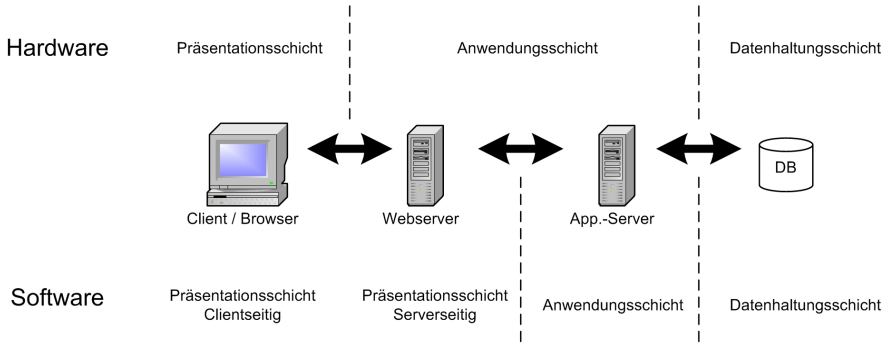

Abbildung 1.1: Unterschiedliche Betrachtung einer 3-Schichten-Architektur

In dieser Arbeit werden zunächst (Kapitel 2.1) einige allgemein erklärende Worte zum Thema webbasierende Benutzeroberflächen (Frontends) zu lesen sein. Anschließend wird auf die Problematiken bei der Entwicklung von Web-Frontends eingegangen. Eines der gravierendsten Probleme ist die erwähnte Vermischung von HTML und Skriptcode. Im darauf folgenden Kapitel (2.2) werden einige bereits vorhandende Lösungsansätze vorgestellt und diskutiert, die sich mit den Problematiken beschäftigen. Es wird aufgezeigt, dass diese Ansätze zwar teilweise ziemlich nützlich sind, aber dennoch auch gewisse Schwächen aufweisen. Aus dieser Diskussion folgen entsprechende Anforderungen für eine verbesserte Lösung zur Erleichterung der Web-Frontend-Entwicklung (Kapitel 2.4). Der anschließende Entwurf einer verbesserten Lösung enthält ein neues Konzept, welches aufgabenorientiert ist. Es soll hier also kein Code-Generierungswerkzeug entwickelt werden, welches nicht alle Aufgaben einer Webseite verallgemeinert und versucht, für alle diese Aufgaben den Code zu generieren, sondern es wird ein System entwickelt, welches für bestimmte, konkrete Aufgaben (die durch den Frontend-Entwickler mitgeteilt werden) den Code ganz gezielt generiert. Dadurch soll die Effektivität des generierten Codes (zum Beispiel weniger Overhead) und die Benutzung des Systems verbessert werden. Letzteres soll dadurch gewährleistet sein, dass der Benutzer das System nur mit den Informationen füttern muss, die für die Lösung eines bestimmten Problems benötigt werden, obwohl die Problemlösung auch komplizierter sein kann. Dieser Entwurf des Systems (Kapitel 4) sieht einen

Editor vor, mit dem der Entwickler die dynamischen Stellen einer HTML-Seite (welche mit einem WYSIWYG-Editor erstellt wurde) kennzeichnen kann nach dem Prinzip „an dieser Stelle möchte ich später dies und das ausgegeben haben“. Im selben Editor soll er die zu benutzenden Problemlösungen wählen können, dem System also mitteilen, welche konkrete Aufgabe die Seite lösen soll. Die Code-Generierung soll ebenfalls aus dem Editor heraus erfolgen. Allgemeine Einstellungen wie Datenbankserver etc. soll der Entwickler zentral tätigen können. Das System soll als Plug-in der Eclipse-Plattform realisiert werden, damit viele Standardfunktion und -features wie Projektverwaltung, Texteditoren etc. nicht neu entwickelt werden müssen. In Kapitel 5.4 wird die konkrete Eingliederung in Eclipse entworfen und anschließend die Realisierung des Systems beschrieben. In den beiden letzten Kapiteln (6 und 7) wird dann erörtert, in wie weit die entstandene Lösung einen Fortschritt gegenüber den bisherigen Ansätzen darstellt und wie groß das Potential des Systems für die Nutzung und Weiterentwicklung des Systems ist.

Kapitel 2

Entwicklung von Web-Frontends

2.1 Allgemeines zu Web-Applikationen

Dynamik im Netz

Die Nutzung des WWW mit dynamischen Inhalten, also mit Web-Seiten, deren Inhalt sich bei jedem Aufruf ändern kann und die Benutzereingaben annehmen und verarbeiten, ist heutzutage nichts ungewöhnliches. Waren dynamische Inhalte zu Anfängen des WWW auf Gästebücher und ähnliches beschränkt, so sind heutzutage komplexe Systeme, wie Online-Auktionen, virtuelle Marktplätze oder Reiseverkehrsauskünfte über das WWW zugänglich.

Aufbau und Funktionsweise von Web-Applikationen

Systeme die über das Internet mit den Anwendern kommunizieren, nutzen als Schnittstelle meist den Internet-Browser des Anwenders. Die gesamte Benutzeroberfläche wird in der Regel als in HTML-Code verfasste Webseiten realisiert. Die Anwendungslogik läuft in allen gängigen Systemen auf dem Server ab, so dass der Anwender weder zusätzliche Ressourcen, noch eine spezielle Softwareinstallation benötigt. Er bekommt vom Server lediglich eine dynamisch erzeugte HTML-Seite zugesendet, die sein Browser anzeigt. Der Benutzer kann auf diesen Seiten durch vom Server erzeugte Links navigieren oder auch gegebenenfalls Eingaben in Textfelder, Checkboxen etc. tätigen. Auf dem Server läuft eine Software die, abhängig von den Anfragen und Eingaben des Benutzers, HTML-Seiten mit unterschiedlichem Inhalt erzeugt. Diese Software kann in verschiedenen Programmiersprachen entwickelt werden und kann auf unterschiedlichen Betriebssystemen laufen.

Prinzipiell bestehen die entstehenden Architekturen kleinerer bis mittelgroßer solcher Systeme aus 3 Schichten

- die clientseitige Präsentation mit dem Browser

- die serverseitige Präsentation (Erzeugung von HTML) und die Anwendungslogik auf dem Web-Server. Dies ist die sogenannte „Mittelschicht“.
- die Datenhaltung auf dem Datenbank-Server

In komplexeren Web-Anwendungen wird die Mittelschicht gespalten, so dass die Zugriffe auf die Datenhaltungsschicht in einer zusätzlichen Schicht gehandhabt werden. Man erhält also aus der Mittelschicht die serverseitige Präsentationsschicht und die Anwendungslogik-Schicht (siehe [Res03]).

Diese Arbeit wird sich mit der Entwicklung auf der serverseitigen Präsentationschicht beschäftigen.

Die dynamische Erzeugung von HTML-Seiten wird oft durch Skript-Sprachen wie PHP, JSP oder ASP bewerkstelligt. Hierbei wird die Funktionalität direkt in HTML-Code eingebettet. D.h., Entwickler solcher Web-Oberflächen entwerfen eine Seite in HTML und kennzeichnen die Stellen, an denen dynamische Inhalte erscheinen sollen. An diesen Stellen wird der entsprechenden Programmcode eingefügt. Es entsteht dadurch ein Quelltext, in dem abwechselnd HTML-Code und Script-Code (PHP-Code bei PHP-Skripten, Java bei JSP-Skripten oder VBScript/JScript bei ASP-Skripten) zu lesen ist. Diese Arbeit wird sich hauptsächlich mit der Frontend-Entwicklung in diesen Sprachen befassen. Die Möglichkeit CGI-Skripte einzusetzen oder andere Skriptsprachen wie z.B. Perl, unterscheidet sich im Ansatz dahingehend von den oben genannten Skriptsprachen, dass sie nicht in HTML eingebunden sind, sondern den HTML-Code selbst über Ausgabefunktionen ausgeben, was im Prinzip das Gegenteil zum obigen Ansatz ist. Diese Arbeit konzentriert sich lediglich auf den Ansatz des in HTML eingebetteten Skript-Codes. Die Entwicklungsarbeit in der serverseitigen Präsentationsschicht kann in zwei Entwickler-Rollen unterteilt werden.

- den Designer, welcher das Erscheinungsbild von Inhalten entwirft. Hierzu benutzt er häufig Designer-Tools.
- den Programmierer, welcher das vom Designer entwickelte „Outfit“ einer Seite mit Daten auffüllt.

Abbildung 2.1 soll dieses Zusammenspiel verdeutlichen.

Wiederkehrende Aufgaben für Frontend-Entwickler

Bei der Entwicklung von Web-Applikationen entstehen trotz unterschiedlichster Projekte immer wiederkehrende Probleme. Ein solches fast Standard-Problem ist die simple tabellarische Darstellung von Datenbankinhalten als HTML-Seite. Zum Beispiel eine Kundenliste eines Unternehmens, eine Menge von Auktionsartikeln oder Aktienkursen. Der Aufbau solcher Seiten ist prinzipiell immer der gleiche. Auf der einen Seite befindet sich die Formulierung einer Datenbankabfrage (üblicherweise in SQL), welche die aufzulistende Datenmenge beschreibt

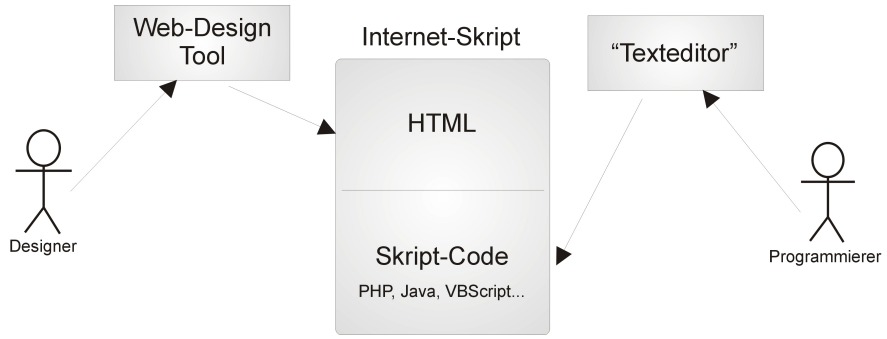

Abbildung 2.1: Beteiligte Rollen bei der Web-Frontend-Entwicklung

und auf der anderen Seite das Design, in dem die Datenmenge erscheinen soll. Gegebenenfalls existieren für jeden Datensatz innere Abfragen und Funktionen (zum Beispiel, um die Menge von abgegebenen Geboten für jeden Auktionsartikel anzuzeigen). Im Falle einer 3-Schichten-Architektur, in der die serverseitige Präsentationsschicht mit dem Zugriff auf die Datenhaltungsschicht zusammenfällt (s.o.), bedarf die Erzeugung solcher Seiten im Prinzip folgender Schritte:

- Aufbau einer Datenbankverbindung und Senden einer DB-Anfrage (evtl. abhängig von Benutzereingaben)
- Schreiben des Seitenkopfes und des Seiteninhaltes bis zur ersten Tabellenzeile in HTML (einschließlich Eröffnung der Tabelle und evtl. Spaltenüberschriften)
- Schreiben des sich wiederholenden Bereichs der Tabelle für jeden Datensatz (Damit sind die eigentlichen Tabellenzeilen gemeint). Hierbei wechseln sich Skript-Code und HTML-Code ab. Als Rahmen dient meistens eine While-Do-Schleife in der Skript-Sprache. In dieser wird der nötige HTML-Code für jede Zeile geschrieben.
- Schreiben des Seiteninhaltes nach dem Wiederholbereich (Schliessen der Tabelle und anschließende Inhalte der Seite)
- Schließen der Datenbankverbindung

Die ersten beiden und letzten beiden Schritte können jeweils untereinander vertauscht werden, da es zum Beispiel nicht nötig ist, dass bereits Daten aus der Datenbank eingelesen werden, zu einem Zeitpunkt, zu dem der Seiteninhalt bis einschließlich des Tabellenkopfes geschrieben wird. Die Abbildungen 2.2 und 2.3 sollen das Vorgehen durch beispielhaften Quelltext in PHP und die daraus resultierende HTML-Seite verdeutlichen.

In einer 4-Schichten-Architektur würden anstelle der Datenbankaufrufe (Verbindungsaufbau, Anfrage usw.) Delegationen der Anfrage an die hintere Schicht

```php
<?php
  // Aufbau der DB-Verbindung
  $dbc = mysql_connect($DB_HOST_URL,$User,$Pass);
  mysql_select_db($DB_NAME,$dbc);
  // Senden einer DB Abfrage
  $kunden = mysql_query("select * from Kunde");
?>
<head>
  <title>Kundenliste</title>
</head>
<body>
  Vorhandene Kunden in der Datenbank:<br>
  <table>
    <tr>
      <th>Name</th>
      <th>Vorname</th>
    </tr>
    <?php
      // Hauptschleife: Solange das Ende der Ergebnis-
      // menge nicht erreicht -> Neue Zeilen erzeugen
      while ($kundeRec = mysql_fetch_object($kunden))
      {
        echo("<tr>");
        echo("<td>".$kundeRec->Name."</td>");
        echo("<td>".$kundeRec->Vorname."</td>");
        echo("</tr>");
      }
    ?>
  </table>
</body>
<?php
  mysql_close();
?>
```

Abbildung 2.2: Datenbankzugriff mit PHP

```html
<head>
  <title>Kundenliste</title>
</head>
<body>
  Vorhandene Kunden in der Datenbank:<br>
  <table>
    <tr>
      <th>Name</th>
      <th>Vorname</th>
    </tr>
    <tr>
      <td>Meier</td>
      <td>Herbert</td>
    </tr>
    <tr>
      <td>Soltau</td>
      <td>Manfred</td>
    </tr>
    <tr>
      <td>Schuster</td>
      <td>Susanne</td>
    </tr>
  </table>
</body>
```

Abbildung 2.3: Beim Ausführen erzeugter HTML-Code

(Datenhaltungszugriffsschicht) eingesetzt. Soll heißen, dass alle datenrelevanten Operationen (Datenbankzugriffe, evtl. Transaktionsoperationen etc.) nicht mehr in dieser Schicht stattfinden, sondern durch Aufrufe und Delegation erledigt werden.

Die Schwierigkeiten bei der Skript-Programmierung

Da es sich bei der Entwicklung von Web-Frontends, neben den Designfragen, zu einem großen Teil um Programmierung handelt, ist der Einsatz von unterstützenden Tools ratsam, welche die Programmierung in den Skript-Sprachen erleichtern. Eine der am häufigsten durch Tools angebotene Erleichterung ist das Syntax-Highlighting[1]. Eine weitere Unterstützung bietet beispielsweise die automatische Code-Vervollständigung (Code-Completion[2]). Je nach Komplexität der Daten und des Designs wird der zu schreibende Code eventuell schwer bis nicht mehr leserlich, da sich HTML und PHP bzw. JSP oder ASP häufig abwechseln. Für das Auge des Programmierers verschmelzen beide Sprachen zu einem unübersichtlichen Code-Gewirr auch trotz Syntax-Highlighting. Dies ist dann besonders wahrscheinlich, wenn sich auf der zu erzeugenden Seite zusätzlich JavaScript-Elemente (siehe [Mn03b]) befinden.

Oft ist es erwünscht, das Design mit einem WYSIWYG-Editor zu erstellen und die Funktionalität später hinzuzufügen. Da so erstellter HTML-Code von Haus aus sehr komplex ist (bereits durch wenige Design-Ansprüche können bereits zahlreiche komplexe Tabellenkonstrukte in HTML entstehen, welche selbst für geübte HTML-Programmierer nur noch schwer leserlich sind), führt dies erst recht zu Schwierigkeiten beim späteren Einfügen des Skript-Codes. In der Umkehrung ist es nicht möglich HTML-Design mit einem WYSIWYG-Editor nachträglich um bestehenden Skript-Code „herum“ einzufügen. Besonders aufwendig ist eine spätere Änderung des Designs, so dass viel HTML-Code in allen möglichen Skripten angepasst werden muss, was durch die vielen eingebetteten Skript-Code-Stücke entsprechend mühselig wird.

Der Designer und der Programmierer arbeiten beide an dem selben Dokument. Das hat logischerweise zur Folge, dass der Designer durch den Skript-Code „behindert“ wird und der Programmierer durch den HTML-Code. Zudem kann es bei späteren Änderungen natürlich schnell vorkommen, dass der Kollege Code des anderen Kollegen während seiner Arbeit beschädigt.

[1] die farbliche oder dekorative (fett, kursiv etc.) Hervorhebung von Schlüsselwörtern und Blöcken, entsprechend der Sprach-Syntax, im Quelltext

[2] während der Programmierung werden dem Benutzer Vervollständigungen an den Stellen vorgeschlagen, an denen er gerade arbeitet. Zum Beispiel ausgeschriebene Funktionsaufrufe, wenn Anfangsbuchstaben von Funktionen eingegeben wurden. Es wird eine Liste mit Vorschlägen an der aktuellen Cursor-Position angezeigt, aus der ein und auch kein Vorschlag gewählt werden kann

2.2 Vorhandene Lösungsansätze für effizientere Frontend-Entwicklung

Da die oben beschrieben Lösungen für wiederkehrende Probleme eigentlich immer dem gleichen Schema folgen, liegt der Wunsch nahe, eine allgemeingültige Lösung zu finden, die dafür eingesetzt werden kann, die Frontend-Entwicklung effizienter zu gestalten. Einige sollen an dieser Stelle vorgestellt werden.

Triviale Code-Generatoren

Im Internet findet man einige kostenlose oder -günstige Code-Generatoren, welche PHP, ASP oder JSP-Code erzeugen können. Die einfachste Ausprägung solcher Tools ermöglicht es dem Entwickler SQL-Statements anzugeben und einige Kriterien für das Design, wie Farbe, Rahmen etc., festzulegen. Daraufhin wird dem Entwickler ein recht einfaches PHP-Skript generiert, welches die Ergebnismenge der Datenbankabfrage so darstellt, wie es eingestellt wurde. Hierbei ist der Einfluss auf das Design allerdings sehr beschränkt, so dass in den meisten Fällen das Skript nachbearbeitet werden muss. Das führt nicht nur eventuell zu oben genannten Problemen, sondern wird vielleicht durch unübersichtlich generierten Code zusätzlich erschwert.

Reporting-Tools für Datenbanken

Für die namenhaften Datenbanksysteme wie Oracle™ , SQL-Server™ etc. existieren sogenannte Reporting-Tools. Datenbank-Reports dienen grundsätzlich dazu, den aktuellen Datenbestand des Systems in einer gut leserlichen Form auszugeben. Wobei mit „ausgeben“ meistens ausdrucken gemeint ist. In einem Reporting-Tool definiert der Entwickler, wie die einzelnen Felder einer Datenmenge tabellarisch dargestellt werden, wie die Daten gruppiert werden sollen und wie z.B. diese Gruppen äußerlich voneinander getrennt werden. Wurden Datenbank-Reports in der Vergangenheit meist nur für bedruckbare Papierseiten oder für Excel-Sheets erstellt, so bieten heutzutage die meisten Tools auch Reports in HTML-Format (also als Webseiten) an.

Generatoren für „komplette Anwendungen“

Es gibt Tools, welche dem Entwickler versprechen, eine komplette Web-Applikation in ein paar Minuten fertigzustellen. Eines dieser Tools ist phpLens (s. [Nat03]). Hier gibt der Entwickler die zu benutzenden Datenbanktabellen, Feldfilter, Nutzerrechte usw. an und es werden entsprechend der Angaben fertige Tabellenansichten, Formulare zum Bearbeiten und alle zugehörigen SQL-Anfragen in einem Standard-Design generiert. Hier hat der Entwickler einen ähnlich beschränkten

Einfluss auf das Design, wie oben beschrieben. Bei einem solchen „Schnellschuss“ von einer kompletten Web-Applikation wird stets die Annahme gemacht, dass die Anforderungen an das System sich auf rudimentäre Auflistung und Bearbeitung von Datensätzen beschränken. Die Verknüpfung von Tabellen, sowie die Bearbeitung von verknüpften Daten (Beispiel: Ein Kunde kann mehrere Adressen haben.) wird nicht unterstützt. Alle individuellen Anforderungen müssen also nach dem Generierungsvorgang von Hand ausprogrammiert werden, wobei sich hier erstmal in ein leicht komplexes System einzuarbeiten gilt (die Anwendung wurde ja „komplett“ generiert!). Das Generat solcher Tools ist eher eine verallgemeinerte Ausimplemetierung einer trivialen Datenbank-Anwendung, bei der lediglich die Tabellen und Felder ausgetauscht werden.

Komplette Entwicklungsumgebungen für Web-Frontends

Einen Lösungsansatz zur Erleichterung der Web-Frontend-Entwicklung ist die Entwicklungsumgebung CodeCharge™ . CodeCharge™ (s. [Yes03]) ist ein WYSIWYG-Editor mit eingebauten Skript-Generatoren. Das Design wird wie in gängigen Web-Design-Tools (zum Beispiel Adobe™ GoLive™ oder Macromedia™ Dreamweaver™) bearbeitet. SQL-Abfragen und deren variable Ergebnisfelder werden durch den Editor eingefügt, ohne dass Code geschrieben werden muss (außer SQL). Alle Einstellung, wie beispielsweise die bedingte Ausgabe von Seitenbereichen zur Laufzeit, werden editorgestützt vorgenommen und bedürfen keiner Programmiertätigkeit. Der Benutzer von CodeCharge™ kann die Zielsprache, in der das Ergebnis generiert werden soll, aussuchen. Es werden unter anderem die oben genannten Sprachen ASP, JSP und PHP unterstützt. Die Zielsprache kann später gewechselt werden, da die Bearbeitung vor dem Generierungsschritt nicht abhängig von dieser ist. Generiert wird eine komplette Anwendung, die jedoch gänzlich frei definierbares Design trägt.

2.3 Probleme mit diesen Lösungsansätzen

Probleme mit trivialen Code-Generatoren

Der Generator muss nach der gewünschten Zielsprache ausgesucht werden, da die meisten solcher „einfachen“ Generatoren nur eine Sprache unterstützen. Die generierten Skripte können lediglich als Grundgerüst für eine dynamische Seite angesehen werden, denn das Design lässt sich, je nach Tool, gar nicht oder nur eingeschränkt beeinflussen. Daraus ergibt sich, dass der entstehende Code in den meisten Fällen nachbearbeitet werden muss, so dass die Vorteile solcher Generatoren gegenüber der einfachen Wiederverwendung von Code-Stücken via Copy-Paste nicht mehr sehr stark zum Tragen kommen. Geeignet ist diese Anwendung also nur, wenn das Aussehen der Oberfläche unwichtig ist und lediglich

eine Möglichkeit der Dateneinsicht (und -Pflege) geschafft werden soll. Dies könnte bei Seiten, die lediglich der Administration von Datenbeständen dienen sollen und somit nicht öffentlich sichtbar sind, der Fall sein. Die Anwendung trivialer Code-Generatoren unterliegt zusätzlich der Einschränkung, dass es sich um einfachste Datenstrukturen handelt (keine verknüpften Tabellen oder ähnliches), die über das Web zugänglich gemacht werden sollen.

Probleme mit Reporting-Tools

Datenbank-Reports unterliegen der Einschränkung, dass sie konzeptionell nur für das Auslesen und Darstellen der Datenbankinhalte gedacht sind und nicht zur Manipulation dieser Daten. Solche Reports in HTML-Format sind zwar geeignet, den aktuellen Datenbank-Inhalt durch das Web einzusehen, aber sie ermöglichen es einem grundsätzlich nicht, die Daten zu bearbeiten oder gar eine spezielle Anwendungslogik zu kapseln. Somit sind HTML-Reports lediglich für die Dateneinsicht geeignet, jedoch keineswegs, um neue Web-Applikationen zu steuern.

Probleme mit „Komplett“-Generatoren

Versprechen Tools wie phpLens die Erstellung von Web-Anwendungen in Minutenschnelle, so ist dies nicht gelogen. Die generierten PHP-Anwendungen bieten Tabellenansichten, Detailansichten und die nötigen Formulare, um Datensätze zu bearbeiten oder neu zu erstellen. Jedoch mit starken Einschränkungen, denn die generierte Oberfläche kann fast überhaupt nicht individuell angepasst werden (bis auf die Auswahl eines Farbschemas). Deshalb kann auch hier gesagt werden, dass das Generat nur eingesetzt werden sollte, wenn kein Wert auf das individuelle Aussehen der Web-Applikation gelegt wird. Zusätzlich sollten auch keine nicht-trivialen Datenstrukturen (z.B. Beziehungen zwischen Tabellen) auftreten, da diese, wie bei oben genannten Minitools, nicht unterstützt werden können. Möchte man als Entwickler die Generierung von phpLens als Grundgerüst für ein System nutzen, so sollte abgewägt werden, inwieweit die Generate von phpLens mit den Anforderungen an das eigene System übereinstimmen. Eine manuelle Erweiterung könnte sonst aufwändiger werden, als die gänzliche Eigenentwicklung, denn es bedarf einer gewissen Einarbeitung, um den generierten Code umschreiben zu können.

Probleme mit CodeCharge™

Designer von Web-Frontends arbeiten meistens nicht mit Texteditoren, in denen sie HTML-Code schreiben, sondern mit WYSIWYG-Editoren, die den entsprechenden HTML-Code gemäß dem erstellen Design erzeugen. Diese Editoren haben verschiedene Vor- und Nachteile untereinander und so hat natürlich jeder Designer sein Lieblings-Tool, mit welchem er am schnellsten seine besten Ergebnisse

erzielen kann. CodeCharge™ basiert darauf, dass der eingebaute WYSIWYG-Editor benutzt wird. Daraus folgt natürlich, dass der Designer erst einmal Zeit benötigt, um mit diesem umgehen zu können. Zudem muss er auf evtl vorhandene Features seiner bisherigen Tools verzichten.

Wenn neue Zielsprachen (wie jüngst ASP.NET) in den Umlauf kommen oder neue Features von einigen Datenbanken angeboten werden, ist der Entwickler darauf angewiesen, dass YesSoftware™ die entsprechenden Unterstützungsmodule für CodeCharge™ auf den Markt bringt. YesSoftware™ ist somit der einzige Lieferant von eventuelle Erweiterungen dieses Systems.

Es bedarf eines nicht unerheblichen Lernaufwandes, mit dieser Entwicklungsumgebung und ihrer Art und Weise, dynamische Webseiten zu erstellen, umgehen zu können. Richtig effizient wird der Einsatz von CodeCharge™ also erst nach einer längeren Einarbeitungsphase. Dies gilt für den Designer und für den Programmierer.

Da CodeCharge™ dafür gedacht ist, komplette Anwendungen zu erzeugen, werden beim Generierungsschritt immer alle eventuell auftretenden Anforderungen unterstützt. Dadurch wird oft viel Code generiert, der letztlich gar nicht benutzt wird oder der bei eigener Programmierung viel schlanker geworden wäre, da er nicht so allgemeingültiger gehalten sein müsste, wie der von CodeCharge™ generierte.

Für die Arbeit mit CodeCharge™ wird ein eigenes Dateiformat benutzt. Dieses kann nur von dieser Entwicklungsumgebung gelesen werden. Ohne CodeCharge™ sind diese Dateien unbrauchbar. Das bedeutet, sobald der Entwickler zu einer anderen Entwicklungsumgebung wechseln möchte, sind die bisher eingegebenen Daten (Datenbankabfrage etc.) nicht mehr nutzbar.

Lesende Datenbankoperationen können über einen sogenannten Visual-Query-Designer bearbeitet werden, mit dem man, ähnlich wie bei Microsoft Access™ (s. [Cor03b]), Datenbankabfragen visuell erstellen kann. Dies ist allerdings für geübte Entwickler mit Datenbankkenntnissen aufwändiger, als die schlichte Eingabe von SQL-Statements. Schreibende Operationen können in ihrer Art und Weise vom Entwickler nicht beeinflusst werden. D.h. es kann kein Einfluss darauf genommen werde, wie Daten in die Datenbank geschrieben werden (z.B. durch Transaktionen[3]). Datenbankoperationen werden grundsätzlich in den Skript-Code direkt hineingeneriert. Die Aufteilung von serverseitiger Präsentation und Anwendungslogik (wozu Datenbankoperationen zählen) wird nicht unterstützt.

[3]Eine Transaktion ist eine konsistenzerhaltende Operation auf einer Datenbank, d.h. sie lässt die Datenbank in konsistentem Zustand zurück, wenn diese vor Beginn der Transaktion schon konsistent war. (s. [Zeh98])

2.4 Resultierende Anforderungen für eine verbesserte Lösung

Generator-Suite mit modularer, offener Struktur

Für die Unterstützung bei bestimmten wiederkehrenden Problemen (z.B. Tabellenansichten), sollte die Lösung eine Entwicklungsumgebung sein, die modular aufgebaut ist. Der Entwickler sollte die Möglichkeit haben, nur die nötigen Module zu benutzen, damit er das System gezielter auf seine aktuelle Problemstellung bzw. seine Anforderungen ansetzen kann. Es soll also kein allgemeingültiger Standard-Code generiert werden, der für alle Eventualitäten „gerüstet“ ist und somit einen großen Overhead darstellt, sondern Code, der das gewünschte Problem direkt und ohne Umschweife löst. Für unterschiedliche Probleme gäbe es also unterschiedliche Module, statt ein Modul, welches alle Probleme löst. Der Entwickler würde in der Entwicklungsumgebung, zu jedem Problem das entsprechende Modul auswählen und dieses auf das Problem ansetzen.

Die Schnittstelle für neue Module sollte offen sein. So können Entwickler bestimmte wiederverwendbare Lösungen untereinander austauschen bzw. verbreiten, wie eine Art Entwurfsmuster (s. [GHJV93]).

Die Lösungsmodule sollten dabei selbst bekanntgeben können, welche Zielsprache sie unterstützen. Der WebFrontend-Entwickler kann dadurch sicherstellen,dass er nur solche Problemlösungsmodule auswählt, welche Code seiner gewünschten Zielsprache generieren können.

Verzicht auf neue Dateiformate

Wenn es möglich ist, sollte auf die Einführung eines oder mehrerer Dateiformate verzichtet werden. So bleibt gewährleistet, dass geleistete Arbeiten bei einem eventuellen Umstieg auf eine andere Entwicklungsumgebung (oder andere Tools) weiterverwendet werden können bzw., dass zumindest entstehende Informationen nicht verloren gehen.

Trennung von Design und Anwendungslogik

Die Entwicklung des Designs und der Anwendungslogik sollten getrennt vorgenommen werden können. So sollte es zum Beispiel dem Designer auch möglich sein, sein favorisiertes Web-Design-Werkzeug einzusetzen. Außerdem soll dieser sich nicht von eingebettetem Skript-Code irritieren lassen müssen. Dies soll lediglich die Entwicklung betreffen. Die Trennung von Design und Anwendungslogik an sich soll das System nicht gewährleisten können.

Mögliche Entwicklung von n-schichtigen Web-Anwendungen

Es sollte die Möglichkeit geben, dass Web-Anwendungen entwickelt werden können, welche über mehr als eine Server-Schicht verteilt sind (z.B. Präsentation und Anwendungslogik). Hierbei sollten die entsprechenden Delegationen und Funktionsaufrufe schichtübergreifend generiert werden können.

Keine Einbindung speziellen Verhaltens des Systems

Wird ein neues System entwickelt, so kann eine Anforderung sein, dass zum Beispiel schreibende Datenbankoperationen nur transaktionell geschehen sollen oder, dass bei allen Operationen bestimmte Sicherheitsbedingungen erfüllt sein müssen. Solche besonderen Verhaltensweisen von System, lassen sich schwerlich durch Code-Generierung erzeugen. Daher sollte vorerst davon abgesehen werden, all diese Funktionalitäten unterstützen zu wollen, da man Gefahr läuft zu spezielle Anforderungen zu bedienen, deren Lösungen später so nicht eingesetzt werden können. Der mit dem zu entwickelnden System generierte Skript-Code sollte in solchen Fällen als Vorlage verstanden werden und um die Spezialfunktionen von Hand erweitert werden.

Abgrenzung zu den „großen“ Systemen

An dieser Stelle soll darauf hingewiesen sein, dass in dieser Arbeit eine Lösung für die Entwicklung von Frontends mittelgroßer Websites diskutiert wird. Diese Lösung wird nicht die „großen“ Systeme wie SAP™ oder beispielsweise komplexe Marktplatzsysteme[4] namenhafter Anbieter ersetzen. Jemand, der vorhat ein komplexes Marktplatzsystem zu betreiben, ist gut beraten, ein fertiges System zu nutzen und das Frontend nötigenfalls (wenn vom System nicht anders unterstützt) von Hand anzupassen, anstatt mit einer Lösung aus dieser Arbeit das Frontend bequem gestalten zu können, aber dafür die Marktplatz-Logik dahinter (oder entsprechende Schnittstellen) neu entwickeln zu müssen. Ebenso wird es für ein Unternehmen, welches bereits ein SAP™-System hat und nun einen Web-Auftritt starten möchte, sinnvoll sein, die von SAP™ angebotenen WEB-Technologien zu nutzen, anstatt die Verbindung der hier diskutierten Lösung zum SAP-System selbst entwickeln zu müssen.

[4]Elektronische oder virtuelle Marktplätze sind Internet-Applikationen, die den Handel zwischen mehreren Anbietern (evtl. einer bestimmten Branche) und den Kunden (oder „Nachfragern“) über das WWW ermöglichen.

Kapitel 3

Szenario: SAM

Um die Einsatzmöglichkeiten einer verbesserten Unterstützung der Web-Frontend-Entwicklung zu verdeutlichen, hier ein Szenario:

Für das Unternehmen OMD Outdoor, ein Spezialmittler[1] im Bereich Außenwerbung, soll eine Internet-Plattform entwickelt werden, auf der Kunden und Kampagnen-Planer sich über sogenannte „Ambient-Medien“[2] informieren können. Die Anforderungen im Groben lauten

- Alle Angebote an Ambient-Medien sollen in einer Datenbank gespeichert sein.
- Die Angebote sollen in mehreren Kategoriearten kategorisiert werden (z.B. gibt es eine Kategorieart „Format“, in der sich Kategorien wie „DIN A1“ etc. befinden. Ein Medienangebot kann beliebig vielen dieser Kategorien zugeordnet sein, was zum Beispiel bedeutet, dass ein Medium in den Formaten DIN A2 und DIN A1 angeboten wird oder auch, dass es beispielsweise in KEINEM Standardformat angeboten wird)
- Den Angeboten sollen Bilder, Infomateriel (z.B. Powerpoint™-Präsentationen) und Konditionen zugeordnet werden können. Konditionen sollen in verschiedene Arten unterteilt werden. Z.B. Preis, Mindestmenge oder Rabatte.
- Konditionen bestimmter Arten (irgendwelche Sonderrabatte) oder bestimmte Infomaterialien sollen nicht für jedermann sichtbar sein. Natürlich darf auch nicht jeder die Angebote bearbeiten. (Benutzer-Verwaltung).

[1] vermittelt zwischen Werbetreibenden (z.B. Werbeagenturen) und den Anbietern der Medien (z.B. Pächter von Plakatflächen)

[2] damit sind Medien gemeint, die überall in der Umgebung auftauchen, die aber nicht einfache Plakate o.ä. sind. Ein Beispiel hierfür ist Werbung auf den Kopfstützen in Taxen. Sozusagen alle etwas „außergewöhnlichen“ Medien

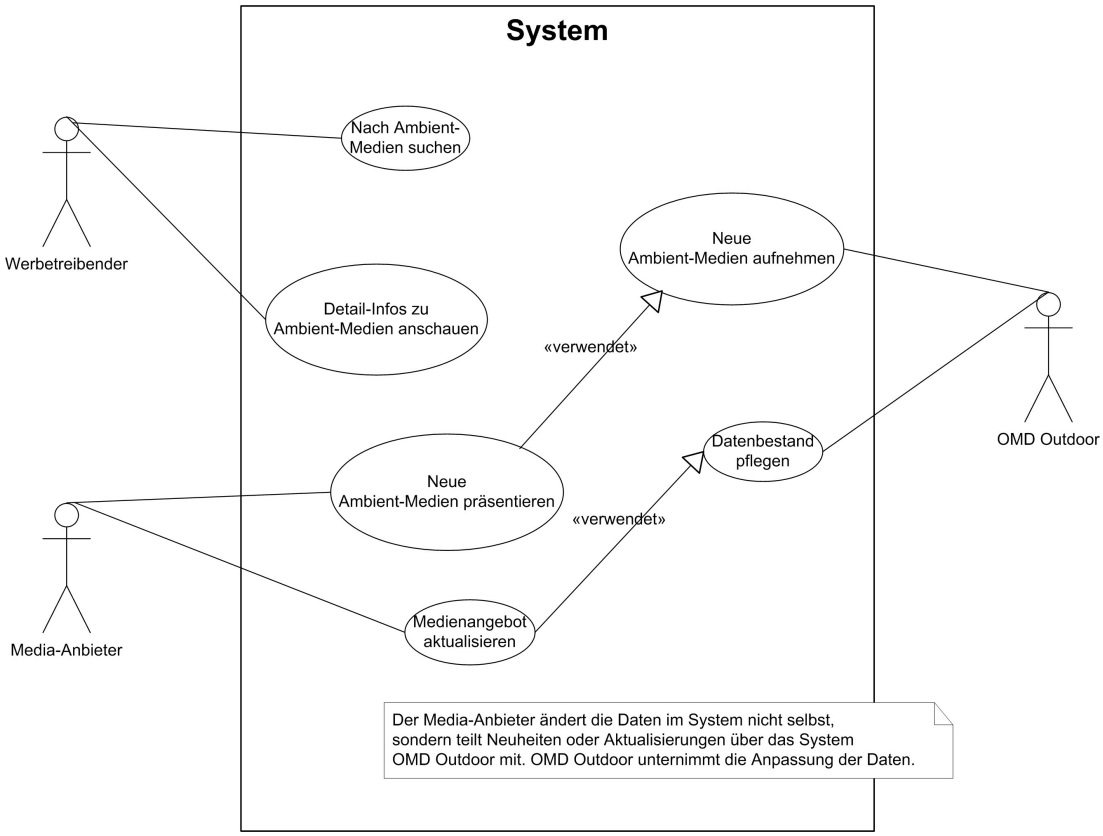

Abbildung 3.1: Anwendungsfälle bei SAM

- Die Datenbank soll nach Stichworten, Anbietern und Kategorien durchsucht werden können.
- Die Applikation soll auf einer Linux-Plattform mit PHP und einer MySQL-DB laufen. (Vorgabe aus Kostengründen)

Abbildung 3.1 zeigt zusammenfassend die Anwendungsfälle und Rollenverteilung im SAM-System auf. Der Name „SAM“ ist eine Abkürzung und steht schlicht für „Search for Ambient Media“.

Bei der Entwicklung dieses Systems ergeben sich unter anderem folgende wiederkehrenden Aufgaben:

- Darstellung von Abfrageergebnissen in einer tabellarischen Übersicht. Dazu gehört beispielsweise die Auflistung aller Anbieter von Ambient-Medien mit Verweis auf Detailansichten der einzelnen Anbieter.
- Darstellung von Eingabeformularen für die Bearbeitung von Datensätzen. Dazu gehört zum Beispiel ein Bearbeitungsformular für Anbieter oder für einzelne Medienangebote.

Wie in 2.1 beschrieben, sieht die Lösung für diese Aufgaben prinzipiell gleich aus. Wenn auch die Reihenfolge einzelner Schritte variiert oder die Auskapselung bestimmter Schritte in andere Systemmodule (evtl. schichtübergreifend) erfolgt, so bleibt dies zumindest innerhalb eines Projektes recht einheitlich (Ausnahmen gibt es natürlich überall). Um die Entwicklung nun zu vereinfachen, wäre es sinnvoll, das in 2.1 beschriebene - oder ein abgewandeltes - Vorgehen nur einmal definieren zu müssen und bei der Entwicklung aller Seiten eines Web-Auftritts darauf zurückzugreifen. Wie eine Lösung für diese Idee aussehen kann, welche nicht die in 2.3 erörterten Mängel aufweist, soll in den folgenden Abschnitten diskutiert werden.

Kapitel 4

Allgemeines Design des Systems

4.1 Lösungsentwurf

Der Grundgedanke beim Entwurf der neuen Lösung ist,dass der Entwickler weiß, welche Aufgabe bzw. welches Problem eine Internetseite (also ein Skript) lösen soll. Bekommt der Entwickler die Möglichkeit dem System dieses Wissen mitzuteilen, so kann dieses dazu benutzt werden, gezielter - auf die entsprechende Aufgabe bezogen - Code zu generieren. Dazu wird das System für jede Art von Problem ein Lösungsmodul zur Auswahl bieten. Ist die Wahl getroffen, müssen im HTML-Code die dynamischen Stellen (Stellen, an denen ein bestimmter Inhalt zur Laufzeit eingefügt werden soll) entsprechend gekennzeichnet werden. Hierzu wird das System einen Editor beinhalten, der diese Kennzeichnung/Markierung ermöglicht. Abbildung 4.1 zeigt den vorgesehenen Arbeitsfluss des Systems. Daraus wird deutlich,dass zunächst das Design entworfen wird (dies geschieht außerhalb des Systems). Anschließend soll die entstandene Datei (wahrscheinlich eine HTML-Datei) in das System eingelesen werden. Danach stellt der Frontend-Entwickler alle Plattformeigenschaften ein. Diese sind global und müssen später nicht für jede Seite neu eingestellt werden. Dann wählt er die zu lösende Aufgabe der geladenen Seite aus. Anschließend werden mit Hilfe des entstehenden Editors die dynamischen Stellen im HTML-Code markiert. Nun kann der aufgabenbezogene Code generiert werden. Nach der Code-Generierung muss das Skript noch an spezielle Anforderungen angepasst werden, da das System nicht den Anspruch haben wird, Code für alle Aufgaben und Probleme der Welt generieren zu können.

Editor und Markierungen

Was für die angestrebte Lösung benötigt wird, ist zunächst einmal ein Editor, der das Design als HTML einlesen kann und in dem der Webfrontend-Entwickler alle dynamischen Stellen einer Seite kennzeichnen kann. Im Fall von „SAM“ handelt es sich zum Beispiel um die Seite, in der alle Anbieter aufgelistet werden. Diese Seite wurde ja mit Hilfe eines externen Design-Tools vom Webdesigner

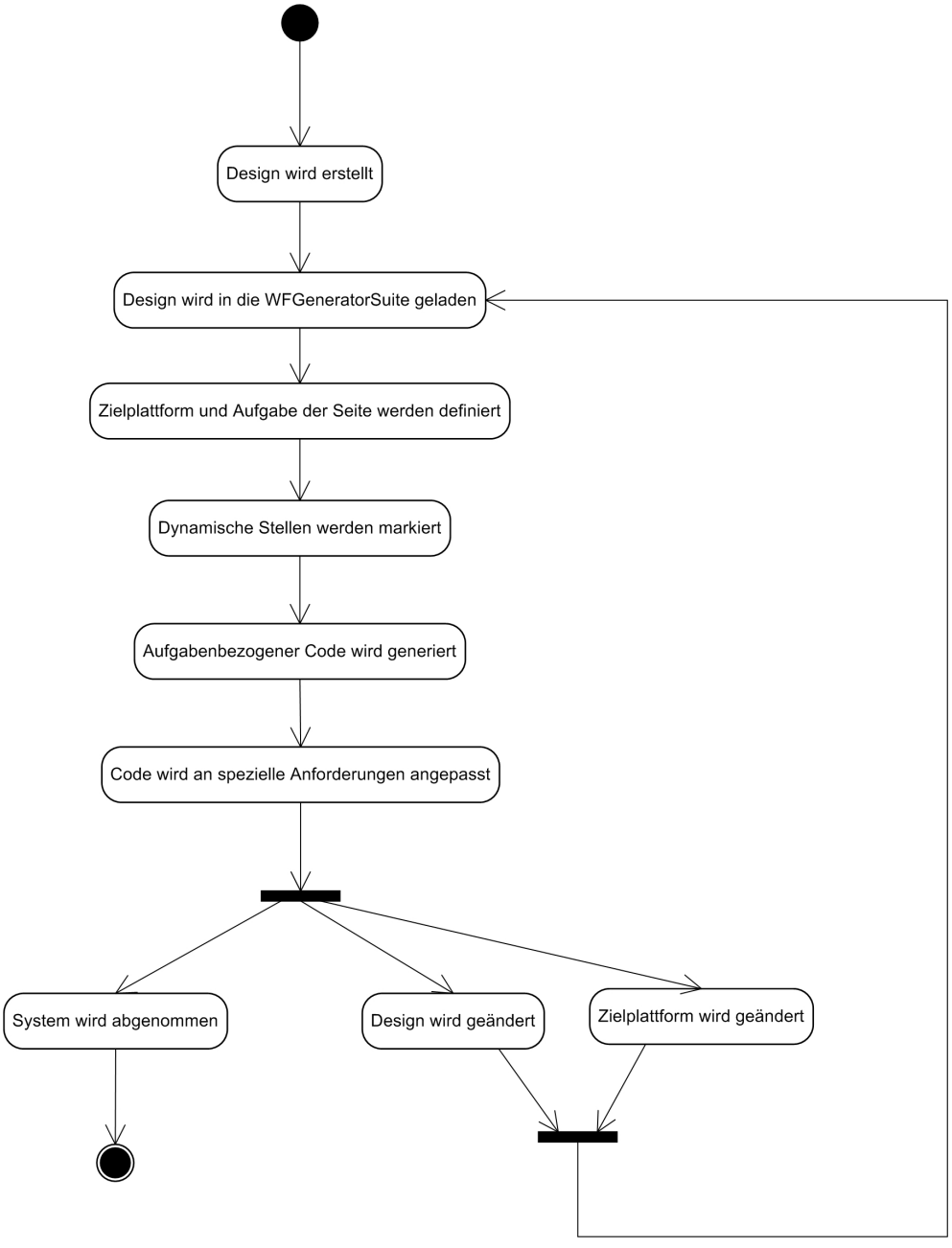

Abbildung 4.1: Vorgesehener Einsatz der Generator-Suite

gestaltet und liegt jetzt in besagtem Editor als HTML-Code vor. Im nächsten Schritt muss der Entwickler wählen, welches Problem er lösen möchte. Hier das Problem der Auflistung von gefundenen Datensätzen (Anbietern). Nun muss ja irgendwie gekennzeichnet werden, welcher Bereich im HTML-Code derjenige ist, welcher für jeden Anbieter wiederholt wird oder allgemeiner: welcher Bereich der „Wiederholbereich“ für jeden gefunden Datensatz einer bestimmten DB-Abfrage ist. Außerdem muss markiert werden, an welcher Stelle die einzelnen Feldinhalte des Abfrageergebnisses eingesetzt werden sollen. Die Frage dabei ist, wie bzw. wo speichert man diese Markierungen (Wiederholbereich/Felder)?

- Man könnte sie einfach im Hauptspeicher halten. Allerdings gingen diese Informationen natürlich nach Schliessen des Editors verloren, so dass der Entwickler sie jedesmal neu eingeben muss.

- Man könnte sie in spezielle Dateien schreiben, die mit dem aktuellen HTML-File verknüpft werden. So könnten diese Dateien beim nächsten Start des Editors mit diesem HTML-File wieder geladen werden. Eine Anforderung in Kapitel 2.4 war aber, dass möglichst auf neue Dateiformate verzichtet werden soll, damit diese Informationen auch ohne den speziellen Editor lesbar sind.

- Die dritte Möglichkeit wäre, die Kennzeichnungen direkt in den HTML-Code einzupflechten. Dies vorzugsweise als HTML-Kommentar, so dass die Datei weiterhin eine gültige HTML-Datei bleibt, welche vielleicht weitere Male in Design-Tools geladen werden kann. Um die Lesbarkeit des HTML-Codes im Editor zu erhöhen wäre zu überlegen, ob man die Einpflanzung der Markierungen als Kommentare erst beim Speichern der Datei vornimmt und während der Arbeit die entsprechenden Bereich „sinnvoller“ hervorhebt. Es wäre zum Beispiel denkbar, den Wiederholbereich einer Auflistung mit einer anderen Hintergrundfarbe hervorzuheben, statt ihn lediglich durch Kommentare abzugrenzen. Beim Wiedereinlesen der (kommentierten) HTML-Datei könnten die Kommentare dann wieder in die optimierte Darstellung umgesetzt werden. Was man entwickeln würde, wäre also ein Texteditor mit farblicher Hervorhebung bestimmter Bereiche. Zusätzlich muss der Entwickler natürlich problemspezifische Daten, wie etwa das benötigte SQL-Statement für die Abfrage der Anbieter, eingeben können. Diese könnten ebenfalls als Kommentar mitgespeichert werden und beim Arbeiten mit dem Editor in einer zusätzlichen Maske editierbar sein.

Angesichts der Nachteile der ersten beiden Ansätze, soll hier die dritte Möglichkeit, die Einpflechtung der problemspezifischen Daten in den HTML-Code, weiterverfolgt werden. Das bedeutet,dass im Beispiel der Auflistung aller Anbieter von Ambientmedien (SAM-Projekt, siehe 3) folgende Daten als Kommentare im HTML-Design abgelegt werden:

```
...

<!-- Problem: Tabellarische Darstellung-->
<!-- SQL: "Select ID, Name from Anbieter"-->
<table>
  <tr>
    <th>Nr.</th>
    <th>Name</th>
  </tr>
<!-- Wiederholbereich-Anfang -->
  <tr>
    <td><!-- Feld: ID--></td>
    <td><!-- Feld: Name--></td>
  </tr>
<!-- Wiederholbereich-Ende -->
</table>

...
```

Abbildung 4.2: Einpflechtung von Markierungen relevanter Bereiche im HTML-Design

- Die Art des Problems, welches gelöst werden soll (tabellarische Darstellung) bzw. ein identifizierendes Merkmal des entsprechenden Lösungsmodules
- Die Markierung des Wiederholbereiches. Bei anderen Problemen kann es natürlich auch Markierungen geben, die anderen Zwecken dienen. Es wird also verschieden Typen von Markierungen geben.
- Die Kennzeichnung der Stellen, an denen die Feldinhalte der Datensätze eingesetzt werden sollen
- Ein SQL-Statement, welches die Abfrage aller Ambientmedien-Anbieter formuliert.

Abbildung 4.2 zeigt prinzipiell, wie der entstehende HTML-Code nach der Kommentierung aussehen würde. Der Leser fragt sich vielleicht, warum an dieser Stelle überhaupt von einem Texteditor die Rede ist bzw. ob eine grafische Darstellung des Design, so wie es entwickelt wurde, nicht praktischer ist. Das Problem hierbei ist,dass der Frontend-Entwickler eventuell Markierungen an Stellen des HTML-Codes vornehmen kann, die in einem WYSIWYG-Editor nicht zugänglich wären, da sie nicht direkt sichtbar sind.

Module für Problemlösungen

Für die Wiederverwendung von Problemlösungen muss, wie in den Anforderungen erörtert, eine modulare Struktur erschaffen werden, welche es ermöglicht, neue

Module für Problemlösungen an das System „anzuschliessen“. Lösungsmodule sollen für ein bestimmtes Problem anhand des vorliegenden Designs und der gegebenen Markierungen den erforderlichen Code generieren.

Zielsprachen

Es soll gewährleistet sein, dass das System Skripte in mehreren Zielsprachen generieren kann. Hier gäbe es prinzipiell 3 Wege.

- Jedes Problemlösungsmodul gibt bekannt, welche Sprache es unterstützt. Der Benutzer des System selektiert die Problemlösungsmodule so, dass alle zu benutzenden Module auch seine gewünschte Zielsprache unterstützen. Wird im System die Generierungsmethode des Moduls aufgerufen, so wird die Zielsprache an diese Methode übergeben und das Modul ruft intern die entsprechende Methode auf, welche den gewünschten Code in der gewünschten Sprache erzeugt. Man erhielte also für jede unterstütze Zielsprache eine Lösungsmethode für ein und dasselbe Problem.

- Man erstellt eine zusätzlich Schnittstelle, welche rein für die Unterstützung der einzelnen Zielsprachen zuständig ist. Hier werden sprachunterstützende Module angeschlossen. Diese geben auf Anfrage bekannt, wie ein bestimmtes Programm-Konstrukt in ihrer unterstützten Sprache aussieht. Die Lösungsmodule würden auf solche Konstrukte anfordern, wenn sie sie benötigen. Beispielsweise fordert ein Problemlösungsmodul ein Sprachmodul auf, ihm die Syntax für eine Wertzuweisung zu erstellen, indem es ihm den Variablennamen und den Wert (welcher wiederum ein Ausdruck sein kann) übermittelt. Zurück kommt immer eine Zeichenkette, die in den Scriptcode eingefügt werden kann. Ein ähnlicher Ansatz wäre die Generierung von einem Zwischencode, welcher in einem weiteren Schritt von einem Sprachmodul übersetzt wird.

Die erste Möglichkeit birgt das Risiko, dass Entwickler von neuen Modulen sich auf eine bestimmte Zielsprache „einschiessen“, die sie selbst am meisten benutzen. Bei der Realisierung von Problemlösungen durch die 2. Möglichkeit, muss jede Problemlösung nicht programmiert werden, sondern die Generierung des erforderlichen Codes muss programmiert werden. Dies geschieht zum Beispiel durch eine Folge von Methodenaufrufen an ein Sprachmodul. Dies ist natürlich für Entwickler eines Problemlösungsmodules gewöhnungsbedürftig, da man gezwungenermaßen etwas „um die Ecke“ denken muss. Vor allen Dingen bei Verschachtelungen von Sprachelementen (Geklammerte Ausdrücke, Befehlsblöcke etc.) wird die Programmierung ziemlich anstrengend bzw. konfus. Ein weiteres Problem besteht darin,dass es nicht möglich ist, einen einzigen Befehlssatz kompakt zu halten, wenn dieser mehrere unterschiedliche Sprachen abbilden können soll und je komplexer dieser Befehlssatz aussieht, desto schwieriger wird natürlich auch

seine praktische Anwendung. Daher macht es Sinn, sich für die erste Möglichkeit zu entscheiden und mit dem Nachteil zu leben, daß nicht alle Lösungsmodule alle Sprachen beherrschen. Denn wenn die Entwicklung solcher Module dadurch erheblich leichter ist, als auf dem 2. Weg, dürften sich mehr Entwickler finden, die solche Module entwickeln.

Projektverwaltung

Neben dem Editor gibt es in dem angestrebten System eine Projektverwaltung. Einzelne Projekte kapseln hier für den Entwickler bestimmte projektbezogene Voreinstellungen wie die gewünschte Zielsprache oder Verbindungsdaten für die Datenbank. Die Projektverwaltung wird also dem Entwickler ermöglichen, diese Einstellungen einmal an einer Stelle zu unternehmen, so dass diese Einstellungen für alle in dem Projekt befindlichen Dateien gelten. Die Entwicklung dieser Projektverwaltung hat jedoch vorerst niedrigere Priorität. Kern des Systems soll die Code-Generierung sein.

Benutzung des Systems

Abbildung 4.3 verschafft einen Überblick, über die in den vergangen Kapiteln enthaltenen Anwendungsfälle bei der Erstellung von Web-Frontends mit einer verbesserten Unterstützung.

Das Sequenzdiagramm in Abbildung 4.4 soll den Ablauf der einzelnen Arbeitsschritte während der Frontend-Entwicklung mit diesem System erklären. Hier wurde allerdings die Erstellung und Konfiguration des Web-Frontend-Systems stark vereinfacht, da dies ja nicht der Kern der Sache ist.

4.2 Identifizierte Objektklassen

Eine der Kernklassen des entstehenden Systems ist sicherlich der Texteditor. Ein Editor sollte Markierungen unterschiedlicher Typen setzen und lesen (auch darstellen) können. Beim Laden von Dateien, müssen Markierungen aus Kommentaren ausgelesen werden und die Kommentare im angezeigten HTML-Code ausgeschnitten werden. Analog müssen vorhandene Markierungen im Editor vor dem Speichern wieder in Kommentare umgesetzt werden. Die Editor-Klasse enthält deshalb Methoden für das Öffnen, Speichern und Schliessen von HTML-Dateien.

Das Extrahieren und Einsetzen von Markierungen den HTML-Code übernimmt die Design-Klasse. Diese kapselt den HTML-Code und alle darin enthaltenen Markierungen. Der Editor instantiiert immer dann, wenn er eine HTML-Datei öffnet, dieses Design. Als nächstes wird das Design alle im Text enthaltenden Kommentare auslesen und versuchen, diese als Markierungen zu interpretieren. Anschließend werden die Kommentare entfernt.

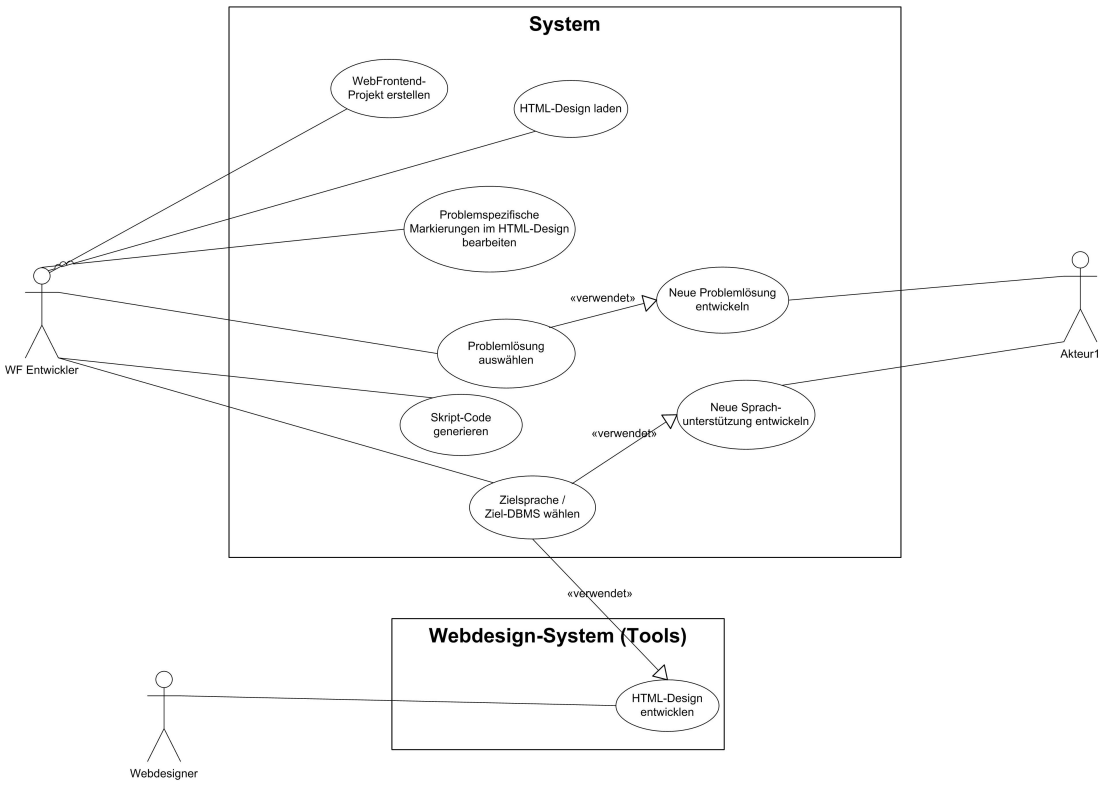

Abbildung 4.3: Anwendungsfälle für das geforderte System

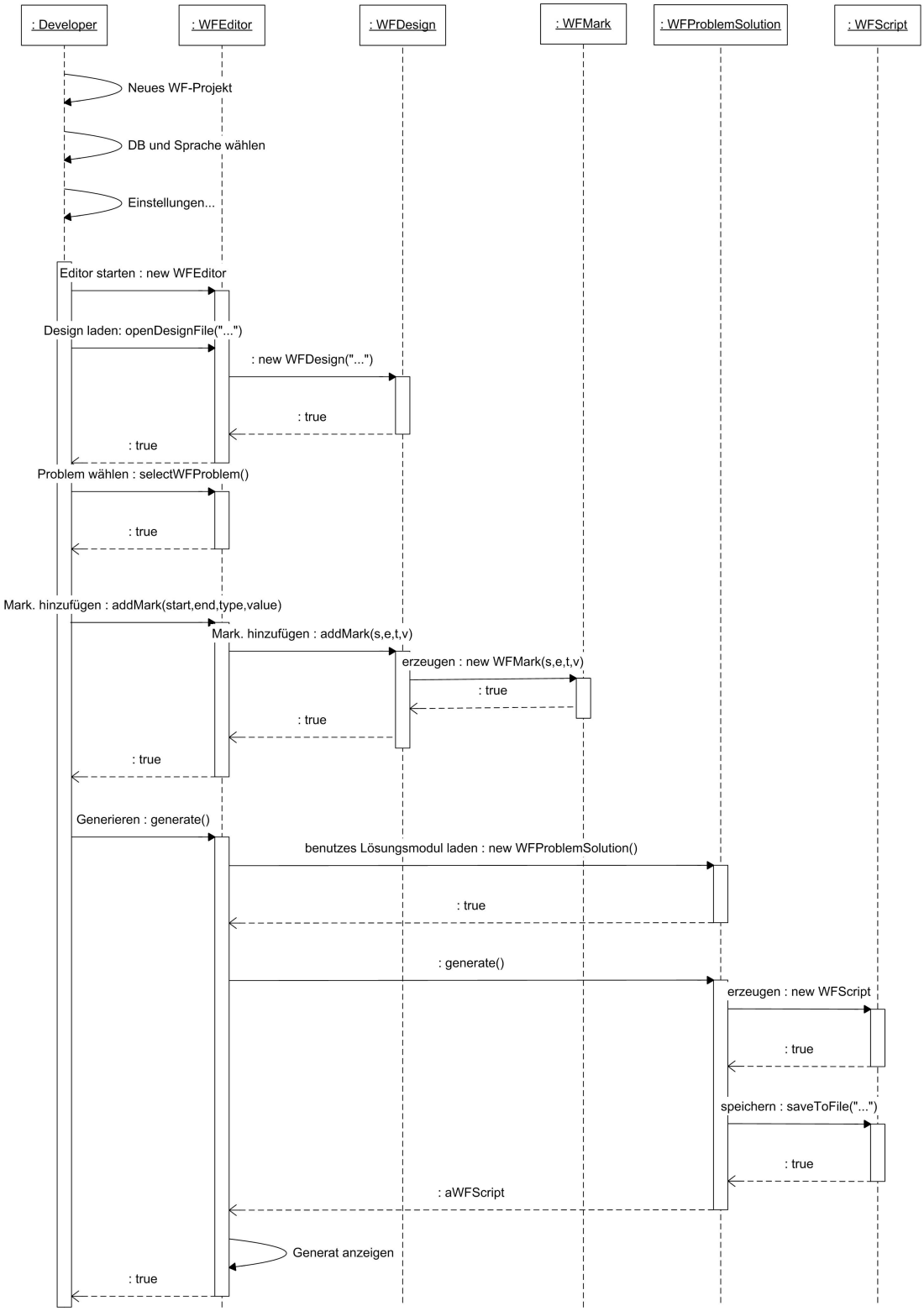

Abbildung 4.4: Eine mögliche Sequenz während der Frontend-Entwicklung

Alle Markierungen werden in einer eigenen Datenstruktur (Klasse) realisiert. Die Markierungsobjekte werden in dem Design-Object aggregiert. Diese Markierungsobjekte können dann auf Anfrage an die einzelnen Lösungsmodule gereicht werden. Was die unterschiedlichen Typisierungen von Markierungen zu bedeuten haben, ist für den Editor nicht wichtig. Dies ist nur für die einzelnen Lösungsmodule interessant. Lösungsmodule werden einfach dem Design mitteilen,dass Sie gern alle Markierungen vom Typ „X“ erhalten möchten. Letztendlich wird es deshalb reichen, die unterschiedlichen Typen einer Markierung als eindeutigen String festzuhalten, denn die Bedeutung eines Markierungstypes bedarf außer der Typidentifizierung (durch einen String) keiner weiteren speziellen Daten, welche es erforderlich machen würden, unterschiedliche Markierungsklassen zu entwerfen.

Jedes Lösungsmodul erbt von einer abstrakten Klasse für Lösungsmodule. Zum Generieren des Codes, welcher das eigentliche Problem lösen soll, besitzt jedes Lösungsmodul eine eigene Implementation der „generate“-Methode. Die „generate“-Methoden aller relevanten Lösungsmodule wird vom Editor aufgerufen. Bevor dieser das tut, ruft er die „canGenerate“-Methode auf. Hier prüft das Lösungsmodul, ob es mit den aktuellen Daten (Markierungen), welche im Design enthalten sind in der Lage ist, den Skriptcode für die Problemlösung zu generieren. Beispielsweise wäre hier zu prüfen, ob bestimmte Markierungen überhaupt vorhanden sind oder ob sich bestimmte Markierung innerhalb oder außerhalb anderer Markierungen befinden.

Alle beschriebenen Klassen und deren Beziehung zueinander sind in Abbildung 4.5 noch einmal aufgezeigt.

4.3 Das System im Überblick

Aus dem Anwendungsfalldiagramm 4.3 ist ersichtlich, wer vor diesem System sitzt. Es ist nicht der Webdesigner, denn seine Arbeit wird zuvor an anderer Stelle getan. Es ist der Frontend-Entwickler. Dieser soll nun statt Funktionalität in das Design zu programmieren, für wiederkehrende Probleme das neue Code-Generierungssystem nebst Editor benutzen. Das Vorgehen, in einem Editor etwas auszuarbeiten und es anschließend von derselben Applikation in etwas ausführbares umzusetzen erinnert hier sehr an die Arbeit mit einer IDE[1]. Tatsächlich stimmen die Anforderungen an eine IDE mit denen an dieses System prinziell überein. Es gibt einen Quelltext-Editor, eine Projektverwaltung und schließlich die Code-Generierung, welche als Pendant zum Compiler einer IDE verstanden werden kann.

Nun wäre es möglich, solch eine IDE neu zu entwickeln. Dies würde aber unnötig Zeit in Anspruch nehmen, da man das ganze Rad (Fenster-Erzeugung,

[1](Integrated Development Environment), Programmierumgebung, in der Quelltext-Editor, Compiler/Linker, Debugger und Projektverwaltung in einer Anwendung untergebracht sind. Nähere Infos siehe [Aca03]

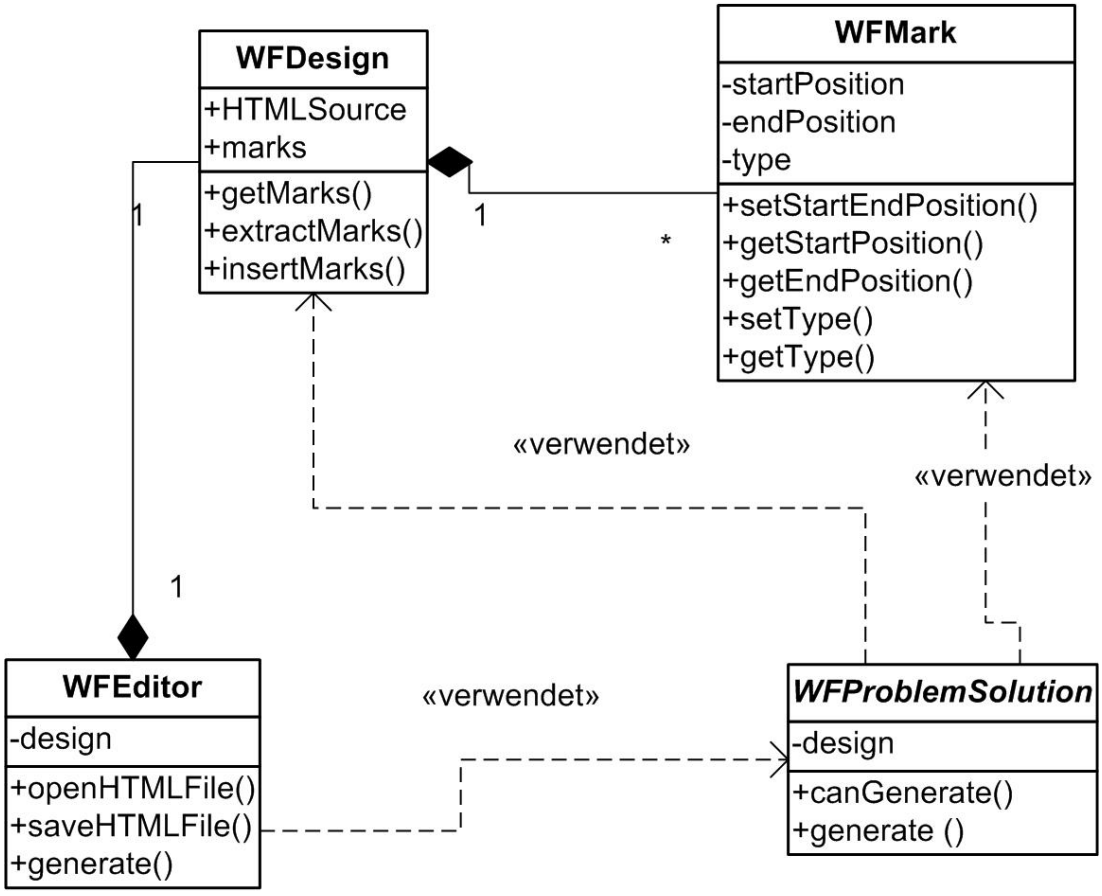

Abbildung 4.5: Die Kernklassen des neuen Systems

Projektverwaltung, Texteditor etc.) neu erfinden müsste. Daher zwängt sich die Überlegung auf, nach einer vorhandenen IDE zu suchen, welche es erlaubt, sich um die benötigten Module erweitern zu lassen. Eine solche IDE hätte nicht nur den Vorteil, dass man das Rad nicht neu erfinden muss, sondern auch,dass der Frontend-Entwickler sonstige Aufgaben (zum Beispiel Programmierung im Backend-Bereich oder Vervollständigung von generiertem Skript-Code) ggf. in derselben Umgebung tätigen könnte, in der er die Code-Generierung nutzt. Somit wäre auch die Projektverwaltung automatisch aufgabenübergreifend nutzbar. Eine solche IDE, die genügend Flexibilität bietet, soll im folgenden Abschnitt vorgestellt werden.

Kapitel 5

Konkretisiertes Design und Realisierung des Systems

5.1 Die Eclipse-Umgebung

Eclipse (s. [Con03]) ist ein Open-Source Projekt. Die Entwicklungsumgebung Eclipse basiert grundlegend auf einem Plug-in-Mechanismus. Eclipse ist von unterster Ebene an mit diesem Mechanismus aufgebaut. Zwischen der Java-Virtual-Machine[1] und den Plug-ins befindet sich lediglich eine unaustauschbare Komponente. Die sogenannte „Platform Runtime“, welche für das Laden und Ausführen aller Plug-ins verantwortlich ist. Alles, was über diesem festen Kern sitzt, ist bereits variabel. In dem ausgelieferten Eclipse-System von IBM befindet sich bereits eine Menge von Plug-ins, welche eine komplette Java-Entwicklungsumgebung darstellen. Startet man Eclipse, so sieht es also auf den ersten Blick aus, wie eine gewöhnliche IDE. Diese Entwicklungsumgebung kann man jetzt nach Belieben durch Plug-ins (entweder eigenentwickelte oder von Fremdherstellern) erweitern. Jedes Plug-in kann dabei auf verschiedensten Ebenen des Eclipse-Systems Erweiterungen anbieten. Auf diese Weise stellt Eclipse ein sehr flexibles System dar, welches sich nicht nur ständig weiterentwickelt (dadurch, dass Menschen auf der ganzen Welt Plug-ins entwickeln können) sondern sich auch noch individuellen Szenarien individuell anpassen kann. Das ist auch nötig, da es keinen großen Sinn macht, eine Entwicklungsumgebung erschaffen zu wollen, die für alle Zwecke und jeden Entwickler perfekt ist. Allerdings ist die gezielte Optimierung für konkretere Zwecke durchaus sinnvoll und möglich. Die Entwicklungssprache für Erweiterungen dieses Systems ist auf Java festgelegt. Erweiterungen selbst können aber die Unterstützung von weiteren Programmiersprachen bereitstellen, so dass Eclipse als Entwicklungsumgebung für C++ oder ähnliches eingesetzt werden könnte. Aufgrund des Plug-in-Prinzips wird Eclipse auch als PDE (Plug-in Development Environment) bezeichnet.

[1] Interpreter für die Programmiersprache Java. (s. [Mic04])

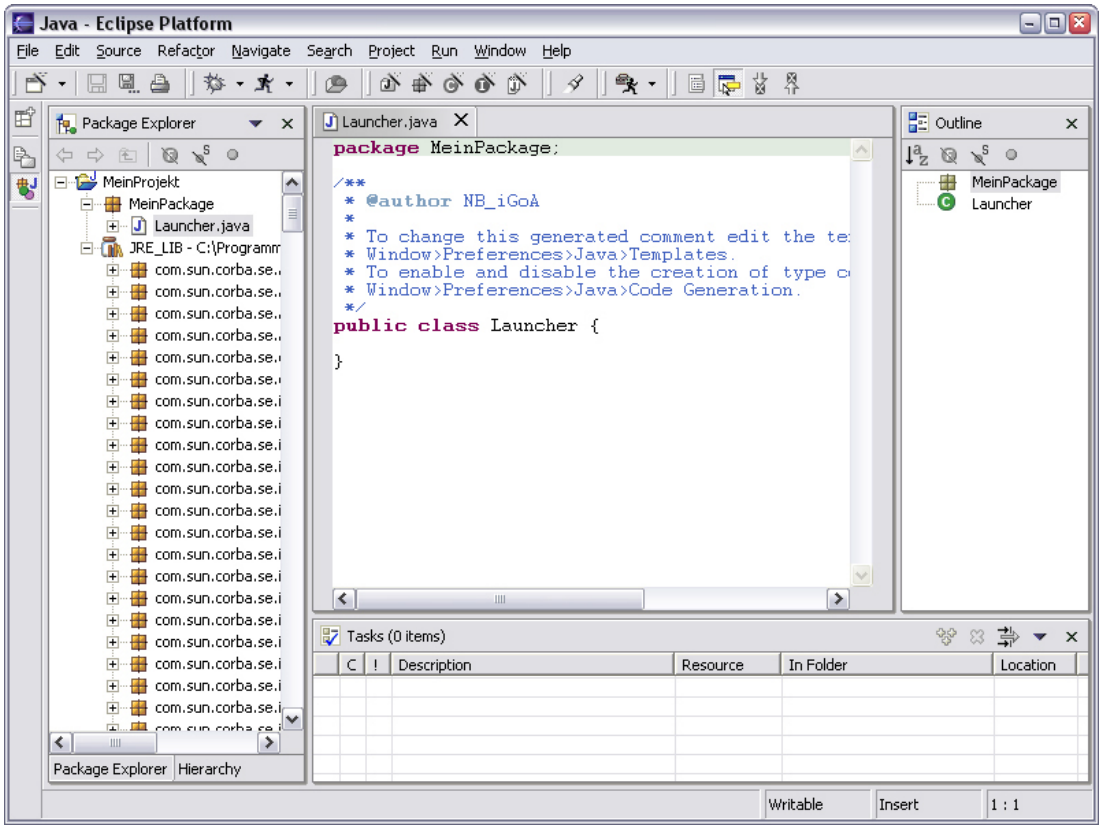

Abbildung 5.1: Die Eclipse-Oberfläche

Eclipse äußerlich

Installiert der Benutzer Eclipse auf seinem Rechner und startet zum erstenmal, so offenbart sich ihm auf den ersten Blick eine IDE, wie sie einem heutzutage bekannt sind (siehe Abb. 5.1). Links befindet sich eine Projektordner- oder Package-Sicht, in der Mitte ein Editor (falls eine Datei geöffnet ist), eventuell rechts ein Outline-View Methoden, Instanzvariablen usw.) und eventuell unten eine Konsole mit Fehlermeldungen oder anderen Ausgaben eines laufenden oder abgelaufenen Java-Programmes. Oben befindet sich die Hauptmenüleiste. Darunter eine Symbolleiste.

Die Anordnung der verschiedenen Sichten und Editoren kann vom Benutzer frei umgestaltet werden. Per Drag-And-Drop können die einzelnen Elemente überlappend, nebeneinander und übereinander angeordnet werden. So könnte sich also beispielsweise der Package Explorer auf der rechten Seite befinden und der Outline-View auf der linken. Die Konsole kann über dem Editor sein oder auf diesem (Der Benutzer kann dann zwischen beiden hin- und herschalten). Diese Oberfläche von Eclipse wird als Workbench bezeichnet. Als Workspace wird der Teil des Dateisystems bezeichnet, in dem der Benutzer seine Projekte speichert. Das Besondere bei Eclipse ist, daß all diese offensichtlichen Elemente der Umgebung, sowie die nicht offensichtlichen Elemente veränderbar, erweiterbar und austauschbar sind. Entwickler kann beispielsweise den Package Explorer durch eine Eigenentwicklung ersetzen, welche sich besser seinen Anforderungen anpasst. Ein anderes Beispiel ist die Unterstützung neuer Dateitypen durch neue Editoren. Diese Möglichkeiten bestehen dadurch, dass all diese Elemente der Entwicklungsumgebung Erweiterungen bzw. Plug-ins sind (Ein Plug-in kann mehrere Erweiterungen anbieten). Alle Editoren (für alle unterstützten Dateitypen) sind Erweiterungen des Systems, welche nahtlos integriert sind. Das gleiche gilt für alle oberflächlichen Teile des System, wie Views und Buttons in der Symbolleiste, sowie für alle nicht oberflächlichen Teile, wie Compiler, Debugger, Refactoringfunktionen etc.. Wie dies technisch möglich ist, soll im folgenden Kapitel erörtert werden.

Die Technik dahinter

Der Erweiterungsmechanismus

Der Erweiterungsmechanismus von Eclipse basiert im Grunde darauf, dass eine Erweiterung in Java geschrieben wird und an den dafür vorgesehenen Punkt in der Umgebung angeschlossen wird. Solche Erweiterungen werden in Plug-ins gekapselt. Ein Plug-in kann eine oder mehrere Erweiterungen anbieten. Gleichzeitig kann das Plug-in neue Punkte definieren, an denen andere Plug-ins ihre Erweiterungen anschließen können. So entsteht ein Netzwerk von Erweiterungen, in dem Entwickler neue Erweiterungen auf bestehenden aufbauen können oder Erweiterungen auf beliebig tieferer Ebene anbringen können.

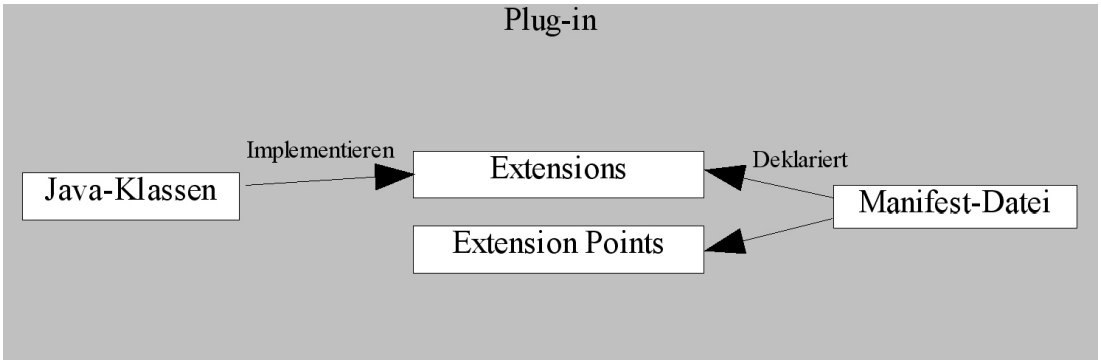

Abbildung 5.2: Zusammensetzung von Plug-ins

Erweiterungen und Erweiterungspunkte

Das Eclipse-System wird durch Erweiterungen (Extensions) aufgebaut. Jede Erweiterung des Systems knüpft an einen definierten Erweiterungs-Punkt (Extension-Points). Jedes Plug-in kann eine oder mehrere Erweiterungen anbieten. Dabei muss definiert werden, an welche Erweiterungs-Punkte Erweiterungen angeknüpft werden. Es ist auch möglich ein und dieselbe Erweiterung gleichzeitig an mehrere Erweiterungs-Punkte zu binden. Dies ist beispielsweise sinnvoll, wenn eine bestimmte Funktionalität von mehreren Stellen aus - z.B. über Kontext-Menüs und über das Hauptmenü - erreichbar sein soll. Jedes Plug-in kann seinerseits auch Erweiterungspunkte definieren, an die weitere Plug-ins anknüpfen können. Erweiterungspunkte werden durch einen einmaligen Namen (ID) definiert. Zusätzlich wird grundsätzlich ein Schema für neuartige Erweiterungspunkte angegeben. Das Schema, welches als XSD-oder als EXSD-Datei (s. [W3C03b])verfasst wird, enthält die Informationen darüber, welcher Art die Erweiterungen an diesem Punkt sein sollten. Beispielsweise gibt es den Erweiterungspunkt org.eclipse.-ui.editors an den, wie der Name verrät, Editoren angeschlossen werden können. Entsprechend gibt es für diesen Punkt ein Editor-Schema, in dem alle Angaben enthalten sind, die das Plug-in über die angeschlossenen Editoren wissen sollte.

Plug-ins

Ein Plug-in besteht aus zwei Teilen. Einerseits den Java-Klassen, welche die Funktionalität enthalten und andererseits eine sogenannte Manifest-Datei, welche in XML verfasst wird. Sie enthält eine eindeutige ID für dieses Plug-in und Informationen darüber, an welchen Erweiterungspunkten im Eclipse-System das Plug-in Erweiterungen bereitstellt und welche Klassen die Erweiterungen darstellen. In der Manifest-Datei werden zusätzlich Abhängigkeiten von anderen Plug-ins oder Bibliotheken festgehalten. (siehe Abb. 5.2)

Um Erweiterungen an einen bestimmten Erweiterungspunkt zu knüpfen, muss die Plug-in-ID des Plug-ins, welches den Erweiterungspunkt anbietet, zusammen

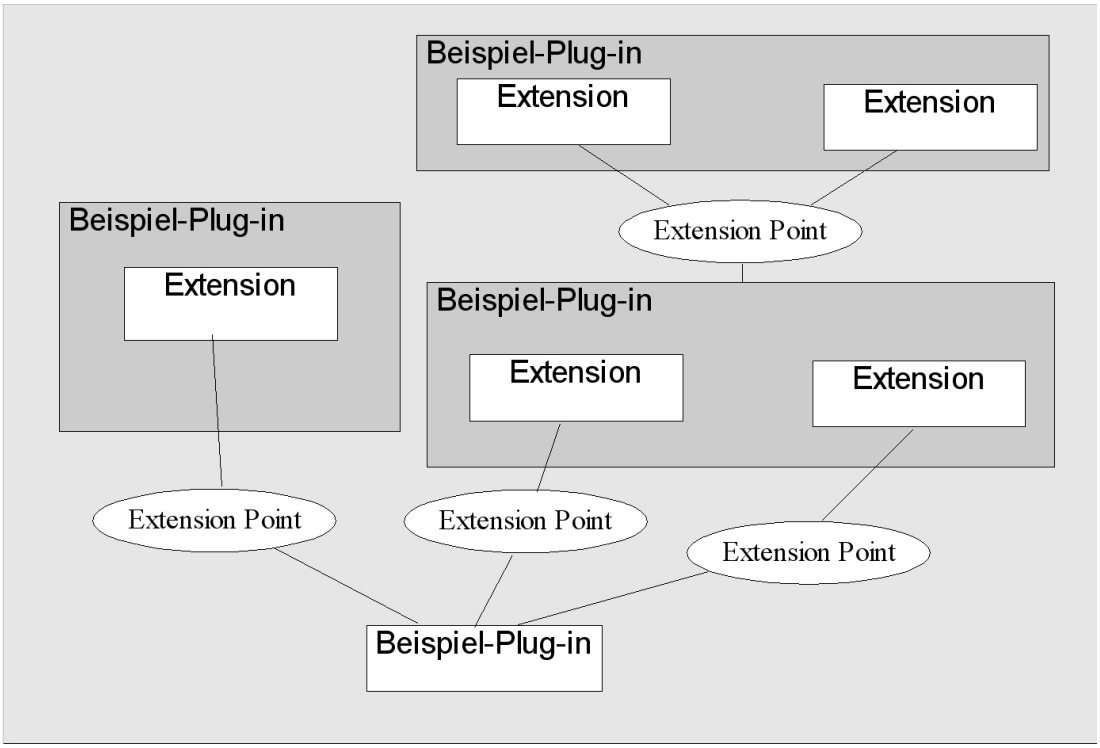

Abbildung 5.3: Zusammenspiel der Plug-ins

mit der ID des Punktes angegeben werden. Desweiteren muss das dem Schema des Erweiterungspunktes entsprechende XML-Tag aufgeführt werden und angegeben sein, welche Java-Klasse die Erweiterung beinhaltet.

Einbettung von Plug-ins

Im Eclipse-Ordner des Dateisystems gibt es einen Plugins-Ordner. In diesem befindet sich für jedes Plug-in ein Unterordner, welcher den Namen der Plug-in-ID trägt. In diesen Ordnern befinden sich die Java-Klassen als JAR-Dateien und die Manifest-Datei (hat immer den Namen Plugin.XML). Bei jedem Start von Eclipse werden alle Plug-ins, die sich im Plug-in-Ordner befinden, automatisch registriert und eingebunden. Soll ein neues Plug-in installiert werden, muss nach dem Erzeugen des Plug-in-Ordners, samt dessen Inhalt, lediglich Eclipse einmal neu gestartet werden. Bei Eclipse erfolgt das Laden von Plug-in-Klassen dynamisch nach Bedarf, um Ressourcen zu sparen. Dies wird durch die Auslagerung der Metainformationen über Plug-ins in die Manifest-Dateien unterstützt. Eclipse hat durch die Manifest-Dateien jederzeit alle nötigen Informationen über alle Plug-ins, ohne deren Java-Klassen geladen haben zu müssen.

Eclipse als Plattform für das Frontend-Entwicklungssystem

Zusammenfassend kann man festhalten,dass Eclipse durch seine offene und flexible Struktur nicht nur den Anforderungen aus Kapitel 4.3 genügt, sondern darüber hinaus das Potential zu einer verbreiteten Entwicklungsumgebung besitzt, welche den Entwicklungsprozess von Software komplett unterstützt, indem für alle üblichen Tätigkeiten während der Softwareentwicklung unterstützende Plug-ins entwickelt werden können.

Das Design des WebFrontend-Entwicklungssystemes als Erweiterung der Eclipse-Plattform ausfällt, soll in den folgendem Kapitel beschrieben werden.

5.2 Design der Eclipse-Lösung

Das Generierungssystem für Web-Frontends besteht im Kern aus einem Eclipse-Plug-in. Dieses bietet folgende Erweiterungen und Erweiterungspunkte an.

- Eine Erweiterung der Eclipse-Plattform in Form eines Editors für HTML-Designs. Dies wird das Instrument, mit dem der Web-Frontend-Entwickler arbeiten soll. Dieser lädt das HTML-Design, wählt die zu lösende Frontend-Aufgabe und setzt alle nötigen Kennzeichnungen im Design. Anschließend kann die Generierung gestartet werden. Hauptelement dieses Editors wird ein Textfeld, in dem der HTML-Quelltext angezeigt wird. Hierin nimmt der Entwickler die Markierungen vor.

- Erweiterungspunkt für Sprachunterstützungen. Neue Sprachunterstützungen können dann als Eclipse-Erweiterungen herausgegeben werden. Eine oder mehrere solcher Erweiterungen werden als Eclipse-Plug-in geliefert.

- Erweiterungspunkt für Problemlösungen. Hier gilt das gleiche, wie bei Sprachunterstützungen.

Der prinzielle Aufbau des entstehenden Systems lässt sich durch Abbildung 5.4 verdeutlichen.

Ein neuer Editor

In Eclipse gibt es die Möglichkeit neue Editoren[2] einzubetten, welche Dateien mit bestimmter Endung zugeordnet werden. Soll heißen, wenn der Benutzer im Project-Navigator auf eine Datei mit dieser Endung doppelklickt, so wird automatisch der dafür vorgesehene Editor geöffnet. Dabei kann es sich um einen

[2]Werkzeuge zum Bearbeiten von Dokumenten. Für unterschiedliche Dokumentarten (Texte, Bilder etc.) gibt es unterschiedliche Editoren

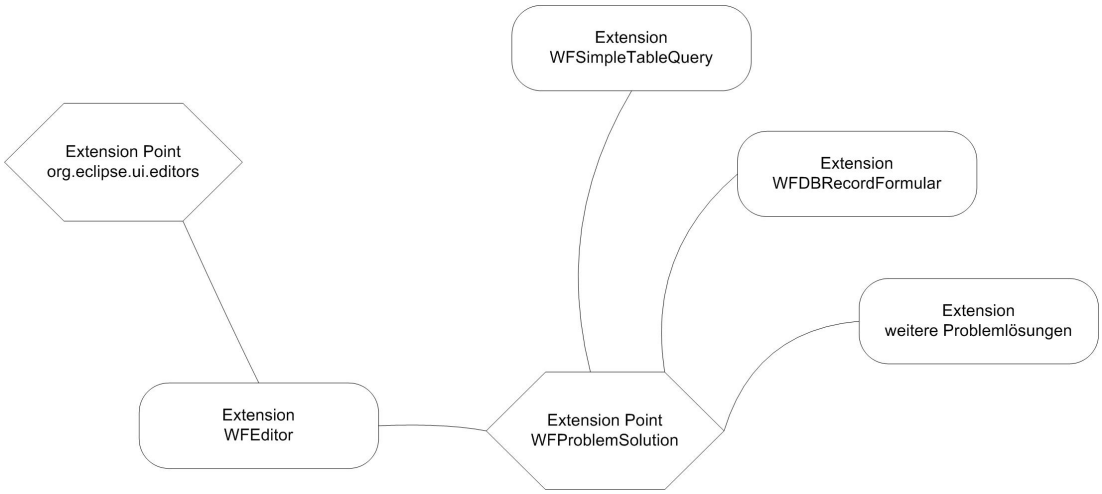

Abbildung 5.4: Aufbau der Eclipse-Lösung

Texteditor oder einen grafischen Editor handeln. Im Falle von Webfrontends, deren Design mit einem beliebigen Design-Tool erzeugt werden, macht ein Editor, der mit HTML-Dateien umgehen kann Sinn.

Der zu entwickelnde Editor soll ein sogenannter MultiPageEditor sein. Ein solcher ist ein Editor, welcher über mehrere, durch Registerkarten aufrufbare, Seiten mit Steuerelementen verfügen kann. In diesem Fall wird auf der ersten Seite der HTML-Quelltext des Designs angezeigt und auf der zweiten Seite die Lister der benutzten Problemlösungs-Plug-ins bzw. die Liste aller angebotenen Problemlösungen. Auf dieser Seite entscheidet der Frontend-Entwickler, welche Aufgabe die geladene Seite erfüllen soll.

Wie ein solcher MultiPageEditor in Eclipse aussieht, zeigt Abbildung 5.5. In der Abbildung ist gerade die Seite mit dem HTML-Source gewählt. Im umrandeten Bereich befinden sich die „Laschen“ der Registerkarten, mit denen man zwischen den Seiten des Editors hin- und herspringen kann. Ein Klick auf „Problem Solutions“ soll den Benutzer auf die Seite bringen, auf der er die angebotenen Problemlösungen auswählen kann.

Unterstützung von Markierungen

Auf der ersten Seite des MultiPageEditors, auf der der HTML-Quelltext angezeigt wird, soll der Frontend-Entwickler die Markierungen der signifikanten Stellen des Designs setzen, indem er mit der Maus oder per Tastatur einen Bereich markiert und über einen Button oder eine Tastenkombination eine Markierung in dieser Stelle setzen kann. In diesem Moment muss vom System nachgefragt werden, welchen Typ von Markierung er hier setzen möchte. Wie in Kapitel 4.1 erwähnt, gibt es unterschiedliche Typen von Markierungen. Diese werden von den einzelnen Problemlösungsmodulen definiert. Die Definition von neuen Markierungsty-

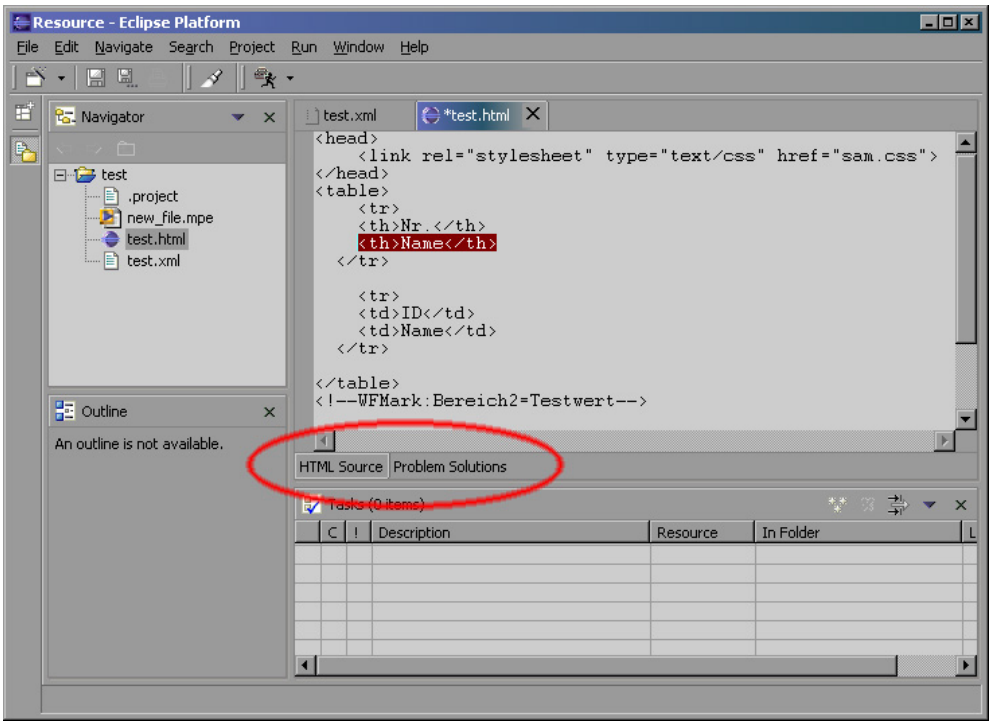

Abbildung 5.5: Ein MultiPageEditor in Eclipse

pen, welche von einem Problemlösungs-Plug-in benötigt werden, geschieht mit der „Anmeldung“ des Plug-ins in seiner Manifest-Datei. Beim Hinzufügen vom Markierungen soll der Editor aus einer Liste von Typen wählen lassen, welche durch die/das gewählte/n Problemlösungs-Plug-in/s definiert wurden. Zusätzlich soll es an dieser Stelle möglich sein, der Markierung einen Wert in Form einer Zeichenkette hinzuzufügen. Zum Beispiel ein SQL-Statement.

Diese Markierungen werden dann farblich hervorgehoben und beim Speichern des Designs, zusammen mit der Menge benutzter Problemlösungen, als Kommentar in der HTML-Datei gespeichert. Entsprechend wird beim Laden eines HTML-Designs überprüft, ob sich entsprechende Kommentare darin befinden und diese gegebenenfalls in farbliche Hervorhebungen überführt.

Erweiterungspunkte

Wie in 2.4 beschrieben, soll stets die Möglichkeit bestehen, neue Module mit wiederverwendbaren Problemlösungen für Webfrontends einzubetten. Um neue Module an das bestehende System „anzuschließen“ kann sich des Erweiterungsmechanismus von Eclipse bedient werden, indem ein neuer Erweiterungspunkt definiert wird. In der Liste der auszuwählenden Lösungsmodule werden dann automatisch alle angeschlosssenen Problemlösungs-Plug-ins angeboten.

Neben den deklarierten Erweiterungspunkten in der Manifest-Datei, muss

auch die abstrakten Klasse für Lösungsmodule entworfen werden. Dies geschieht nach dem Entwurf aus Kapitel 5.6.

Generierung

Beim Speichern des Dokumentes mit dem entwickelten Editor soll die Generierung angestoßen werden. Hierzu muss der Editor nacheinander alle gewählten Problemlösungsmodule dazu bringen, ihren Code in der gewünschten Sprache in das Design einzubetten. Das Ergebnis jeder einzelnen Code-Generierung wird dann vom Editor jeweils an das nächste Lösungsmodul weitergereicht, damit dieses seinen Code hinzufügt. Zum Schluss soll das Endergebnis dann in einer neuen Datei gespeichert werden und auf dem Webserver nutzbar sein. Zusätzlich sollte jedes Lösungsmodul die Möglichkeit besitzen, eigene neue Dateien zu erzeugen, die eventuell benötigt werden, damit ihre Lösung funktioniert. Das ist zum Beispiel dann sinnvoll, wenn die entstehende Anwendung auf mehrere Schichten verteilt werden soll und das Lösungsmodul die entsprechenden Quellcodes für eine weitere als die Präsentationsschicht erzeugen muss.

5.3 Realisierung

In den folgenden Abschnitten soll nun die konkrete Realisierung des Systems als Eclipse-Plug-ins erklärt werden. Zunächst wird beschrieben, wie die Basisklassen des Systems realisiert sind. Danach wird auf den zu entwickelnden Editor eingegangen. Hierzu noch einmal die geforderten Eigenschaften des Editors, wie sie im bisherigen Design erörtert wurden.

- Der Editor hat zwei Seiten. Eine mit einem Texteditor, welcher den HTML-Source enthält. Eine mit den zu benutzenden Problemlösungsmodulen
- Der Texteditor stellt Markierungen von dynamischen Stellen im HTML-Source mit einer anderen Hintergrundfarbe dar.
- Der Benutzer kann mit der Maus Text markieren und dort neue Markierungen von solchen Stellen einfügen.
- Der Benutzer kann den Markierungen einen Wert zuweisen und deren Typ bestimmen.
- Die möglichen Markierungstypen hängen von den auf Seite 2 des Editors gewählten Problemlösungsmodulen ab
- Der Benutzer kann Markierungen wieder entfernen

- Der Benutzer kann auf der zweiten Seite des Editor aus möglichen Problemlösungsmodulen diejenigen auswählen, welche er für seine aktuelle Aufgabe nutzen möchte. Dazu gibt es zwei Listen. Eine mit den möglichen Lösungen und eine mit den gewählten

- Der Benutzer kann in den Preferences von Eclipse spezielle Einstellungen bezüglich der Generator-Suite vornehmen. Zum Beispiel Zielsprache, Datenbank etc.

- Der Inhalt der Liste der angebotenen Problemlösungen ist von der gewählten Zielsprache abhängig. Es erscheinen nur Problemlösungsmodule, welche die aktuelle Zielsprache unterstützen.

- Beim Speichern von HTML-Dateien werden die benutzten Problemlösungen und alle Markierungen in HTML-Kommentare umgesetzt und mit gespeichert

- Beim Laden von HTML-Dateien werden eventuell vorhandene Kommentare als Markierungen bzw. gewählte Problemlösungen interpretiert und aus dem auf Seite 1 des Editors sichtbaren HTML-Code ausgeschnitten.

Realisierung des Haupt-Plug-ins

Zunächst werden die Basisklassen `WFMark` und `WFDesign` implementiert, damit die Grundstruktur der Daten des Systems vorhanden ist und weitere Systemteile, wie der entstehende Editor darauf bauen können. Erstere besitzt neben den Instanzvariablen `startPosition`, `endPosition`, `type`,`value` und `id`, den entsprechenden Gettern und Settern zwei Methoden, um einen HTML-Kommentar als Markierung zu interpretieren und um eine Markierung in einen HTML-Kommentar umzusetzten. Diese Methoden werden in der Klasse WFDesign verwendet. WFMark ist eher eine weniger funktionelle Klasse, sondern dient in erster Linie der strukturierten Datenhaltung.

Die Klasse WFDesign ist schon einen Tick komplexer. Ihre Realisierung lässt sich am besten anhand von Pseudo-Code (Listing 5.1) dokumentieren.

```
1  ...
2      private String HTMLSource;
3      private HashMap marks;
4      private Vector usedProblemSolutions;
5
6      // es folgen diverse Getter und Setter
7
8      public void extractMarks()
9      {
10         // finde alle Kommentare des HTML-Sources
```

```
11          // und versuche sie durch die Klasse WFMark
12          // zu interpretieren. Entferne anschlie{\ss}end
13          // die Kommentare aus dem Source
14
15          // lese ausserdem die "benutzten Problemloesungen"
16          // aus.
17      }
18
19      public void insertMarks()
20      {
21          // Erstelle fuer jedes WFMark Object einen
22          // HTML-Kommentar und fuege diesen an ent-
23          // sprechender Stelle in den Source ein
24
25          // schreibe ausserdem fuer alle benutzten Problemloesungen
26          // einen Kommentar in den HTML-Code
27      }
28
29      public void addProblemSolution()
30      {
31          // fuege eine zu benutzende Problemloesung zu
32          // der Liste benutzter Problemloesungen hinzu
33      }
34
35      public void removeProblemSolution()
36      {
37          // entferne eine Problemloesung aus der Liste
38      }
39
40  ...
```

Listing 5.1: Prinzielle Realisierung der WFDesign-Klasse

Wenn hier von „Problemlösungen“ gesprochen wird, so sind hier lediglich identifizierende Zeichenketten von eben solchen gemeint. Die tatsächliche Instanziierung von Problemlösungsmodulen findet erst später statt.

Anschließend wird das Haupt-Plug-in des Systems entwickelt. Dieses Plug-in enthält den besprochenen Editor, welcher an den entsprechenden Extension Point angeschlossen wird. Der Editor wird als Standard-Editor für HTML-Dateien registriert. Der für den neuen Editor relevante Ausschnitt der Manifest-Datei ist in Listing 5.2 zu sehen.

```
...
   <extension
         point="org.eclipse.ui.editors">
      <editor
            name="WebFrontend Editor"
            icon="icons/wficon.gif"
            extensions="html"
            class="de.igoa.WFGeneratorSuite.editors.WFEditor"
            id="de.igoa.WFGeneratorSuite.editors.WFEditor">
      </editor>
   </extension>
...
```

Listing 5.2: Einbindung des MultiPage-Editors in das Plug-in-Manifest

Der MultiPage-Editor

Die Eclipse-Platform stellt eine Klasse für Multipage-Editoren zur Verfügung, die nach eigenem Bedarf erweitert werden kann. Der Standard-MultiPage-Editor stellt eine Registerkartenkomponente zur Verfügung, auf denen die verschiedenen Seiten des Editors (die Pages) enthalten sind. Zu Beginn ist diese Komponente leer. Erst während der Initialisierung des Editors werden neue Seiten hinzugefügt und mit Eingabeelementen gefüllt. Diese Aufgaben werden durch die erbenden Klassen realisiert. Die Klasse `MultiPageEditorPart` enthält einige Methoden, welche durch einen neuen MultiPage-Editor überschrieben werden können. Die wichtigsten werden im folgenden erklärt. Abbildung 5.6 zeigt auf der linken Seite die Vererbungsbeziehung des neuen Multipage-Editors und die überlagerten Methoden (die restlichen Methoden wurden hier zur Übersichtlichkeit weggelassen). Die Erklärung der rechten Abbildungshälfte erfolgt noch.

`createPages()` wird beim öffnen des Editor automatisch aufgerufen und ist für die Erzeugung der Steuerelemente zuständig. In der neuen Klasse `WFEditor` werden 2 Pages erstellt und dem Editor hinzugefügt. Der zuständige Code sieht prinzipiell wie in Listing 5.3 folgt aus.

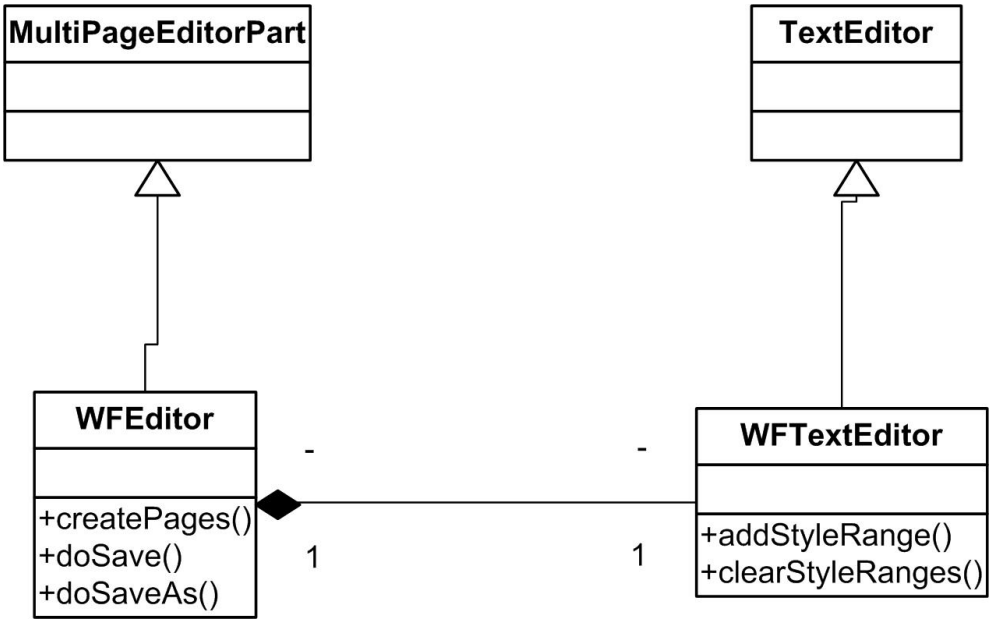

Abbildung 5.6: Vererbung von Standard-Editoren in Eclipse

```
...

void createPage0()
{
    textEditor = new WFTextEditor();
    textEditor.setParent(this);
    int index = addPage(textEditor, getEditorInput());
    setPageText(index, "HTML Source");
}

void createPage1()
{
    Composite composite = new Composite(getContainer(), SWT.NONE);

    // Steuerelemente erzeugen und dem Composite hinzufuegen
    // - Liste mit vorhandenen Problemloesungen
    // - Buttons zum aus- und abwaehlen von Problemloesungen
    // - Liste mit benutzten Problemloesungen
    // ...

    int index = addPage(composite);
    setPageText(index, "Problem Solutions");
}

...

protected void createPages() {

    createPage0();
    createPage1();

    // nachdem der Texteditor geoeffnet ist, kann auf den geladenen
    // HTML-Source zugegriffen werden
    String source = textEditor.getText();

    // damit wird ein WFDesign-Object erstellt
    // und die Markierungskommentare extahiert
    design = new WFDesign(source);
    design.extractMarks();
    ...
}

...
```

Listing 5.3: Implementation des MultiPage-Editors

Weitere wichtige Methoden sind `doSave` und `doSaveAs`. In diesen kann das Verhalten des MultiPage-Editors beim Speichern von Dokumenten definiert werden. In diesem konkreten Fall, sollen vor dem Speichern die Markierungen und die benutzten Problemlösungen wieder in HTML-Kommentare umgesetzt werden. Hierzu wird die Methode `insertMarks` des WFDesign-Objektes aufgerufen, der kommentierte HTML-Source dem Texteditor zugewiesen und anschließend erst gespeichert.

Auch für die Entwicklung von speziellen Texteditoren wie den geforderten bietet die Eclipse-Plattform Vorlagen. Der einfachste Texteditor den Eclipse mitbringt hat bereits einen vergleichbaren Funktionsumfang wie das von Microsoft Windows™ bekannte Notepad. Er erlaubt neben der einfachen Texteingabe das Markieren von Textbereichen und alle Zwischenablagefunktionen (Kopieren, Einfügen, Ausschneiden). Zudem können Textbereiche mit veränderten visuellen Eigenschaften, so genannte StyleRanges, definiert werden. Dies muss jedoch programmiert sein und lässt sich nicht standardmäßig durch Benutzereingaben hervorrufen. Diese StyleRanges können dazu benutzt werden, um Markierungen darzustellen. Nachdem der Grundriss des MultiPage-Editors realisiert ist, wird also ein standardmäßiger Texteditor vererbt und angepasst, sodass er den obigen Anforderungen genügt. So wird nun, wie im Design-Kapitel erörtert, auf einer Seite das Multipage-Editors der Texteditor für die Markierung des HTML-Source erstellt und auf der zweiten Seite die Auswahl der zur Verfügung stehenden Problemlösungsmodule. Der auf Seite 1 des WFEditors integrierte Texteditor wird, wie beschrieben, von einer Standard-TextEditor-Klasse vererbt und es werden Methoden zum hinzufügen von farblich markierten Bereichen hinzugefügt. Die Klasse `TextEditor` implementiert bereits einen kompletten Text-Editor. Über darin vorhandene Methoden können sogenannte `StyleRanges` erzeugt werden. Diese Objekte kapseln die erwähnte farbliche oder sonstige Hervorhebung innerhalb des Textes. Beispielsweise wird Syntax-Highlighting ebenfalls mit StyleRanges realisiert. Syntax-Highlighting an sich soll hier aber vorerst nicht „ablenken“, da dies den Rahmen der Arbeit sprengt. Die neue Klasse `WFTextEditor` besitzt nun eine Methode zum externen Hinzufügen von solchen StyleRanges. Diese Methode kann dann vom WFEditor (der neue MultiPage-Editor, welcher den TextEditor integriert hat) für jede Markierung aufgerufen werden. Das tut der WFEditor gleich nachdem er das WFDesign alle Markierungen hat auslesen lassen. Ändern sich die Markierungen (es wird eine hinzugefügt oder entfernt), so werden alle StyleRanges entfernt und erneut erstellt. Die Vererbung des TextEditors wird auf der rechten Seite der Abbildung 5.6 deutlich. Auch die Integration des Texteditors in den WFEditor ist hier eingezeichnet. Der WFEditor stellt auf diese Weise alle Markierungen im HTML-Code mit einer anderen Hintergrundfarbe (je nach Verschachtelungstiefe der Markierung) dar.

Markentyp und -Wert sind anhand der Hintergrundfarbe natürlich nicht erkennbar und eine permament sichtbare Einfügung der Informationen in textueller oder symbolischer Weise in den HTML-Code würde wieder die Übersicht-

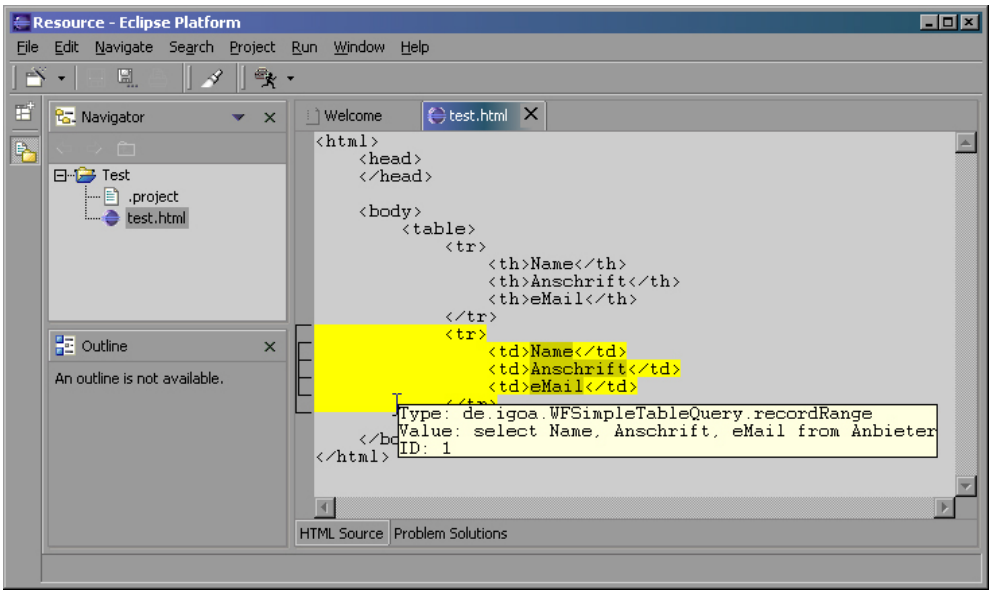

Abbildung 5.7: HTML-Source-Seite des WFEditors mit angezeigter Hoverbox

lichkeit nehmen. Daher wurde hier entschieden,dass diese Informationen genau dann erscheinen, wenn der Benuter mit der Maus auf einer solchen Markierung stehenbleibt. Der Benutzer kann dann einfach mit der Maus die Markierungen inspizieren. Hierzu implementiert der WFEditor eine Ereignisbehandlung für das sogenannte „Hovern“[3]. Die entsprechende Implementation ist in Listing 5.4 abstrakt skizziert.

```
1   void onHover(x,y)
2   {
3       // Frage den Texteditor, welcher Zeichenindex sich unter
4       // Position x,y befindet
5
6       // Suche Markierungen, die eventuell an dieser Position im
7       // Text sitzen.
8
9       if markierungGefunden()
10      {
11          // zeige eine Hover-Box mit ID, Typ und Wert
12          // des WFMark-Objektes
13      }
14  }
```

Listing 5.4: Hover-Funktion des Editors

Wie die zwei Seiten des so entstehenden MultiPage-Editors aussehen, zeigen

[3]Stehenbleiben des Mauszeigers auf einem bestimmten Objekt

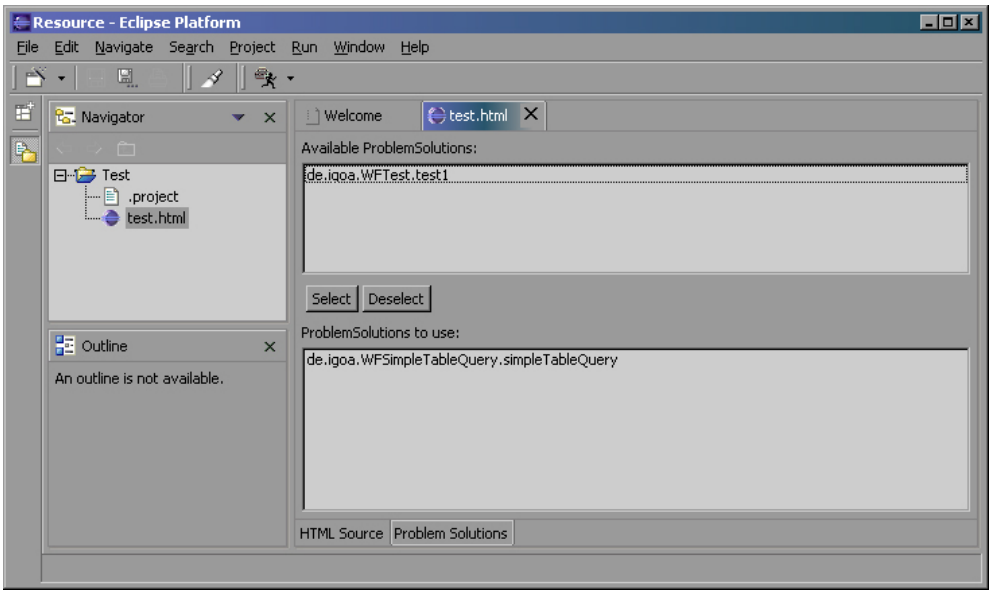

Abbildung 5.8: Auswahlseite für Problemlösungsmodule des WFEditors

die Screenshots 5.7 und 5.8. Letzterer wurde in dem Moment erstellt, als das Hover-Ereignis ausgelöst wurde und dementsprechend die Daten der Markierung über der sich die Maus befand angezeigt wurden.

Es werden drei neue Actions in Form eines selbstdefinierten Popup-Menüs für den Multipage-Editor erstellt. Eine zum Einfügen von Markierungen, eine zum Entfernen solcher und einen zum Generieren. Die Einfüge- und Entfernen-Aktionen sind jeweils nur aufrufbar, wenn ein Bereich im HTML-Code zuvor markiert wurde. Um ein solches Popup-Menü zu erstellen, kann ein entsprechender Extension Point der Eclipse Plattform benutzt werden. Den so erstellten Ausschnitt aus der Manifest-Datei des Hauptplug-ins zeigt Listing 5.5

```
1  <extension
2        point="org.eclipse.ui.popupMenus">
3     <viewerContribution
4        targetID="#TextEditorContext"
5        id="org.eclipse.ui.articles.action.contribution.popup.editor">
6        <action
7           accelerator="CTRL+SHIFT+M"
8           label="Remove Mark"
9           icon="icons/red_dot.gif"
10          class="de.igoa.WFGeneratorSuite.actions.WFRemoveMarkAction"
11          menubarPath="additions"
12          enablesFor="+"
13          id="de.igoa.WFGeneratorSuite.actions.popup.
               WFRemoveMarkAction">
14       </action>
```

```
15          <action
16              accelerator="CTRL+M"
17              label="Add Mark"
18              icon="icons/red_dot.gif"
19              class="de.igoa.WFGeneratorSuite.actions.WFAddMarkAction"
20              menubarPath="additions"
21              enablesFor="+"
22              id="de.igoa.WFGeneratorSuite.actions.popup.WFAddMarkAction">
23          </action>
24          <action
25              accelerator="CTRL+SHIFT+G"
26              label="Generate"
27              icon="icons/red_dot.gif"
28              class="de.igoa.WFGeneratorSuite.actions.WFGenerateAction"
29              menubarPath="additions"
30              id="de.igoa.WFGeneratorSuite.actions.popup.WFGenerateAction"
                    >
31          </action>
32      </viewerContribution>
33  </extension>
```

Listing 5.5: Definition des Popup-Menüs

`ViewerContribution` ist das von Eclipse definierte Element für den Erweiterungspunkt `org.eclipse.ui.popupMenus`. Das Element enthält eine Sequenz von `action`-Elementen. Diese enthalten die Attribute, um Tastenkürzel, angezeigter Text, ein Symbol, eine ID und schließlich die implementierende Klasse anzugeben. Zusätzlich kann mit dem Attribut `enablesFor` festgelegt werden, wann die Aktionen verfügbar sind. Hier wurde dieses Attribut für die Aktionen „Add Mark“ und „Remove Mark“ so gesetzt,dass die beiden Aktionen dann verfügbar sind, wenn ein oder mehrere Objekte im Editor markiert sind. Dadurch kann sichergestellt werden,dass das Hinzufügen und Entfernen von WFMarks nur dann möglich ist, wenn zuvor Text vom Benutzer markiert wurde. Eine entsprechende Überprüfung ist also seitens das Editors nicht nötig. Das angezeigte Popup-Menü ist in Abbildung 5.5 zu sehen.

Mit Aufruf der Einfüge-Aktion erscheint ein Dialog, in dem der Benutzer einen Markentyp auswählen muss. Die Auswahl an Markentypen hängt von den auf Seite 2 des Editor ausgewählten Problemlösungen ab. Ist zum Beispiel die Problemlösung für die einfache tabellarische Auflistung gewählt, so wäre ein möglicher Markentyp der Wiederholbereich für die Datensätze. Ein Markentyp definiert auch, ob eine Markierung einen Wert zugewiesen bekommen kann. Im Falle des Wiederholbereichs wäre ein sinnvoller Wert die SQL-Abrage. Diese muss ja irgendwo hinterlegt werden und passt hier am besten.

Wird die Entfernen-Aktion aufgerufen, so werden alle Markierungen aus dem Design entfernt, welche von der zuvor erfolgten Textmarkierung (zum Beispiel

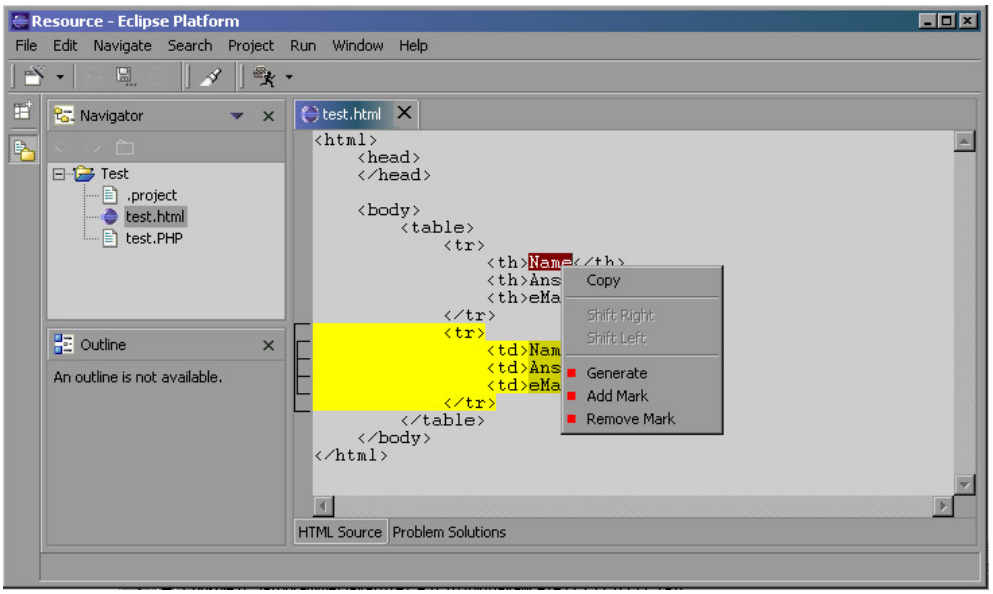

Abbildung 5.9: Das Popup-Menü des WFEditors

mit der Maus) komplett eingeschlossen waren.

Die globalen Einstellungen, wie Datenbankverbindung, Zielsprache oder Datenbanktyp werden über den Preferences-Mechanismus, der in Eclipse bereits vorhanden, ist realisiert. Dieser Mechanismus ermöglicht es mit einfachsten Schritten Variablen zu definieren, die automatisch persistent an ein Plug-in gebunden werden. Das heißt, wenn zum Beispiel eine Variable „DBHost“ für das entwickelte Haupt-Plug-in definiert ist, so wird ihr Wert immer automatisch geladen/gespeichert, wenn die Eclipse-Platform gestartet/beendet wird. Für die Veränderung der Variablenwerte durch den Benutzer muss lediglich ein kleiner Editor bereitgestellt werden. Dieser ist dann über das Hauptmenü „Window“-¿„Preferences“ erreichbar, wo sich auch alle anderen Einstellungen der Plattform vornehmen lassen. Nun werden für die Code-Generierung stets die hier vorgenommenen Einstellungen benutzt. Nähere Details über Preferences in Eclipse würden den Rahmen dieser Arbeit jedoch sprengen, da dies auch nicht Kern der Sache ist. Der interessierte Leser kann sich jedoch auf der Eclipse-Homepage (s. [Con03]) ausführlich darüber informieren. Der entwickelte Preferences-Editor stellt sich wie Abbildung 5.10 zeigt dar.

Realisierung der Problemlösungs-Schnittstelle

Für die Problemlösungsmodule wird ein neuer Extension Point entwickelt. Wie in Kapitel 5.1 erwähnt, werden neue Erweiterungspunkte stets durch XML-Schemas begleitet, welche die Elemente definieren, die eine Erweiterung ausmachen. In diesem Fall wird ein Schema für Problemlösungen definiert. Es enthält ein Pro-

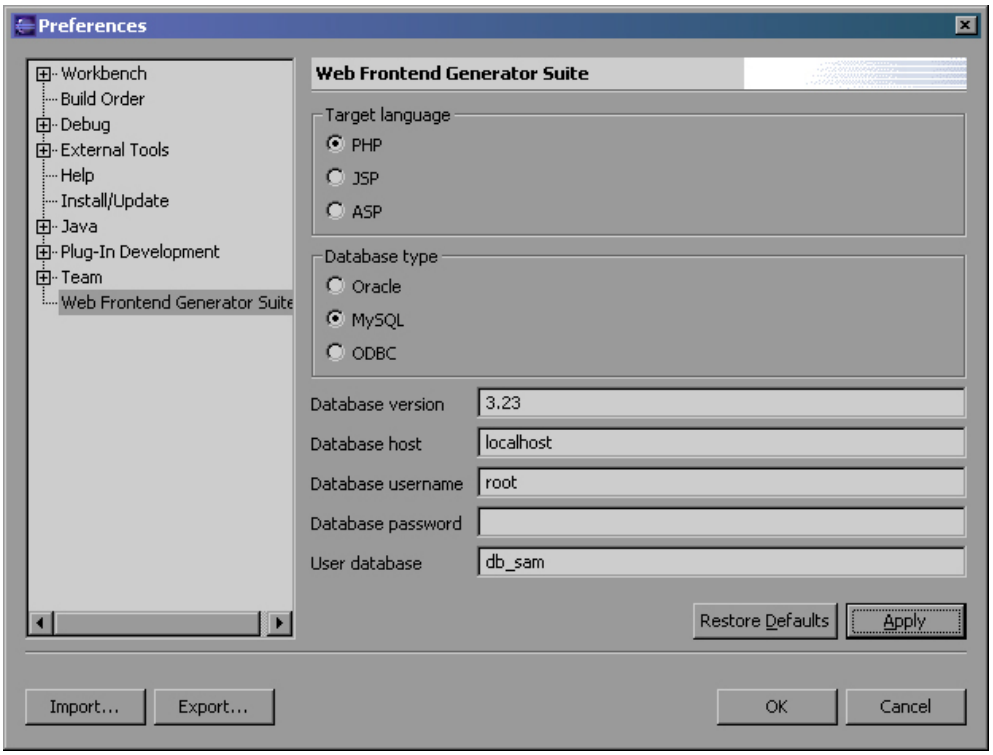

Abbildung 5.10: Preferences-Editor für die Generator-Suite

blemlösungs-Element mit einem Attribut für die eindeutigen Bezeichner der Problemlösung, die unterstützten Sprachen des Lösungsmoduls und die implementierende Klasse. Diese Klasse muss immer ein Nachfolger der abstrakten Klasse WFProblemSolution sein und die Methoden `canGenerate` und `generate` implementieren. Zusätzlich gibt es innerhalb des Problemlösungs-Elements eine Sequenz von Markierungstypen. Die Listings 5.6 und 5.7 zeigen das Schema und eine beispielhafte Anwendung (Ausschnitt aus einer Manifest-Datei) dessen, in der ein Problemlösung mit 2 Markierungstypen definiert wird.

```
1  <element name="extension">
2       <complexType>
3          <sequence>
4             <element ref="problemSolution" minOccurs="1" maxOccurs="
                  unbounded"/>
5          </sequence>
6          <attribute name="point" type="string" use="required" />
7       </complexType>
8    </element>
9
10   <element name="markType">
11      <complexType>
```

```
12          <attribute name="name" type="string" use="required" />
13          <attribute name="hasValue" type="boolean" use="required" />
14          <attribute name="defaultValue" type="string" />
15          <attribute name="description" type="string" />
16          <attribute name="id" type="string" />
17       </complexType>
18    </element>
19
20    <element name="problemSolution">
21       <complexType>
22          <sequence>
23             <element ref="markType" minOccurs="1" maxOccurs="unbounded"
                  />
24          </sequence>
25          <attribute name="id" type="string" use="required" />
26          <attribute name="name" type="string" />
27          <attribute name="description" type="string" />
28          <attribute name="class" type="string" use="required" />
29          <attribute name="languages" type="string" use="required" />
30       </complexType>
31    </element>
```

Listing 5.6: XSD-Schema des Extension Points

```
<extension
      point="de.igoa.WFGeneratorSuite.problemSolutions">
   <problemSolution
               id="de.igoa.WFSimpleTableQuery.simpleTableQuery"
         name="Simple table query"
         languages="PHP"
         class="de.igoa.WFSimpleTableQuery.SimpleTableQuerySolution">
        <markType
             name="Record Range"
             hasValue="true"
             defaultValue="select * from XYZ"
             description="The range which will be repeated for each
                 record"
             id="de.igoa.WFSimpleTableQuery.recordRange">
        </markType>
        <markType
             name="Record Field"
             hasValue="false"
             description="One Field of the current Record"
             id="de.igoa.WFSimpleTableQuery.recordField">
        </markType>
   </problemSolution>
</extension>
```

Listing 5.7: Nutzung des Extension Points

Nächster Schritt ist die Entwicklung erster Problemlösungserweiterungen und die Nutzung der Plug-in-Schnittstelle des Haupt-Plug-ins. Also zum Beispiel die Auflistung aller angeschlossenen Problemlösungen an den Erweiterungspunkt. Für das Auslesen der angeschlossenen Erweiterungen wird die PluginRegistry benutzt. Das ist eine von Eclipse bereitgestellte Klasse, welche die Anmeldung aller Plug-ins durch deren Manifest-Dateien kapselt. Über die PluginRegistry, welche als Singleton[4] realisiert ist, kann ein ExtensionPoint-Objekt von dem entsprechenden Erweiterungspunkt (durch die ID gekennzeichnet) erhalten werden. Dieses wiederum kann die angeschlossenen Erweiterungen als entsprechende Objekte zurückgeben und diese kapseln die in XML formulierten Elemente und Attribute. Über diesen Weg kann der Editor alle vorhandenen Problemlösungen im System „finden“. Listing 5.8 zeigt die Nutzung der PluginRegistry. Hier wird eine Funktion zur Verfügung gestellt, welche alle am System angeschlossenen Problemlösungs-Erweiterungen in Form ihrer IDs als Zeichenketten-Array zurückgibt. Die vorhandenen Problemlösungen werden dabei über die in den Preferences gewählte Zielsprache gefiltert. Es werden also lediglich solche Problemlösungen

[4]Eine einzelne Instanz einer Klasse, welche allerorts im System wiederverwendet wird. s. [GHJV93]

zurückgegeben, die die gewünschte Sprache unterstützen.

```
public String[] getAvailableWFProblemSolutionIDs() {
    // erstmal pruefen, welche Sprache gewaehlt ist (s. Preferences)
    String language = getPreferenceStore().getString(IWFPreference.
        LANGUAGE).toUpperCase();

    try
    {
        Vector answer = new Vector();

        // hier wird der Extension Point ausgelesen
        IExtension[] ext = Platform.getPluginRegistry()
                    .getExtensionPoint("de.igoa.WFGeneratorSuite.
                        problemSolutions")
                    .getExtensions();

        for (int i = 0; i < ext.length; i++)
        {
            IConfigurationElement[] psce = ext[i].
                getConfigurationElements();
            for (int j = 0; j < psce.length; j++)
            {
                // wenn das Loesungsmodul die gewuenschte Sprache
                    unterstuetzt hinzufuegen
                if (psce[j].getAttribute("languages").toUpperCase().
                    indexOf(language) >= 0)
                {
                    answer.add(psce[j].getAttribute("id"));
                }
            }
        }
        return (String[])answer.toArray(new String[ext.length]);
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null;
    }
}
```

Listing 5.8: Nutzung der PluginRegistry

Wird eine Problemlösung schließlich benutzt und die Generierung aufgerufen, so kann der Editor das „class“-Attribut der Erweiterungen benutzen, um daraus eine in Java nutzbare Instanz zu erzeugen. Da zuvor festgelegt wurde,dass alle Problemlösungen von der abstrakten Klasse WFProblemSolution erben, kann der

Editor nun eben jene Klasse als Schnittstelle nutzen, um die Methoden `generate` bzw. `canGenerate` aufrufen zu können. Das Ergebnis der Generierung einer Problemlösung (ein Design, mit eingefügtem generierten Code) wird vom Editor ggf. an die nächsten Problemlösungen weitergereicht. Dieser Vorgang ist im Haupt-Plug-in folgendermaßen realisiert:

```
1  public void generate() {
2      // alle benutzten Problemloesungen nehmen
3      String[] pss = design.getUsedProblemSolutions();
4
5      // gewuenschte Zielsprache auslesen
6      String language = WFGeneratorSuitePlugin.getDefault().
           getPreferenceStore().getString(IWFPreference.LANGUAGE);
7
8      // Geoeffnete Datei des Editors nehmen (HTML-Datei)
9      IFile inputFile = ((IFileEditorInput)getEditorInput()).getFile();
10
11     // Nun alle Problemloesungen durchgehen
12     for (int i = 0; i < pss.length; i++)
13     {
14         // Das XML-Element aus dem Manifest der Problemloesung
15         nehmen
16         // Dies wird von einer Methode des Haupt-Plug-ins
17         uebernommen
18         IConfigurationElement psce = WFGeneratorSuitePlugin.getDefault()
               .getWFProblemSolution(pss[i]);
19         Object pso = psce.createExecutableExtension("class");
20         WFProblemSolution ps = (WFProblemSolution)pso;
21         if (ps.canGenerate(tempDesign, language))
22         {
23             ps.generate(tempDesign, language, inputFile.getFullPath(),
                   WFGeneratorSuitePlugin.getDefault().getPreferenceStore())
                   ;
24         }
25     }
26
27     createNewFile(inputFile.getFullPath().removeFileExtension().
           addFileExtension(language),tempDesign.getHTMLSource());
28 }
```

Listing 5.9: Generierung durch alle benutzten Lösungsmodule

Der Code wurde etwas vereinfacht, da einige weitere Maßnahmen erforderlich waren, um die Funktionalität des Editor zu erhalten (Exception-Handling etc.). Diese Methode wird immer dann ausgeführt, wenn der Benutzer die Generierungs-Aktion aus dem Popup-Menü des Editors aufruft.

Realisierung von Problemlösungsmodulen

Die exemplarische Implementierung einer Problemlösung zeigt Listing 5.10.

```
1  public class TestProblemSolution extends WFProblemSolution {
2      public TestProblemSolution()
3      {
4          super();
5      }
6
7      public void generate(WFDesign design, String language, IPath
           sourcePath, IPreferenceStore pstore)
8      {
9          design.insertText(0,"ES HAT GEKLAPPT");
10     }
11
12     public boolean canGenerate(WFDesign design, String language)
13     {
14         return true;
15     }
16 }
```

Listing 5.10: Ein triviales Lösungsmodul

Die in diesem Listing implementierte Problemlösung benötigt keinerlei Markierungen und gibt daher beim Aufruf von `canGenerate` immer `TRUE` zurück. Zudem ignoriert diese Problemlösung die übergebene Sprache und alle Einstellungen (Datenbankeinstellungen etc.). Letztere werden an die Methode `generate` über das `IPreferenceStore`-Object übermittelt. Über dieses hat der Entwickler Zugriff auf die Einstellungen, die der Benutzer in seiner Eclipse-Umgebung vorgenommen hat. Das einzige was diese Problemlösung letztendlich tut ist,dass sie an den Anfang des HTML-Code den Satz „ES HAT GEKLAPPT“ einfügt. Dies soll lediglich zur Demonstration dienen und hat natürlich keinerlei praktischen Wert. Die der Arbeit beiliegende CD beinhaltet eine „echte“ Problemlösung, welche tatsächlich nutzbaren PHP-Code generiert. Der interessierte Leser kann sich den entsprechenden Source-Code darauf anschauen.

- Zu Beginn wird eine Datenbankverbindung aufgebaut
- Überall dort, wo der Entwickler eine Wiederholbereichs-Markierung (Markierung vom Typ de.igoa.WFSimpleTableQuery.recordRange) gesetzt hat wird eine Datenbankabfrage getätigt und eine Schleife gestartet
- In dieser werden überall dort die Datenbankfeld-Inhalte ausgegeben, wo der Entwickler eine entsprechende Markierung (de.igoa.WFSimpleTableQuery.recordField) vorgenommen hat.

Das Vorgehen entspricht dem, des in Kapitel 2.1 vorgestellten Beispiel-Skriptes. Eventuell fällt dem Leser bei der Einsicht des Lösungsmodul-Codes auf der CD auf, dass das generierte Skript hier stets eine MySQL-Datenbankverbindung aufbaut und den Datenbanktyp (und die Version) in den Preferences ignoriert. Korrekterweise müsste das Lösungsmodul tatsächlich für die unterschiedlichen Datanbanktypen, unterschiedlichen Code erzeugen. Das wäre gerade bei PHP wichtig, da hier für jeden Datenbanktypen ein anderer Befehlssatz vorhanden ist (warum auch immer). Der nötige Code für das „komplette“ Lösungsmodul würde dann aber auf jeden Fall den Rahmen dieser Arbeit sprengen. Der Leser möge das auf der CD befindliche Lösungsmodul daher als vereinfachte Darstellung betrachten.

Kapitel 6

Evaluation

Nachdem das Haupt-Plug-in und eine erste Problemlösung entwickelt wurden, soll diese nun einmal in ihrer praktischen Anwendung ausgewertet werden. Das Beispiel „Tabellarische Darstellung von Datenbankinhalten“ wurde in dieser Arbeit schon öfter gebracht, da die Anwendung diese Problemlösung auch sicherlich eine ist, an der sich das System und dessen Nutzen am besten demonstrieren lässt. Man stelle sich vor, man möchte, wie im SAM-Szenario beschrieben, eine Tabelle mit allen Anbietern darstellen. Nun liegt das Design in HTML vor und es wird das entsprechende Problemlösungsmodul eingesetzt. An den Stellen, an denen Feldinhalte angezeigt werden sollen, wurde im Design-Tool der Feldname hineingeschrieben. Mit der HTML-Datei, welche anschließend in das Generator-System eingeladen wird, wird nun verfahren, wie in den Beispielen vorheriger Kapitel: Zunächst wird in den Einstellungen des Systems die Zielsprache PHP und die Datenbank MySQL gewählt. Dann werden die für SAM benötigten Verbindungsdaten eingegeben. Anschließend wird die tabellarische Darstellung als Problemlösung ausgewählt. Der Wiederholbereich für jeden Datensatz wird mit der Maus markiert und dann wird eine Markierung vom Typ „Widerholbereich“ eingefügt. Als Wert bekommt sie das SQL-Statement „select ID, Name, Anschrift, Ort from Anbieter“ eingegeben. Nun werden die einzelnen Feldnamen markiert und bekommen den Markierungstyp „Datensatzfeld“. Nun kann generiert werden. Die entstandene PHP-Datei kann jetzt auf dem Webserver von SAM eingesetzt werden.

Dieses Vorgehen wurde im SAM-Projekt für einige Seiten des Auftrittes verfolgt, um zum Beispiel die Anbieter-Auflistung, die Medien-Auflistung oder die Auflistung der registrierten Benutzer zu realisieren. Der generierte Code hat erwartungsgemäß immer funktioniert. Die Handhabung des System gestaltete sich recht einfach, da ja, grob zusammengefasst, stets nur markiert werden musste. Wie in 2.4 erwähnt, soll das Generat nicht jede spezielle Funktion beinhalten, sondern als solide Basis für die letztendliche Lösung dienen. Ein Beispiel hierfür wär die Auflistung der Medien. Hier sollen die Bedingungen in der SQL-Abfrage dynamisch, abhängig von Eingaben des Endbenutzers, erzeugt werden. Diese Funk-

tionalität lässt sich nicht mit dem bisher entwickelten Lösungsmodul erreichen. Das würde auch nicht sonderlich viel Sinn machen, da die dynamische Zusammensetzung des SQL-Statements eine sehr spezielle Funktion ist, die außerhalb des SAM-Projektes kaum wiederverwendet werden würde. Dennoch stellte die Generator-Suite eine große Hilfe dar, da man sehr schnell den Grundaufbau jeder Seite fertigstellen konnte. Um die Arbeit mit dem System noch effektiver zu gestalten, wäre ein Lösungsmodul denkbar, welches für eventuelle Spezialfunktionen die Funktionsrümpfe und -aufrufe generiert. So könnte man etwa einen Feldnamen im Design markieren und durch einen neuen Markierungtypen festlegen,das statt den Feldinhalt auszugeben, eine Funktion mit dem Feldinhalt als Argument aufgerufen werden soll. Dort, wo sonst die Ausgabe des Feldes generiert wird, wird also ein Funktionsaufruf generiert. Mit dieser leicht zu realisierenden Verbesserung dürfte das Lösungsmodul einiges an Mächtigkeit gewinnen.

Über dieses eine Problemlösungmodul hinaus, werden im Folgenden die Vor- und Nachteile des Systems gegenüber den in Kapitel 2.2 vorgestellten vorhandenen Lösungen zur Entwicklung von Web-Frontends erörtert.

Der Vorteil dieses Systems gegenüber trivialen Code-Generatoren, die man kostenlos im Internet findet, ist klar. Das Aussehen von generierten Seiten ist bei diesen kaum individuell beeinflussbar. Die entwickelte Generator-Suite ist dagegen in der Lage, aufwendige selbsterstellte Design zu verarbeiten und die nötigen Code dort einzupflechten.

Auch gegenüber Reporting-Tools für Datenbanken hat das System den entscheidenden Vorteil,dass Web-Oberflächen erschaffen werden können, welche auch die Manipulation von Datenbankinhalten vorsehen und/oder spezielle Anwendungslogik besitzen. Allerdings ist dieser Vergleich zugegebenermaßen unfair, da Reporting-Tools auch gar nicht für derartige Anwendungen konzipiert sind, sondern, wie der Name verlauten lässt, zum leichten Erstellen von Datenbank-Berichten gedacht sind.

Im Vergleich zu Generatoren für „komplette Anwendungen“, hat die Generator-Suite den klaren Vorteil,dass auch hier das Design (ähnlich wie im vorletzten Absatz) frei definierbar ist. Diese Design-Freiheit hat natürlich den Preis,dass der Entwickler nicht mit ein paar Mausklicks eine komplett funktionierende Anwendung erhält. Spielt der Web-Frontend-Entwickler mit dem Gedanken auf die Schnelle eine Anwendung zu erschaffen, welche keine speziellen Funktionalitäten außer dem Auflisten, Bearbeiten, Einfügen und Entfernen von Datensätzen und eventuell einfacher Verknüpfungen von Datensätzen enthalten soll, so ist er eventuell mit einem solchen Generator gut bedient. Jedoch muss er dafür auf das individuelle Aussehen der entstehenden Anwendung verzichten. Wenn das Aussehen keine Rolle spielt, da die Anwendung beispielsweise nur intern benutzt wird, könnte eine automatische Komplett-Generierung die Lösung sein (vorrausgesetzt, es wird ein Generator-Tool für die gewünschte Zielsprache gefunden). Allerdings ist in den meisten Fällen davon auszugehen,dass eine Anwendung früher oder später mit speziellen Funktionen bestückt werden muss. Spätestens ab diesem

Punkt ist diese Generierungsart nicht mehr sehr hilfreich. Dann kann die Nutzung dieses Systems hilfreich sein.

Das Frontend-Entwicklungstool CodeCharge™ greift die Nachteile von den erwähnten Code-Generatoren auf, eleminiert diese und stellt eine Möglichkeit dar, um Web-Anwendungen mit eigenem Design und spezieller Funktionalität zu entwickeln. Damit ist dieses Tool in jedem Fall mächtiger als etwa ein Generator, welcher eine Komplettanwendung nach festem Schema ausspuckt. Dennoch finden sich, wie in Kapitel 2.3 beschrieben, auch bei CodeCharge™ einige Tücken mit denen Frontend-Entwickler zu kämpfen haben. Diese Nachteile führen direkt zu den Anforderungen an das nun entwickelte System. Es wird nun überprüft, ob das System den Anforderungen, welche im Kapitel 2.4 erörtert wurden genügt und damit wirklich eine lohnende Alternative gegenüber CodeCharge™ und den anderen Lösungsansätzen darstellt.

Das System sollt eine modulare, offene Struktur haben, damit es um Problemlösungen für wiederkehrende Probleme der Web-Frontend-Entwicklung erweitert werden kann und diese nahtlos integriert. Dank des Plug-in-Mechanismus von Eclipse ist dies gelungen. Mittels neuer Eclipse-Plug-ins können mit geringem Aufwand neue Problemlösungen in das System integriert werden. Allein durch die Speicherung der Manifest-Datei zusammen mit den implementierenden Java-Klassen im Plug-in-Verzeichnis vom Eclipse wird ein Problemlösungsplugin erkannt und dessen Erweiterungen können benutzt werden. Eine Problematik bei Systemen wie CodeCharge™ ist,dass immer viel Code generiert wird, der eventuell, aber eben nicht sicherlich, benötigt wird. Da CodeCharge™ nicht aufgabenorientiert konzipiert ist, muss der erzeugte Code sehr viel Funktionalität haben wovon meistens viele Teile niemals benutzt werden. Dadurch plustert sich das Generat auf und verbraucht bei der Ausführung mehr Resourcen. Zudem ist das Generat kaum leserlich und somit für eine eventuell notwendige manuelle Nachbearbeitung denkbar ungeeignet. Durch den aufgabenorientierten Ansatz der entwickelten Generator-Suite kann dieses Problem beseitigt werden. Der Frontend-Entwickler wählt ja direkt die Aufgabe aus, die er lösen möchte. Das entsprechende Lösungsmodul kann ganz gezielt Code generieren, der diese Aufgabe und nur diese löst. Dadurch ist das Generat besser leserlich und effizienter.

Beim Einsatz neuer Systeme kann es ein Nachteil sein,dass neue Dateiformate benutzt werden. Darin gespeicherte Informationen sind ohne das System meist nicht mehr lesbar. Bei einem eventuellen Umstieg zu einem späteren Zeitpunkt gehen diese Informationen dann verloren. Daher wurde auf neue Dateiformate soweit es möglich war verzichtet. Die wichtigen Informationen stecken alle in den vom System kommentierten Design-Dateien (meistens HTML-Dateien). Zudem sind die Informationen durch das Konzept der Markierungen relativ leicht entzifferbar. Ob dies ein großer Nutzen ist, sei einmal dahingestellt. Jedenfalls wäre der Entwickler in der Lage, eine eventuell beschädigte Datei manuell wieder herzustellen, was bei einem proprietären Dateiformat meistens nicht der Fall sein wird. An einer Stelle werden allerdings schon Informationen in nicht offensichtlicher

Weise gespeichert. Und zwar die Umgebungseinstellungen von Eclipse, zu denen auch die Preferences des Generator-Suite-Plug-ins zählen. Jedoch kann man wohl sagen, daß Informationen wie Datenbankverbindung, Zielsprache und so weiter wiederherstellbar wären, falls das Eclipse-Dateisystem beschädigt würden oder man sich später für eine andere Umgebung entscheiden würde.

Die Trennung zwischen Designentwicklung und Entwicklung der Anwendungslogik ist ein weiterer Punkt. Diese Trennung kann in kleineren Unternehmen ohne Bedeutung sein, da hier ein Entwickler oft sowieso mit beiden Aufgaben betraut ist. Jedoch ist dies bei mittelständischen Unternehmen nicht der Fall. Daher ist die getrennte Entwicklung, sofern sie von einem System ermöglicht wird, sehr angenehm. Beim Einsatz der Generator-Suite kann der Entwickler sein eigenes favorisiertes Design-Werkzeug nutzen, ohne sich um eventuell vorhandenen Skript-Code zu kümmern. Der Entwickler der Anwendungslogik kann dieses (komplexe) Design in den Editor laden und kann die Markierungen der Stellen mit dynamischen Inhalt vornehmen, indem er diese Stellen mit der Maus markiert. Er muss dort keinen Code einfügen, was schnell zu Unübersichtlichkeit führen kann, wenn sich HTML und Skript-Code vermischen. Auch für die spätere Nachbearbeitung des generierten Codes, welche ja für nicht wiederverwendbare Funtkionalität gar nicht auszuschließen ist, kann vorgesehen werden,dass dieser Code in vorgenerierte Funktionsrümpfe hineingeschrieben wird, die außerhalb des zentralen HTML-Skript-Code-Gewirrs stehen (zum Beispiel am Dateianfang). Das erhöht die Übersichtlichkeit während der Nachprogrammierung enorm. Der nächste Schritt in diese Richtung ist,dass Problemlösungsmodule die Entwicklung von mehrschichtigen Web-Anwendungen zulässt, indem sie anwendungslogische Funktionen lediglich als Delegationen in das Frontend-Skript generieren. Die Implementation der Funktion findet in einer anderen Datei statt, welche später auf der nächsten Anwendungsschicht ausgeführt wird. Ob die Funktion an sich nun auch generiert werden kann, hängt vom Einzelfall ab. Jedoch ist festzustellen,dass durch entsprechende Problemlösungsmodule eine mehrschichtige Anwendung entwickelt werden kann, was mit Tools wie CodeCharge™ nicht möglich ist.

Zur Verdeutlichung der Nutzung des Systems wird in folgendem Kapitel noch die Idee für ein potentielles zweites Problemlösungsmodul vorgestellt.

Weitere Problemlösungsmodule

Das zweite Lösungsmodul, welches für das System entwickelt werden könnte, wäre ein Formulargenerator, welcher Code erzeugt, der später zum Bearbeiten oder Einfügen neuer Datensätze eingesetzt werden kann. Im Web-Design-Tool würde das HTML-Formular ohne Eingabeelemente (Textfelder, Listboxen etc.) erstellt werden. Statt der Eingabeelemente würde der Feldname dort stehen. Im Editor des neuen Systems würde nun dort eine Markierung mit entsprechendem Markentyp (Textfeld, Listbox etc.) erfolgen. Zusätzlich wird das ganze Formular

als solches markiert und bekommt als Wert die SQL-Abfrage des Datensatzes. Das Lösungsmodul kann dann ein Skript generieren, welches ein Formular mit gefüllten Feldern darstellt. Ein weiterer Schritt wäre,dass das gleiche Lösungsmodul ein zusätzliches Skript zum Speichern der geänderten Daten generiert, sodass die Aktualisierung der Datenbank ebenfalls vorhanden ist. Mit diesem Lösungsmodule könnte sehr viel Arbeit gespart werden, denn in vielen mir bekannten Web-Projekten beansprucht die Entwicklung der Formulare und deren Funktionalität sehr viel Zeit. Auch hier gilt aber,dass die generierten Skript sich höchstwahrscheinlich nicht sofort nach dem Generierungsvorgang perfekt in das zu entwickelnde System einfügen. Allerdings ist es hier nicht sinnvoll zu versuchen, ein Alles-Könner-Skript zu erzeugen, welches sehr viel Overhead hat und eine Nachbearbeitung verkompliziert. Genauso wenig sinnvoll erscheint es natürlich auch ein Lösungsmodul für sehr spezielle Formulare zu entwickeln, wenn dieser Typ von Formularen nicht wiederverwendet werden kann.

Insgesamt betrachtet, ist mit diesem System ein nützliches Werkzeug für mittlere und kleine Web-Projekte entstanden. Die Generator-Suite wird mit hoher Wahrscheinlichkeit in weiteren Projekten der Firma OMD Outdoor eingesetzt werden und wird dort mit entsprechenden Lösungsmodulen einen vergleichsweise schnellen Entwicklungsfortschritt der Web-Frontends mit sich bringen.

Die intendierte Nutzung des Systems

Abbildung 4.1 zeigt den vorgesehenen Arbeitsfluss bei der Nutzung der Generator-Suite. Er beginnt mit der Erstellung des Design, welche außerhalb des Systems mit dem bevorzugten Tool des Web-Designers stattfindet. Darauf folgt das Laden der entstandenen HTML-Datei in den Editor der Generator-Suite und die Markierung der relevanten Stellen im Design. Anschließend sollte die Zielplattform etwas spezifiziert werden. Darunter ist zu verstehen,dass der Nutzer des Systems angibt, welche Zielsprache, welche Datenbank etc. er benutzt. Danach kann das System mit Hilfe der ausgewählten Problemlösungsmodule den aufgabenbezogenen Code generieren. Dieser muss daraufhin eventuell an spezielle Anforderungen angepasst werden.

Ab diesem Punkt gibt es im Arbeitsfluss drei Möglichkeiten, wie es weitergeht. Entweder die Web-Applikation wird abgenommen oder es gibt nachträglich Änderungen des Designs oder der Zielplattform. In den beiden letzteren Fällen gelangt der Entwickler wieder an den Punkt „Design wird in die WFGenerator-Suite geladen und markiert“. Genau hier birgt die Generator-Suite bei jetziger Funktionalität einen Nachteil. Bei mehrmaliger Code-Generierung werden manuell nachprogrammierte Code-Stellen wieder überschrieben. Daher wäre eine wichtige zukünftige Verbesserung die Kommentierung von manuell nachbearbeitetem Code, sodass ein Lösungsmodul bei erneutem Generieren den manuell erstellten Code erkennt und diesen nicht überschreibt, sondern nur die Code-Stücke

darum. Man stelle sich etwa vor, der Frontend-Entwickler ließe sich durch ein Problemlösungsmodul einen Funktionsaufruf und einen entsprechenden Funktionsrumpf generieren. Wenn er nun den Funktionsrumpf mit Code füllen würde, könnte eine entsprechende Kommentierung verhindern,dass das Lösungsmodul beim nächsten Generierungsdurchgang die ausprogrammierte Funktion wieder durch einen leeren Rumpf ersetzt. Die Verbesserung würde Frontend-Entwickler davor schützen, manuell fertiggestellte Skripte zu überschreiben. Die Kommentierung würde blockweise nach dem Prinzip „Hier fängt manueller Code an“ bzw. „Hier hört manueller Code auf“ von statten gehen. Solche Block-Kommentare könnten bereits an den richtigen Stellen vom Lösungsmodul mitgeneriert werden. Eine entsprechende Verbesserung könnte die genannte Lücke des Systems schließen. Lediglich die Änderung der Zielsprache hätte zufolge,dass manuell programmierter Code in diese neue Sprach übersetzt werden müsste.

Trennung von Anwendungslogik und Design

Wie bereits in der Einleitung erwähnt, findet bei der Verwendung von Internetskripten, die HTML-Code erzeugen, keine eindeutige Trennung von Anwendungslogik und Design statt. Dieses System hilft zwar, die Entwicklung von beidem zu trennen, jedoch führt die Nutzung von HTML nach wie vor zu Unannehmlichkeiten wie sie beispielsweise im obigen Kapitel erörtert werden. Das Ineinandergreifen von Design und Funktionalität beeinträchtigt stets die Lesbarkeit und Wartbarkeit solcher Skripte. Daran kann dieses System aber an sich nichts ändern. Die einzige Möglichkeit, diesem Nachteil zu entkommen liegt darin, sich von der Seitengestaltung in HTML zu trennen und zum Beispiel Ansätze wie XHTML (siehe Einleitung) zu verfolgen. Die Generator-Suite könnte den Frontend-Entwickler auch auf diesem Weg begleiten. Da der Editor des Systems ein Texteditor ist, ist es natürlich möglich, statt HTML-Dateien andere Dateien zu laden, die eben in XHTML formuliert sind. Dem Editor kann es schlichtweg „egal“ sein, ob er HTML oder etwas anderes einliest. Zudem wäre es ohne weiteres möglich, Problemlösungsmodule auf XML-Basis zu entwickeln, die Skript-Code generieren, welcher wiederum wohlgeformten XML-Code erzeugen kann. Es wäre für das System also kein Problem, sich von HTML loszulösen. Lediglich der Designer eines Web-Auftrittes müsste sich daran gewöhnen, Seiten nicht mehr als ganzes gestalten zu können (top-down), sondern das Aussehen einzelner untergeordneter Strukturen definieren muss, um später das gewünschte Gesamtausehen der Seite zu erreichen (bottom-up).

Kapitel 7

Ausblick/Fazit

Wie im letzten Kapitel erwähnt, handelt es sich bei dem entstandenen System um eine Unterstützung der Web-Frontend-Entwicklung in mittleren oder kleinen Projekten. Für große Projekte ist der Einsatz bewährter Frameworks/Systeme ratsam, denn bei Projekten größerer Komplexität ist die Entwicklungszeit im Verhältnis zu den Kosten solcher System kleiner, als bei weniger komplexen Projekten. Zudem ist auch der Support bei bekannten großen Systemen vorhanden, welcher in Pannenfällen großer Projekte unabdinglich ist. Die Abgrenzung dieses Systems von solchen, die in Großprojekten eingesetzt werden, war ja auch eine Anforderung aus Kapitel 2.4. Für kleinere bzw. mittlere Projekte kann die Generator-Suite jedoch ein sehr interessantes Werkzeug sein. Das liegt zum einen daran,dass sie weniger der in Kapitel 2.3 erörterten Nachteile beinhaltet, diese sogar durch ihr Konzept beseitigt. Natürlich hängt die Mächtigkeit dieses System sehr stark von den zur Verfügung stehenden Problemlösungsmodulen ab. Wenn es beispielsweise auch in Zukunft nur eine oder zwei Problemlösungsmodule gäbe, dann hätte der Frontend-Entwickler nichts davon,dass dahinter ein sehr flexibles System steckt, da der effektive Nutzen einfach zu gering ist. Nur für eine Problemlösung wird er sich wahrscheinlich nicht einmal Eclipse installieren wollen (es sei denn, er benutzt Eclipse bereits). In diesem Fall könnte man auch fast wieder zu einer der Lösungen aus Kapitel 2.2 greifen.

Für das Unternehmen OMD Outdoor ist die Entwicklung auf jeden Fall eine Hilfe, da die hier benötigten wiederverwendbaren Problemlösungen identifiziert wurden und zum Teil fertiggestellt sind. Ob die Generator-Suite darüber hinaus Frontend-Entwicklern helfen kann, hängt von folgenden Punkten ab

- Könnten die bereits entwickelten Problemlösungen tatsächlich in „fremden“ Projekt so effektiv eingesetzt werden? Hier gibt es zu bedenken,dass jedes Unternehmen seine etwas eigene Art der Software-Entwicklung hat und so kann es auch sein,dass andere Entwickler beispielsweise lieber per Hand programmieren, weil sie keine Zeit haben, um sich mit der Generator-Suite vertraut zu machen. Es könnte auch sein,dass andere Entwickler bereits ihre

eigenen Werkzeuge besitzen oder eben auf eine der Lösungen aus Kapitel 2.2 setzen

- Würden sich Entwickler finden, die die Zeit und Energie haben, das System um neue Lösungsmodule zu erweitern? Hier spielt ja gerade Zeit und Energie (Lust) die größte Rolle. Natürlich kommt hinzu,dass die Generator-Suite als neues, unbekanntes System Schwierigkeiten haben dürfte, Entwickler für sich zu begeistern. Dies würde natürlich zur Folge haben,dass sich außerhalb von OMD Outdoor niemals jemand mit der Entwicklung von Problemlösungsmodulen für die Generator-Suite beschäftigen wird und ohne neue Problemlösungsmodule das System für externe Entwickler „uninteressant“ wird.

Abgesehen von der Mächtigkeit durch die Erweiterbarkeit des Systems und der Vielfalt an möglichen Problemlösungsmodulen, ist weiteres Potential des System durch Verbesserungen zu erkennen. Eine gute Verbesserung wäre zum Beispiel die Nutzung des Builder-Mechanismus von Eclipse. Dieser erlaubt es, eigene Builder in Eclipse zu integrieren. Ein neuer Builder für WebFrontends könnte die automatische Generierung beim Speichern eines markierten Designs übernehmen. Desweiteren könnte mit einer Aktion das ganze Projekt neu durchgeneriert werden. Das könnte dann von großem Nutzen sein, wenn innerhalb eines Projektes der Wechsel der Datenbank beschlossen wird. In diesem Fall bräuchte der Frontend-Entwickler lediglich die Einstellungen verändern und anschließend das ganze Projekt neu builden lassen. Diesen Mechanismus kennt man üblicherweise von der Programmierung in einer IDE, wo ebenfalls des öfteren das ganze Projekt oder Teile dessen auf einmal neu kompiliert werden.

Sollte die Zeit vorhanden sein, werden im Unternehmen OMD Outdoor weitere Problemlösungsmodule (darunter einige der erwähnten Ideen) entwickelt werden. Dies wird die Generator-Suite zu einem stetigen Begleiter bei Web-Frontend-Aufgaben machen. Eine weitere Überlegung ist nun, das System auf einer öffentlichen Open-Source Plattform wie SourceForge[1] vorzustellen, um so Entwickler dazu zu bringen, über die Entwicklung weiterer Problemlösungsmodule nachzudenken. Bei entsprechendem „Erfolg“ könnte das System theoretisch ein Eigenleben entwickeln und mit der Zeit, ähnlich die Eclipse-Plattform selbst, heranwachsen.

Bei OMD Outdoor ist es auch denkbar,dass man früher oder später auf XML-basierte Seitenentwicklung umsteigt, um die Entwicklung und Wartung des Web-Auftrittes sauberer zu gestalten. Trotz dieses Umstieges könnte die Generator-Suite eine gute Unterstützung bei der Entwicklung der Skripte sein, denn auch wenn sich dann statt HTML XML mit Skript-Code vermischen würde, wäre die Programmierung von Skripten dennoch prinzipiell gleich.

[1]SourceForge ist eine zentrale Internet-Plattform für OpenSource-Projekte. Dort können Projekte erstellt, Source-Dateien gespeichert werden etc. (s. [Net03])

Literaturverzeichnis

[Aca03] IT Academy. Glossar von it-academy. http://www.it-academy.cc/-content/glossary_browse.php, Stand 20.12.2003.

[BAS+03] Stig Sther Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, Andrei Zmievski, and Jouni Ahto. Php handbuch. http://www.php.net/manual/de/, Stand 20.12.2003.

[Con03] Eclipse Consortium. The eclipse project. http://www.eclipse.org, Stand 20.12.2003.

[Cor03a] Microsoft Corporation. Active server pages. http://msdn.microsoft.-com/library/default.asp?url=/nhp/default.asp?contentid=28000522, Stand 12.06.2003.

[Cor03b] Microsoft Corporation. Access home page. http://www.microsoft.-com/office/access/default.asp, Stand 13.07.2003.

[GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. *Lecture Notes in Computer Science*, 707:406–431, 1993.

[Gmb03] Wissen.de GmbH. Wissen.de lexikon. http://www.wissen.de, Stand 05.10.2003.

[Mic04] Sun Microsystems. http://java.sun.com/docs/books/tutorial/get-Started/intro/% -definition.html, Stand 26.02.2004.

[Mn03a] Stefan Mnz. Selfhtml: Einführung. http://selfhtml.teamone.de/-intro/index.htm, Stand 20.12.2003.

[Mn03b] Stefan Mnz. Selfhtml: Javascript/dom. http://selfhtml.teamone.de/-javascript/index.htm, Stand 20.12.2003.

[Nat03] Natsoft(M). phplens php application server. http://www.phplens.-com, Stand 12.06.2003.

[Net03] Open Source Development Network. Sourceforge.net. http://-www.sourceforge.net, Stand 20.12.2003.

[Res03] IT Research. Das informationstechnik-glossar von it research. http://www.it-research.net/de/info/glossary/, Stand 05.08.2003.

[SM03] inc. Sun Microsystems. Java server pages (tm) technology. http://java.sun.com/products/jsp/, Stand 05.06.2003.

[W3C03a] W3C. http://www.w3.org/TR/xhtml1/, Stand 15.08.2003.

[W3C03b] W3C. W3c xml schema. http://www.w3.org/XML/Schema, Stand 15.08.2003.

[Yes03] YesSoftware. Codecharge™. http://www.codecharge.com, Stand 15.06.2003.

[Zeh98] Carl August Zehnder. *Informationssysteme und Datenbanken*, chapter Datenbanken, page 47. vdf Hochschulverlag, 1998.