Master's Thesis

A Domain Specific Language for Building AJAX-enabled Web Application

carried out at the

Information Systems Institute
Distributed Systems Group
Technical University of Vienna

under the guidance of
Univ.Prof. Dr. Schahram Dustdar
and
Univ.Ass. Dipl.-Ing. Johann Oberleitner
as the contributing advisor responsible

by

Hannes Etl
Josefistraße 20 - 7132 Frauenkirchen
Matr.Nr. 0027601

Vienna, 27. November 2007 _____

Acknowledgements

My special thanks go to my advisor Johann Oberleitner for his guidance, advice and support during the whole work. His expertise and research experience were crucial to improve the outcome of this thesis.

I also want to thank my parents for their support during the last years. With their patience and support they made it possible for me to finish my studies and write this master thesis.

Abstract

Nowadays developers of web applications aim at developing dynamic web applications with a high degree of user acceptance. AJAX (Asynchronous JavaScript and XML) is one promising technique for building modern desktop-application-like user-interfaces with a high level of interactivity. This thesis introduces a model-driven approach for building AJAX-enabled user interfaces specified by models and their relationships. Our prototype implementation is based on the Microsoft Domain Specific Language Tools and the Microsoft AJAX framework ASP.NET AJAX. Its key aspect is to accelerate and simplify the development process of professional AJAX-enabled web applications.

Zusammenfassung

Die Entwicklung von Internetapplikationen erfreut sich seit einigen Jahren immer größerer Beliebtheit. Zum einen ermöglichen moderne, dynamische Technologien wie beispielsweise Java Server Pages (JSP) oder Active Server Pages (ASP.NET) die Realisierung beliebiger Anforderungen im Umfeld von Internetanwendungen, zum anderen verfügen die genannten Technologien über spezielle Eigenschaften, welche eine rasche Entwicklung von professionellen Lösungen ermöglichen. Aus Sicht der Benutzerfreundlichkeit sind klassische Internetanwendungen dennoch nicht mit Desktop Anwendungen gleich zu setzen. AJAX (Asynchronous JavaScript and XML) ist eine Technologie, welche eine Verbesserung hinsichtlich Benutzerfreundlichkeit in Internetapplikationen verspricht. In dieser Arbeit wird eine Applikation beschrieben, welche die Realisierung von AJAX basierten Internetanwendungen auf Basis des modelgetriebenen Ansatzes sowohl ermöglicht als auch beschleunigt. Eine graphische Entwicklungsumgebung bietet Entwicklern die Definition des Models einer gewünschten Internetapplikation sowie die Generierung des Quellcodes. Die Implementierung dieser Anwendung basiert auf .NET Technologien. Insbesondere kommen das ASP.NET AJAX Framework von Microsoft zur Integration von AJAX basierten Steuerelementen in der Benutzerschnittstelle sowie die Microsoft Visual Studio DSL Tools zur Realisierung einer domänenspezifischen Sprache zum Einsatz.

Contents

List of Figures

Chapter 1

Introduction

Over the past decade web applications have become increasingly popular. Various available dynamic web technologies like Active Server Pages (ASP.NET) or Java Server Pages (JSP) provide frameworks and rich capabilities to build complex, high functional web applications. At the same time the time-to-market scales down and the complexity of web applications grows continually. For this reason developers aim at making web applications easier, enhancing usability and integrating a high degree of functionality. By using new innovative techniques web applications gradually achieve desktop-like behavior and appearance. Although the movement from desktop applications to web applications has gained much attention and acceptance, interactivity and performance have been lost within this movement [21]. These circumstances bring up new challenges for software developers. Various vendors offer numerous frameworks and tools for various platforms. The ulterior motive of these products is to enhance and standardize the development of web applications. Addressing the user-interfaces of web applications the acronym AJAX (Asynchronous JavaScript and XML) and some other proprietary technologies like Adobe Flash have become increasingly popular. AJAX provides rich capabilities to design professional, desktop-like user-interfaces. As AJAX requires a competent knowledge of various technologies such as JavaScript, DOM, etc. the integration of those technologies still remains a challenge for developers. This fact raises well known shortcomings within software development like prolongation of the development of applications and additional acquirement of knowledge. For this reason various vendors and Open Source communities offer frameworks and toolkits which simplify the development of AJAX-enabled user-interfaces. Nevertheless enhancing web applications by AJAX requires knowledge about AJAX and hands-on experience on at least one AJAX-framework including its various libraries and functionalities. This requires a certain effort and slows down the development process. For this reason this thesis provides an approach which allows developers with less or no experience on AJAX to build professional AJAX-enabled web applications.

Nowadays software development at all is still an error-proneness, time-consuming and cost-intensive process. Nevertheless building applications using 3GLs (3rd Generation Languages) like C# or Java programmatically is a typical approach for solving software development tasks. As the lifecycle of modern software applications abbreviates new techniques need be used continually to be competitive. For this reason software developers are forced to optimize their software development process using innovative technologies, methods and techniques. Model-Driven Development (MDD) is one promising paradigm within software development which has recognized existing shortcomings and accepted this challenge. For this reason this thesis figures out key aspects, capabilities, advantages and disadvantages of Model-Driven Development. Based on the concepts of Model-driven Development this thesis provides a modern model-driven approach for developing AJAX-enabled web applications to improve and ease the process of web application development. A whole section is dedicated to the introduction of MAXAL (Model-Driven AJAX Application Language), which is a code generation tool for ASP.NET based web applications. This code generator is based on ASP.NET, the Microsoft DSL Tools and the ASP.NET AJAX framework which enables developers to generate AJAX-enabled user-interfaces. The fundamental idea of MAXAL is that developers specify models of their web application within the MAXAL-Designer which provides a visual user-interface to create web applications by specifying models and their relationships. For this reason developers build their applications without writing code using a 3GL. Similar to approved user-interface-based tools for designing relational databases, developing AJAX-enabled user-interfaces is realized *Out-Of-The-Box* by *Drag&Drop*. The prototype implementation of MAXAL, which is discussed in detail in this thesis, generates fully AJAX-enabled user-interfaces. Numerous AJAX-based extensions for standard user-interface controls like the ASP.NET Calendar control, the ASP.NET Button control or the ASP.NET Textbox control are integrated. Furthermore horizontal and vertical positioning features and the ASP.NET AJAX AutoComplete Extender are supported and can be used to customize websites. A sample application shows its rich capabilities. In addition this development environment is designed to be programmatically extensible. Developers might realize their own customized user-interface controls or modify existing user-interface components for their purposes without accessing and modifying existing source code of the prototype implementation of MAXAL. This specific feature enables developers to add additional custom controls and functionalities to the prototype implementation of the MAXAL-Designer with reasonable effort. Furthermore this code generation tool provides mechanisms to integrate

data-intensive user-interface controls. For this reason generated artifacts might be included into existing web applications.

1.1 Organization of this thesis

Section 2 provides an introduction to AJAX. On the one hand this section describes its evolutionary history, capabilities, advantages, disadvantages and innovative background ideas. Classical web applications are faced with modern AJAX-enabled user-interfaces. On the other hand this section focuses on implementation details. Numerous examples illustrate core implementation techniques. Furthermore this section introduces some popular, current available AJAX frameworks which simplifies developing AJAX-enabled user-interfaces. Another subsection of Section 2 focuses on ASP.NET. Section 2.4.2 deals with the MS DSL Tools - a toolkit for developing domain-specific languages on the .NET platform. Section 2.4 introduces Model-Driven Development - a modern approach for developing software.

Section 3 introduces MAXAL, its core features, design principles and capabilities. Numerous examples illustrate implementation details. As MAXAL is based on the MS DSL Tools some of the core features of this technology are discussed in detail. Section 3.1 presents a meta-model for developing web applications. This model is used as basis for the development of the source code generation tool MAXAL.

Section 4 provides an evaluation of MAXAL. An implementation of a sample application shows significant measured values to evaluate utilization and applicability of MAXAL.

Finally Section 5 deals with existing approaches and compares these with MAXAL. Advantages and disadvantages of MAXAL and other tools are highlighted.

Section 6 concludes the key aspects of this thesis and provides an outlook for further improvements of the current version of MAXAL.

Chapter 2

Review of the State of the Art

Developing dynamic web applications has become increasingly popular. Although modern web development techniques provide a wide range of capabilities, web application development underlies certain limitations. Web applications are based on the request-wait-response-wait pattern [30], which exhibits problems like limited interactivity. After submitting a request the user has to wait until the response arrives. [30]

Keith Smith [30] illustrates a key problem in classical web applications: A user visits an online travel reservation system and edits a place of departure, the travel destination, preferred dates and times and then submits a request to locate all available flights. After receiving the server's response the user notices that he typed the wrong airport code for the travel destination. Consequently all received flight information is useless. The user enters the correct code and submits the request again. Furthermore sorting the received flights would again cause a request. In the above mentioned scenario we can see that by simply editing the wrong airport code – time, network resources, server resources and bandwidth are wasted, which is not efficient. According to this inefficient process there is a claim for developing more desktop-application-like web applications. Now developers profit of AJAX.

2.1 AJAX

The acronym AJAX (Asynchronous JavaScript and XML) pictures a programming technique for enhancing user's experience on web applications. This technique is a set of technologies mostly developed in the decade of the 1990s. Its component technologies have been improved and are now ready for using within enterprise applications. Google[1] and other companies have already developed early AJAX-enabled web applications like Gmail, Google Maps or Google Groups[2] [25, 30]. The key advantage of AJAX-enabled applications is performing partial page updates

[1] http://www.google.com (28th September 2007)
[2] AJAX-enabled features like a webmail, maps and a discussion platform offered by Google

instead of refreshing the whole page after each request. Classical web applications refresh the whole page even for small changes. Within AJAX-enabled applications changes are made to specific user interface components on a single webpage [21]. The fundamental advantages of this approach are a reduced latency and an improved user-interactivity. Using AJAX web-applications become more interactive and customizable and mature to rich and desktop-like applications [25, 30]. By developing AJAX-enabled web applications the above mentioned scenario can be improved and at the same time usability can be enhanced. The following paragraph shows an AJAX-enabled solution.

As shown in Figure 1 during editing the airport code into the text field, the browser could initiate a request to the server in the background to receive all airports matching the letters typed in the text field and displays a list with all matching airport codes. Subsequently available letters are completed to full words automatically. Consequently the user would immediately notice a wrong input letter before submitting the request. Using this approach in web applications data exchange between client and server is less verbose and no round-trip is necessary. Additionally the browser has a full description of all received flight information, which allows sorting without doing a request to the server. [30]

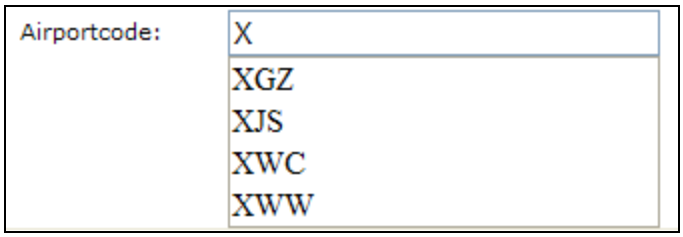


Figure 1: Completion of Matching Airport Codes

2.1.1 Components of AJAX

AJAX is based on a set of approved technologies. Its components are CSS (Cascading Style Sheets), DOM (Document Object Model), XML (Extensible Markup Language), JS (JavaScript) and DHTML (Dynamic Hypertext Markup Language). The following sections discuss each of these technologies in detail.

Cascading Style Sheets (CSS)

CSS addresses the layout of web pages [25]. The basic idea of cascading style sheets is the separation of layout and content within a web page. Within traditional web pages layout and content was mixed with each other. The concept of cascading style sheets enables developers to define styles for groups of user-interface controls. Consequently it is not necessary to define the layout using HTML capabilities [33]. Cascading style sheets are a W3C standard since 1996 [9].

Within AJAX the technique of cascading style sheets provides powerful capabilities. CSS combined with JavaScript and DHTML allows changing layout and position properties of special elements inside the front-end of a web page. Using this combination of techniques visual effects can be realized. For example a menu pops up while hovering with the mouse or a special designed sandglass appears while a user is waiting for the answer of a submitted request. Furthermore the use of cascading style sheets is not limited to HTML. Cascading style sheets can also be applied to XML documents. As XML is a typical exchange format within AJAX, developers have the capability to define the layout for XML documents. Consequently received data can be displayed immediately using the desired layout properties without any additional transformation process. [33]

Document Object Model (DOM)

The Document Object Model, a W3C standard since 1998 [10], defines a platform independent programming interface for developers to create and modify HTML (Hypertext Markup Language) or XML (Extended Markup Language) documents. Using DOM, HTML or XML, documents are mapped as tree structures containing nodes and relationships between nodes. Consequently developers have the capability to dynamically change the appearance of a webpage by using DHTML (Dynamic Hypertext Markup Language), even if the page has already been loaded. All elements within a user-interface are identified by unique identifiers. This concept allows accessing and modifying elements at any time during the execution of a website. [25, 33]

Extensible Markup Language (XML)

XML is a platform independent markup language similar to HTML [34]. XML documents are readable for both machine and man. An XML document consists of various blocks like elements, attributes, processing instructions or comments. In contrast to HTML XML distinguishes between the representation of data and the actual data. While HTML mixes both concerns, XML documents are only responsible for the structuring of the data. [33]

Within AJAX data exchange between client and server is basically processed by using XML communication between client and server asynchronously via HTTP (Hypertext Transfer Protocol) requests. For instance, a request is firstly transformed to XML and subsequently the transformed message is sent to the server via HTTP. Responses are received in the background without interrupting users' current actions on the webpage. [25]

JavaScript (JS)

JavaScript, released in 1995 by Netscape and Sun, is a scripting language which is often used for client-side web development. JavaScript interacts with HTML code and makes web pages more interactive [25]. JavaScript is a key technology when using AJAX. This scripting language is used to call modified data within a webpage without refreshing the whole page. After loading required data from a server DHTML is used to update the view of the webpage. All available AJAX-based frameworks are based on JavaScript to realize AJAX-specific functionalities [33]. Java Script is used to glue all the used technologies – HTML, CSS, XML, DOM and DHTML together. [25]

Dynamic Hypertext Markup Language (DHTML)

Dynamic HTML describes an extension of HTML to develop dynamic web pages that are more animated than classical web pages using pure HTML. Its core technologies are HTML and JavaScript. DHTML enables the development of web pages containing visual effects. For instance while a user passes the cursor over a menu a submenu appears. [25] Shelly and Young [29] provide an example including a form validation. The task is to display an appropriate error

message immediately near the element where the error occurred. Figure 2 shows its source code and how DHTML modifies the web page by accessing and modifying the DOM of the web page.

```
<script type="text/javascript">
function validate()
{
      //do validation
      //return error
      phoneLabel.innerText = "Telephone number must be 9 digits";
}
</script>
```

Figure 2: Form Validation using JavaScript

Before executing the script the HTML code looks like in Figure 3.

```
<label id="phoneLabel" for="phone">Telephone Number</label>
<input id="phone" type="text" />
```

Figure 3: HTML Code before Executing the Script

After executing the script the HTML code looks like in Figure 4.

```
<label id="Label1" for="phone">Telephone number must be 9 digits</label>
<input id="Text1" type="text" value="999" />
```

Figure 4: HTML Code after Executing the Script

2.1.2 Advantages of AJAX

Paulson [25] shows that AJAX-enabled web applications generally show better performance than web applications that are not using AJAX. On the one hand there are no interruptions between submitting and receiving requests because of omitting the page-refresh, on the other hand network traffic between client and server is minimized because only required data needs to be transmitted. Furthermore web application developers generally have hands-on experience with its component technologies.

Unlike Adobe Flash or other proprietary web application technologies, which enhance user experience of web applications, AJAX does not require any additional plug-ins because

JavaScript unlike Adobe Flash is supported *out-of-the-box* by various available browsers. Furthermore recent browser versions of Internet Explorer, Mozilla's Firefox, Netscape and Apple Computer's Safari work with HTTP requests containing XML-based data. However although Adobe Flash requires an additional plug-in, it is no challenge even for laymen to download and install the required plug-in. By the way recent versions of various browsers show an information message if the Macromedia plug-in is required. Consequently a user only has to accept an offered download by clicking on a button. As AJAX itself is not a new technology because of its approved components, integration into classical web pages is no challenge. On the other hand other technologies like Adobe Flash require additional knowledge about integration of written source code.

Smith writes in her thesis [30] that AJAX-enabled web applications have characteristics of distributed systems because the execution of tasks of a program is divided up to both server and client. Within classical web applications web browsers are responsible for displaying results retrieved by an application server, while using AJAX-enabled web applications a server makes use of the client's processing power. However using client side capabilities is not an AJAX-specific feature. Since JavaScript has been available since client scripting and exploiting Browsers' capabilities are matters of public concern. For instance client-side input validation is basically a typical JavaScript task and no AJAX-specific feature as mentioned in [31]. For instance, validating if a user entered text into a required text field, can be implemented without a request.

2.1.3 Disadvantages of AJAX

According to Paulson [25] AJAX is an immature technology which still needs toolkits and frameworks. Smith claims in her thesis [30], published in May 2006, a lack of tool support, especially for designing, developing and debugging AJAX-enabled web applications. In the meantime numerous frameworks like ASP.NET AJAX[3], Backbase[4] or Echo2[5] for various platforms have been implemented. Furthermore some AJAX toolkits are available. For example the Microsoft Corporation published the ASP.NET AJAX Control Toolkit, which is a set of

[3] http://asp.net/ajax (28th September 2007)
[4] http://www.backbase.com (28th September 2007)
[5] http://www.nextapp.com/platform/echo2/echo (28th September 2007)

AJAX-enabled sample controls. Another Toolkit is provided by Google. The Google Web Toolkit (GWT) makes writing AJAX-based web applications easy for developers.

Using JavaScript is advantageous because it is a cross-platform scripting language, but differences in JavaScript implementations across different browsers cause problems. For example some browsers do not support some AJAX-specific features. [25]

Paulson [25] points out that users must get accustomed to AJAX-enabled applications that don't perform like classical web applications. As AJAX is a promising approach to transform web applications to desktop-like applications and users are familiar with classical desktop applications, users expect a desktop-like behavior within web applications and for this reason users will not have to get accustomed with next-generation user-interfaces.

Referring to Paulson [25] AJAX has also raised some security concerns. Although its component technologies have been approved over the last years the combination of techniques are now being used in unproven ways. Using approved security techniques like the Secure Sockets Layer protocol (SSL) AJAX-enabled web applications become secure. [27] However allocating tasks of the web application to the client-side leads to the situation that security is not primarily a server issue anymore as in classical web applications but also a client issue. [31]

2.1.4 Implementing an AJAX-enabled Web Application

Stamey and Richardson present an example [32] showing an AJAX-enabled website based on the LAMP (Linux, Apache, MySQL, and PHP) platform. The implementation focuses on validation with JavaScript as well as processing of input items through the XMLHttpRequest object. As shown in Figure 5 its homepage consists of six hyperlinks. Whenever a user clicks on a hyperlink some text will be displayed on the website without refreshing the whole page. Tasks related to the form submission also include a validation of a provided email address.

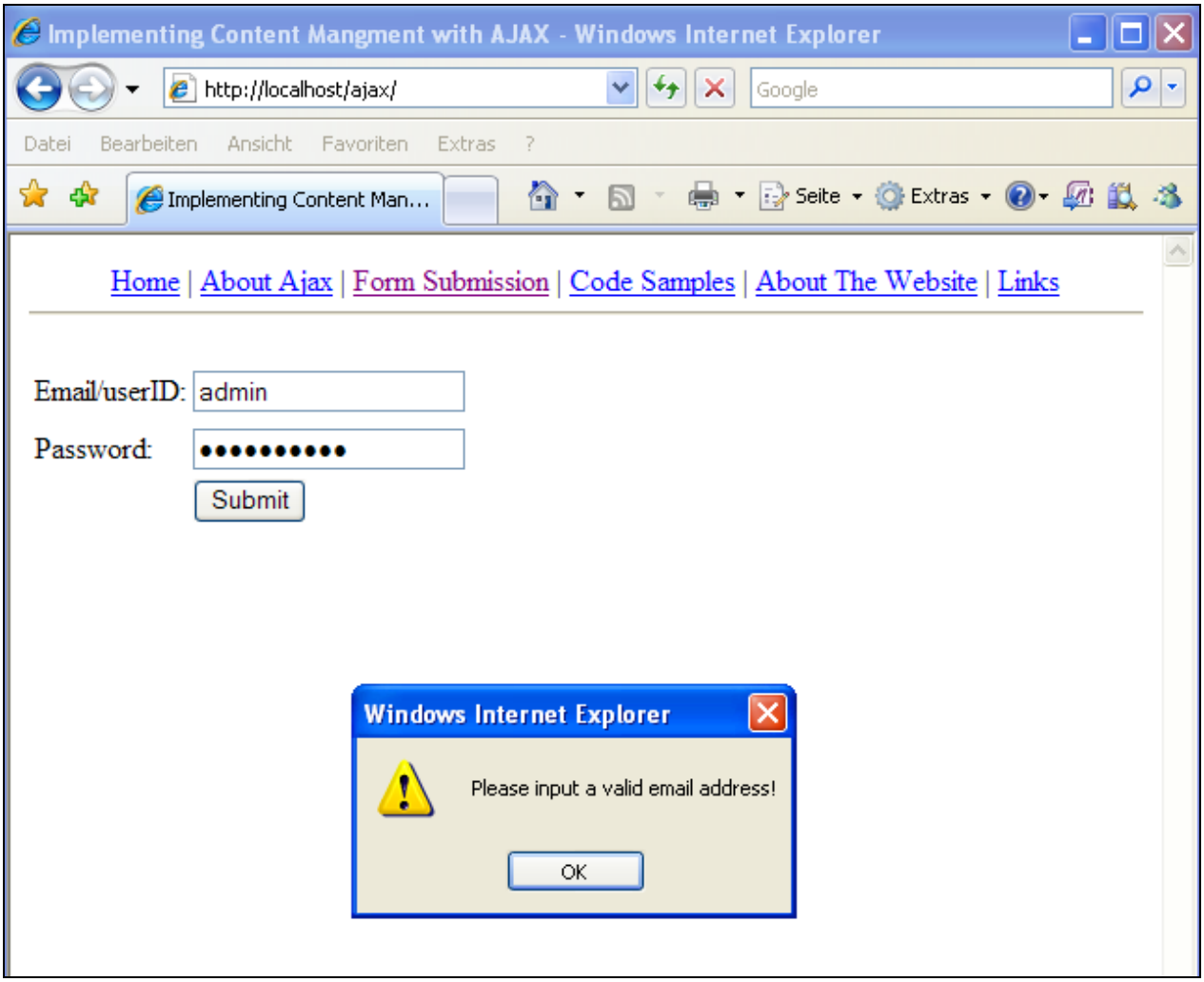


Figure 5: AJAX-enabled Web Application

2.2 ASP.NET

ASP.NET is a key component of the Microsoft .NET framework which enables developers to build high-performance web applications and web services. ASP.NET 1.0 was released in early 2002. When ASP.NET 1.1 was released in late 2003 the focus was not to bring up new features but to tune-up performance, enhance security issues and remove minor bugs of existing components within ASP.NET 1.0. The successor of ASP.NET 1.1 was ASP.NET 2.0 which appeared in 2005. This version provides a comprehensive collection of new higher-level features to the existing technology for web developers. The following sections describe facts, characteristics and core capabilities of ASP.NET 2.0. [19]

Web developers are continuously confronted with a wide variety of browsers. As all of the existing browsers differ in their support of HTML, the challenge is to develop web applications that are supported inside all available environments. Using additionally JavaScript to create more dynamic pages intensifies this problem. ASP.NET addresses this problem by offering a rich collection of web server controls. These controls use clients' capabilities to render their HTML. For instance ASP.NET offers developers validation controls. Therefore ASP.NET uses JavaScript and DHTML if the client supports it. [19]

One of the last steps when realizing a web application project is to deploy the completed web application. This deployment requires the transfer of all web page files, databases and components. Furthermore specific configuration settings are needed. ASP.NET addresses and simplifies this process. Most ASP.NET settings are handled within the XML-based file *Web.Config*. This file is part of each web application. It contains a hierarchical grouping of application settings and can be modified at any time without recompiling the application. [19]

ASP.NET is XML-based

Developers design their web pages, controls and layouts using XML-based syntactical rules like for instance elements and attributes. A typical definition of an ASP.NET control is shown in Figure 7.

ASP.NET is Object-Oriented and Component-based

Developers using ASP.NET have full access to all objects of the .NET Framework. Furthermore typical components and features of the OOP (Object-Oriented Programming) paradigm such as reusable classes, interfaces or polymorphism can be used. [19] Modern imperative programming languages use object-oriented techniques to build reusable components. Frankel [11] illustrates the key advantage of component-based development.

> *"Componentization moves the production process away from reinventing the same solution in different applications, thus improving productivity and decreasing the cost of production."*

Furthermore componentization improves quality because it isolates functionality which allows tool builders to debug and upgrade a component in one place [11].

As ASP.NET contains a comprehensive set of reusable server-based user-interface components, ASP.NET is component-based. Typical front-end controls within ASP.NET are for instance buttons or labels. Similar to JSF (Java Server Faces) on the Java platform this feature enables developers to define the appearance and behavior of user-interface controls without handling request objects and response objects for sending and receiving HTTP requests and responses. Furthermore modifying low-level HTML code is not required.

2.2.1 The Code-Behind Coding Model

The typical encouraged coding model is the *Code-Behind* model. This model separates an ASP.NET web page into two files. While the .aspx markup file contains HTML code and control tags, a .cs file is used to insert source code for a page. This concept is shown in Figure 6. Its major benefit is that the user interface and programming logic are strictly separated which allows an enhanced organization of web applications. [19]

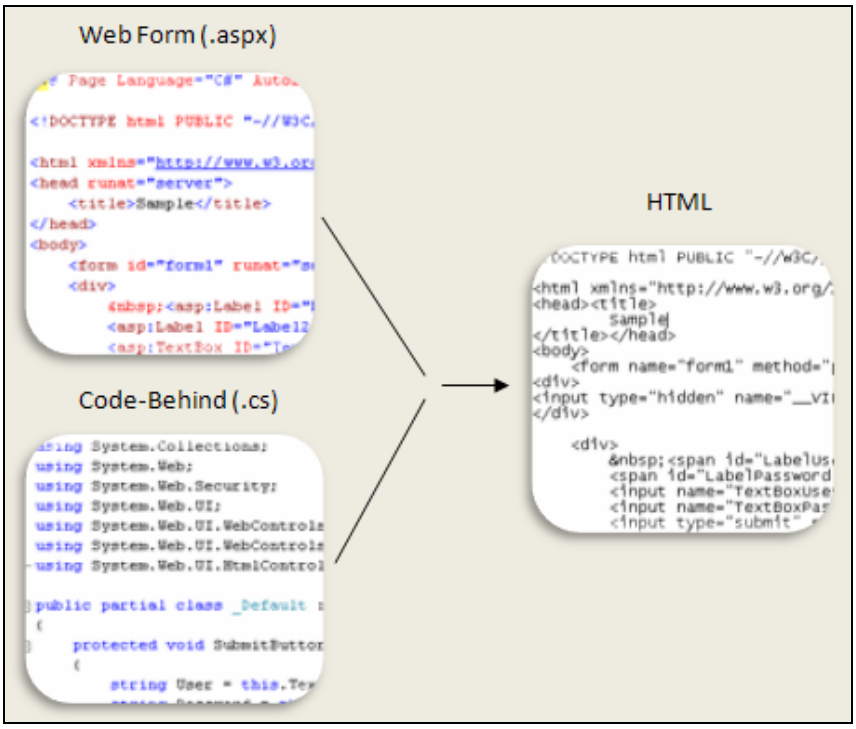


Figure 6: Transform a Web Form and a Code-Behind File to HTML

The example, shown in Figure 7 and Figure 8 illustrate the *Code-Behind* model. Using the *Inline* model the complete source code, including both static and dynamic parts, would be mixed within one single file.

```
01 <%@ Page Language="C#" AutoEventWireup="true"
02  CodeFile="TestFormCodeBehind.aspx.cs" Inherits="TestFormCodeBehind" %>
03
04 <html xmlns="http://www.w3.org/1999/xhtml">
05 <head runat="server">
06   <title> Test Page </title>
07 </head>
08 <body>
09   <form id="form1" runat="server">
10   <div>
11    <asp:Label ID="Label1" runat="server" Text="Click Me!"></asp:Label>
12    <br />
13    <br />
14    <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
15     Text="Button" /></div>
16   </form>
17 </body>
18 </html>
```

Figure 7: ASP.NET 2.0 Code-Behind Model - Web Form

```
01 using System;
02 using System.Data;
03 using System.Configuration;
04 using System.Web;
05 using System.Web.Security;
06 using System.Web.UI;
07 using System.Web.UI.WebControls;
08 using System.Web.UI.WebControls.WebParts;
09 using System.Web.UI.HtmlControls;
10
11 public partial class TestFromCodeBehind : System.Web.UI.Page
12 {
13   protected void Button1_Click(object sender, EventArgs e)
14   {
15     Label1.Text = "Current time: " + DateTime.Now.ToLongTimeString();
16   }
17 }
```

Figure 8: ASP.NET 2.0 Code-Behind Model - Code-Behind File

The .aspx file contains the directive *Page* where developers specify the programming language and the associated code file. The code file is a so called *partial* class. The purpose of partial classes is to allow a class's definition to span across multiple files. [19] This feature is firstly introduced in .NET 2.0. Furthermore Figure 7 and Figure 8 illustrate the fundamental

programming technique to add an event handler to an ASP.NET Button control. [19] To add the *Click* event to an ASP.NET Button the attribute *OnClick* with the name of the event handler has to be added. Event handlers are accessible methods inside the Code-Behind file.

Alternatively event handlers can be assigned to controls through code using delegates, a .NET equivalent of function pointers. The following code fragment shows an example. [19]

```
cmd.Click += new EventHandler(txtName_Click)
```

Figure 9: Event Handlers & Delegates

2.2.2 AJAX-enabled web page using ASP.NET

Steyer and Fuchs show an AJAX-enabled web page based on ASP.NET and C#. [33] As shown in Figure 10 the web application displays a dropdown menu. Whenever a user selects an item from the list, a text related to the selection will be displayed without a full page refresh.

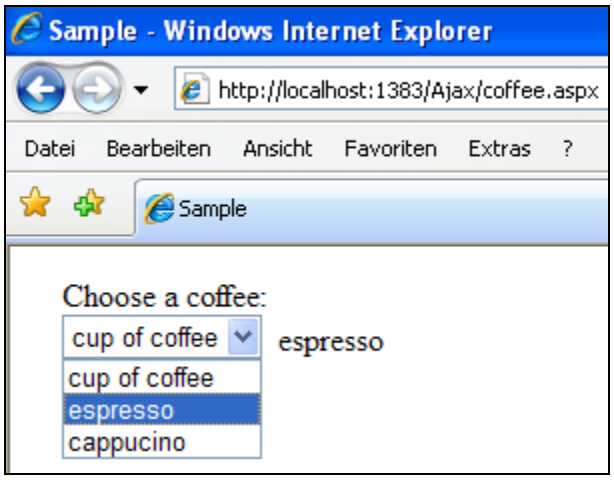


Figure 10: AJAX-enabled Web Page using ASP.NET

This example is based on the technologies C# and ASP.NET that are used for MAXAL described in Section 3. The source code of the ASP.NET based example is shown in Figure 11.

```
01 <%@ Page Language="C#" AutoEventWireup="true"
02 CodeFile="coffee.aspx.cs" Inherits="_coffee" %>
03 <html>
04  <head runat="server">
05   <script language="JavaScript" src="coffee.js" />
06  </head>
07  <body>
08   <form runat="server" name="form">
09    Choose a coffee:
10    <br />
11    <select name="coffee" size="1" onclick="sndReq()">
12    <option> cup of coffee </option>
13     <option> espresso </option>
14     <option> cappuccino </option>
15    </select>
16   </form>
17   <br />
18   <span id="hs"></span>
19  </body>
20 </html>
```

Figure 11: AJAX-enabled Web Page – Web Form

Line 4 provides a reference to an external JavaScript file named coffee.js. Inside this file all client-side AJAX-related tasks are implemented. So this file is the heart of the client-side AJAX engine. From line 8 to line 16 a web form is implemented which is referable inside JavaScript by using its name form. From line 11 to line 15 a drop-down menu is used to select various items. Additionally the event handler *onclick* implemented. Each time the selection changes a JavaScript function will be invoked. Line 18 provides a *span*-container which is used to display received data after a request.

Inside the external JavaScript file *coffee.js* a so called request-object is used to submit and receive data. Before data will be displayed to the user inside the browser, received data will be processed inside this request-object. Therefore the request-object provides methods and properties that can be invoked via JavaScript.

```
01 var reqObject;
02 if (navigator.appName.search("Microsoft") > -1) {
03  reqObject = new ActiveXObject("Microsoft.XMLHTTP"); // IE
04 }
05 else {
06  reqObject = new XMLHttpRequest(); // Mozilla, Safari, etc.
07 }
```

Figure 12: Instantiation of the Request Object

Dependent of the used browser either an ActiveXObject or an XMLHttpRequest object is instantiated. [33]

```
08 function sndReq() {
09  for(int i = 1; i <= 3; i++) {
10   if(this.document.form.coffee.options[i-1].selected) {
11    resObject.open("get","coffee.aspx?what="+i,true);
12    resObject.onreadystatechange = handleResponse;
13    resObject.send(null);
14    break;
15    }
16  }
17 }
```

Figure 13: Submission of a Background Request

As mentioned above the function *sndReq()* will be invoked each time the selection within the drop-down menu changes. Inside a for-loop the selected item of the drop-down menu is determined. If the selected item was found a HTTP connection to the web server will be initialized by using *resObject.open("get","coffee.aspx?what="+i,true)*. This method including its parameters initializes an asynchronous request of the specified web page *coffee.aspx*. Once a response arrives, the client-side AJAX engine invokes the function *handleResponse*(). [6]

```
18 function handleResponse() {
19 const complete = 4 // request completed
20  if(resObject.readyState == complete) {
21   document.getElementById("hs").innerHTML =  // "hs"… name of <span>-tag
22   resObject.responseText;
23  }
```

Figure 14: Update Content of an AJAX-enabled Web Page

In line 20 the state of data transmission is tested. If the request is completed the next step is to display the received data on the webpage. This is done by modifying the *span* container named *hs*. The attribute *innerHTML* allows accessing the content of the *span* container. The property *responseText* of the request-object contains the server response. Retrieved data will be allocated to the *span* container. [33]

2.2.3 Data Binding

Web applications typically provide data-intensive user-interface controls. *Data Binding* allows updating the view's data of a web application. Consider a simple example, such as the list of music titles. While the list could be hard-coded into the text, a better solution is to store the data external to the document within a database table or any other storage. If the music titles change, only the data in the database needs to be modified. Once the modifications are made, the page that displays that data will reflect the changes. For this reason ASP.NET 2.0 offers developers numerous facilities to bind data to user interface controls. The following list shows typical data binding components that simplify the development of ASP.NET web applications. [19]

- **SqlDataSource**
  This feature allows accessing a relational database, e.g. a MS SQL or Oracle database.
- **ObjectDataSource**
  This component creates a connection to a business object or dataset.
- **XmlDataSource**
  This control allows accessing a XML file.
- **AccessDataSource**
  This data binding mechanism allows accessing a MS Access database.

2.3 Frameworks for Developing AJAX-enabled Web Applications

Developers need advanced hands-on experience especially with JavaScript but also with other web technologies to develop professional AJAX-enabled web applications. Consequently developers have struggled with complexities of those techniques. Additionally developing AJAX-enabled web applications requires much effort on testing. [21] For this reason various AJAX frameworks have been developed to ease and accelerate the development process of AJAX-enabled web applications. Those frameworks offer numerous libraries including various functionalities. On the one hand working with those frameworks requires additional hands-on experience. On the other hand developing efficient, stable web applications is supported. [33]

Nowadays various AJAX frameworks are available. One of those is ASP.NET AJAX from Microsoft which is introduced in Section 2.3.4 in detail. Other important frameworks are

Echo2, Google Web Framework (GWT[6]) or Backbase which are described in the following sections.

2.3.1 Echo2

Echo 2 is an open-source AJAX framework for developing web applications using Java. It enables developers to implement web applications based on an object-oriented, component-based and event-driven approach. Tasks like rendering a component or communication with the client are assembled in a specific package called *Web Rendering Engine*. This engine consists of two components – a server-side module written in Java and a client-side module written in JavaScript. A key concept of Echo2 is the so called *Update Manager*. This feature is responsible for tracking updates within the user-interface and for processing input received from the rendering agent. The main task of the *Client Engine*, which is integrated within the client, is synchronizing the client/server state when user operations are performed inside the user-interface. Furthermore Echo2 uses the XML format for performing data exchange between server and client. Echo2 distinguishes between so called *ClientMessages* and *ServerMessages*. [21]

2.3.2 Google Web Framework (GWT)

GWT is another Java-based AJAX framework. GWT comes with a library of widgets that can be used to develop user-interfaces. Using GWT developing user-interface controls for web applications is similar to developing user-interface components using the Java AWT or Java Swing libraries. Its key concept is the process for rendering those user-interface components. GWT compiles Java code to JavaScript code. For this reason Java developers can use a subset of the Java API to add needed functionality to their components on client-side. GWT uses a small client-side engine. For this reason various functionalities are supported on client-side, which minimizes round-trips to the server. Submitting a request to the server is only required when raw data for user-interface components are needed. [21]

[6] http://code.google.com/webtoolkit/ (13th October 2007)

2.3.3 Backbase

Backbase is one of the first commercial AJAX frameworks based on Java. It is still in continuous development. The key element of Backbase is the *Presentation Client*, which is a client-side AJAX engine written in JavaScript. This module supports a declarative programming language called BXML. It consists of a library with various user-interface controls, a mechanism for event-handling and a feature to connect to a web server asynchronously. Furthermore Backbase is based on the Backbase Java Server (BJS). This server contains its own set of user-interface components and extends the JSF (Java Server Faces) framework. Developers develop their web applications by using those components declaratively. [21]

2.3.4 ASP.NET AJAX

ASP.NET AJAX, formerly ATLAS, is an AJAX framework developed by Microsoft. It is based on the ASP.NET library which offers methods that can be used on client-side using JavaScript. The *Client Script Library* extends classical JavaScript. Concepts of the object-orientated programming paradigm as types, namespaces, classes and interfaces are part of it, JavaScript types are adapted to .NET types and JavaScript wrappers have been implemented to support asynchronous server communication. [33] The *Client Script Library* of ASP.NET AJAX consists of several JavaScript files: *ASP.NET AJAX.js* and *ASP.NET AJAXRuntime.js* scripts implement the script core, base class library, component model, UI framework and most controls. The script *AJAXCompat.js* is provided for compatibility issues with the Firefox browser and other Mozilla-based browsers. Additionally the script *ASP.NET AJAXCompat2.js* allows compatibility with the Apple Safari browser. [30] A key benefit of ASP.NET AJAX is that it supports both object-orientated and declarative techniques.

Declarative elements of ASP.NET AJAX

When developing web applications developers usually use both declarative and imperative languages. While HTML for instance is a declarative language, C# and Java are imperative languages. ASP.NET AJAX provides both programming paradigms. For example, when using ASP.NET AJAX the following declarative statement is required [33].

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

Figure 15: ASP.NET AJAX ScriptManager

Server Controls

ASP.NET AJAX provides a set of AJAX-enabled server controls to simplify the development of AJAX-enabled web applications for developers. The following sections describe the *Timer-Control* and the *UpdatePanel-Control* in detail.

- **Timer**

The Timer-Control causes server-requests periodically. For instance, developers can take use of this control to realize a *Counter* control. The code shown in Figure 16 illustrates the implementation of a *Counter* control using the Timer-Control. [33]

```
01 <form id="form1" runat="server">
02  <asp:ScriptManager id="ScriptManager1" runat="server" />
03  <div>
04   <asp:Label ID="Label1" runat="server" Text="Label">
05   </asp:Label>
06  </div>
07  <asp:TimerControl ID="Timer1" Enabled="true" Interval="1000"
08  runat="server" />
09 </form>
```

Figure 16: ASP.NET AJAX Timer – Web Form

The above mentioned code is not part of the HTML specification. This code will be rendered by the ASP.NET AJAX server-engine. The output is HTML code containing JavaScript pieces. The *Code-Behind* file requires the code fragment shown in Figure 17.

```
01 protected void Page_Load(object sender, EventArgs e)
02 {
03   int Counter = 0;
04   if (IsPostBack)
05   {
06        Counter = (int)Session["Counter"];
07        Counter++;
08   }
09   Session["Counter"] = Counter;
10   this.Label1.Text = Counter.ToString();
11 }
```

Figure 17: ASP.NET AJAX Timer – Code-Behind

The basic idea of this implementation is that the current number of the *Counter* control is stored within a session object that stores the current *Counter* value in a variable. As the server contains the most recent value of the *Counter* a server request is indispensible. Each time the page is requested the method *Page-Load* is executed and the *Counter* value will be incremented. After the client obtains the server response the ASP.NET Label control will display the current number.

- **UpdatePanel**

The above shown sample causes a server-request each time the number of the counter is incremented. Consequently the full page is refreshed after each request. This raises a lack of user experience. To solve this problem ASP.NET AJAX offers the *UpdatePanel* control. Using this control allows partial page updates. The key feature of this control is that only those user-interface controls perform updates whose state has changed since the last request. As a consequence there is no full page refresh and updates occur smoothly. To enhance the sample shown in section above the form- section of the web form must be replaced with the following code fragment. [33]

```
01 <form id="form1" runat="server">
02  <asp:ScriptManager ID="ScriptManager1" runat="server" />
03   <div>
04    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
05    <ContentTemplate>
06      <asp:Label ID="Label1" runat="server"
07      Text="Label"></asp:Label>
08      <asp:Timer ID="Timer1" runat="server" Interval="1000">
09      </asp:Timer>
10    </ContentTemplate>
11   </asp:UpdatePanel>
12  </div>
13 </form>
```

Figure 18: Using the ASP.NET AJAX UpdatePanel

The *UpdatePanel* control has a child-element *ContentTemplate*. All user-interface-components inside the *UpdatePanel* control take the advantage of this control. The enhanced sample shown above enables a counter without a full page refresh because the *ASP.NET Label* control und *ASP.NET AJAX Timer* control are sub components of the *UpdatePanel* control. *UpdatePanels* are refreshed after each server request by default. For this reason a web page with multiple *UpdatePanels* causes refreshes on each single *UpdatePanel* even its content has not changed since the last request. [33]

Control Toolkit[7]

The Control Toolkit is a free available collection of AJAX-enabled controls. Developers take use of these controls without implementing any JavaScript source code. The following table shows available controls at the time of the creation of this thesis. [33]

Control	Description
Accordion	Provides multiple expandable and collapsible Panels. One of them is displayed at a time.
AlwaysVisibleControl	Fixes a specific control (e.g. a clock) to a specific position on screen. The control remains at this position even when it is scrolled.
Animation	Provides animation for existing controls (e.g. fade-in effect on a Button)
AutoComplete	Extends a TextBox. A popup panel displays words that begin with the prefix typed into the TextBox.
Calendar	Extends a TextBox. A client-side Calendar control is displayed.

[7] http://www.asp.net/ajax/control-toolkit/ (24th August 2007)

CascadingDropDown	Extends a set of DropDownLists. Each time a value of one of the DropDownLists is selected, appropriate values for all other DropDownlists are retrieved from a web service.
CollapsiblePanel	Allows expanding and collapsing a Panel.
ConfirmButton	Extends a Button (or a derived instance of the class Button) and displays a message when the Button is clicked.
DragPanel	Allows user to drag a Panel to any space of the screen.
DropDown	Provides a SharePoint-style drop-down menu to any control
DropShadow	Adds a shadow to a Panel.
DynamicPopulate	Allows updating the content of a control dynamically.
FilteredTextBox	Prevents a user from entering invalid characters into a TextBox.
HoverMenu	Associates any control with a popup panel do display additional content.
ListSearch	Enables to search items within a ListBox or a DropDownList by typing.
MaskedEdit	Extends a TextBox to restrict the kind of text that can be entered.
ModalPopup	Displays content to the user in a "modal" manner which prevents the user from interacting with the rest of the page.
MutuallyExclusiveCheckbox	Extends a CheckBox controls. A CheckBox gets a RadioButton-like behavior.
NoBot	Provides a bot/spam prevention without requiring any user-interaction.
NumericUpDown	Extends a TextBox. Adds an "up" and "down" Button to increment and decrement the value of the TextBox.
PagingBulletedList	Extends a BulletedList and provides a client-side sorted paging.
PasswordStrength	Extends a TextBox and shows the strength of a password
PopupControl	Can be attached to any control to open a popup window that displays additional content.
Rating	Enables users to select the number of stars that represents their rating.
ReorderList	Allows user to reorder a list by drag & drop.
ResizableControl	Can be attached to any control and allows modifying the size of the control.
RoundedCorners	Applies rounded corners to existing elements.
Slider	Extends a TextBox. It allows the user to choose a numeric value from a finite range.
SlideShow	Targets image controls. It provides Buttons to switch to the next and to the previous image or to stop the show.
Tabs	Creates a set of Tabs that can be used to organize page content.
TextBoxWatermaker	Extends a TextBox. When a watermarked TextBox is empty, it displays a message to the user with a specific CSS style.
ToogleButton	Enables the use of custom images to show the state of the CheckBox.
UpdatePanelAnimation	Allows playing animations both while an UpdatePanel is updating and after it has finished updating.
ValidatorCallout	Enhances the functionality of existing ASP.NET validators.

Table 1: ASP.NET AJAX Control Toolkit Controls

2.4 Model-Driven Development

Software engineers aim at developing innovative software development techniques with the goal of simplifying and accelerating the software development process. The main problems in building software systems are time, costs and error-proneness [28]. Frankel [11] describes typical challenges within the software industry in detail. For instance software projects frequently exceed their budget limit. Consequently the management stops these projects. Other systems prove to be unstable or inflexible over time. Model-Driven Development (MDD) focuses on at least some of these problems. Currently software development is moving towards a model-driven process with its goal to design software at a higher level of abstraction. [16] One of the essential concepts within Model-Driven Development is the term *model*. A model is a simplified, textual or graphical description of an artifact. It should have concrete and non ambiguous semantics. Models are used to represent complex systems. As a model is a simplification of an artifact it helps designers to handle the complexity of real objects. [24]

The fundamental idea of Model-Driven Development is constructing models, specifying transformation rules [18] and generating code and documentation automatically. The OMG's[8] (Object Management Group) *Model Driven Architecture* (MDA[9]), which is a software design approach within Model-Driven Development, defines the terms *Platform-Independent Model* (PIM) and the *Platform-Specific Model* (PSM). The idea of the PIM is to investigate operations and relationships inside a system while hiding details of a specific platform. The PSM takes the PIM as initial point and uses platform-specific features to generate code. As mentioned in [2] Model-Driven Development makes a distinction between the platform-independent model and the platform-specific model. Developing software systems by modeling and transforming models is a more advantageous approach than programming. The fundamental benefits of MDD are portability, interoperability and reusability through architectural separation of concerns. Figure 19 illustrates the relationships between the PIM and PSM.

[8] http://www.omg.org/ (28th September 2007)
[9] http://www.omg.org/mda/ (15th November 2007)

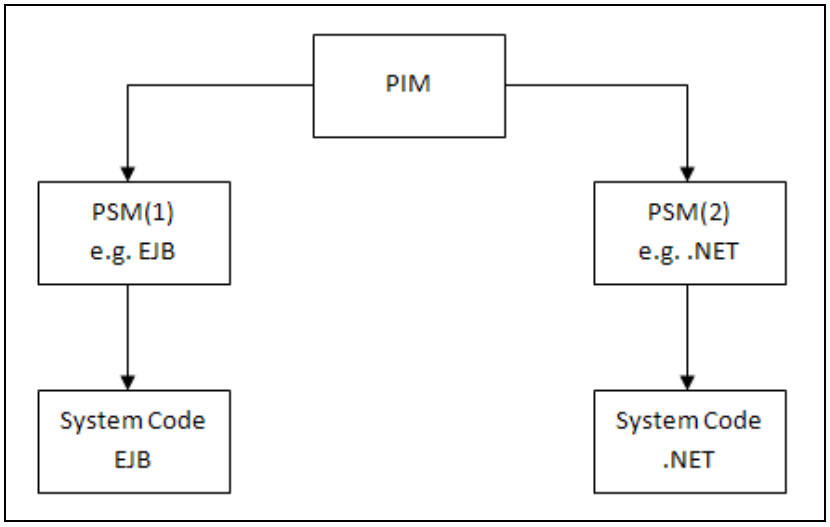


Figure 19: The Relationship between the PIM and the PSM

As shown in Figure 19 developers might realize numerous platform-specific models (PSM) to a corresponding independent model (PIM). This case shows two possible platform specific models: A model based on Java and a model based on the .NET framework. Using platform specific techniques (e.g. EJB on the Java platform) allows the transformation of a PIM into a ready-to-use software system.

2.4.1 Domain Specific Development

Domain Specific Development is a promising approach for enhancing and simplifying the development of applications within a specific domain. The basic idea of Domain Specific Development is to develop a special-purpose language which enables developers to solve various software development problems within a concrete domain. Cook et al. [8] provide a definition for the term *Domain Specific Language*.

> *"A Domain-Specific Language is a custom language that targets a small problem domain, which it describes and validates in terms native to the domain."*

The following example shows a specific domain and its specific language to solve typical software development tasks within this domain. Finding every occurrence of a particular pattern of characters within a file is a well-know domain for developers. For instance a typical task within the domain is to filter all email addresses. Using the special-purpose textual language of

regular expressions, this problem can be solved. Instead of developing a complex algorithm using a general-purpose language like Java, C# or UML to solve this problem, the practitioner realizes a special language which allows solving a whole class of similar problems [8]. For instance finding all occurrences of numbers within a file is a similar task. This problem can be solved with little additional effort when using the special-purpose language of *regular expressions.*

A typical task when developing a new software system is the deployment of the developed application. For this purpose developers frequently use so called installer applications. The installation of software generally requires the execution of similar tasks like copying source files or providing specific settings. Figure 20 displays typical installer concerns.

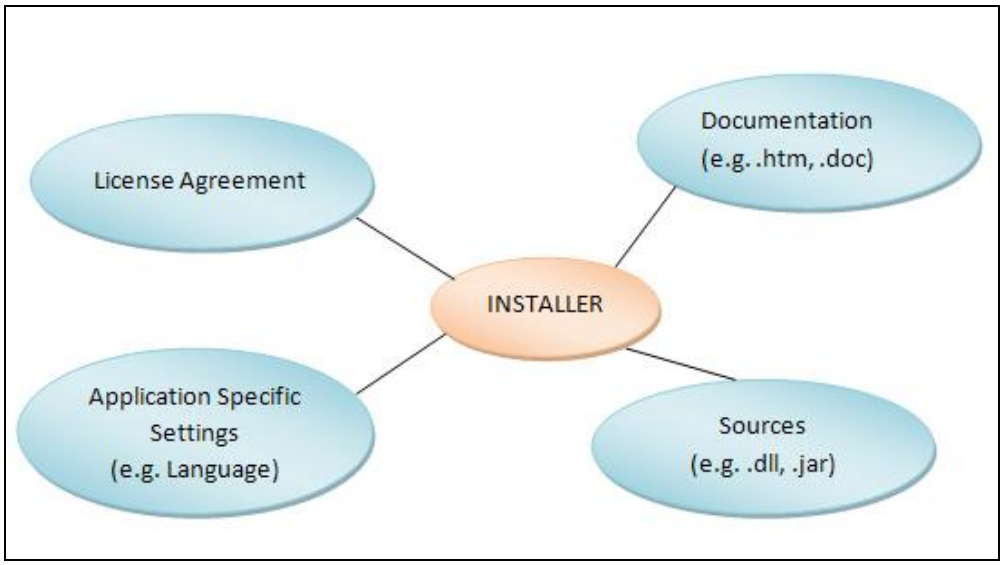


Figure 20: Software Installer Concerns

On the one hand installer applications usually handle similar tasks but on the other hand each installer differs from each other to certain extent. For this reason the use of a DSL might be promising for developing such installer applications. Having a DSL which allows the development of customized installers would be advantageous and accelerate the deployment.

Each task within a domain covers a lot of aspects that are the same. Exactly these parts can be realized once and (re)used for all problems. Each particular task in the future can be solved by creating a model or an expression using the specific language as starting point. Continuously the concrete model is integrated into a fixed part of the solution. The fixed part of the solution is realized by writing code manually using a general purpose language. [8] Essential

concepts, components and their relationships within Domain Specific Development are illustrated in 22.

A Domain Specific Language is also known as *meta-model* because it is itself a model that defines all possible models that can be designed by developers. [2] The meta-model represents the main elements of a specific domain.

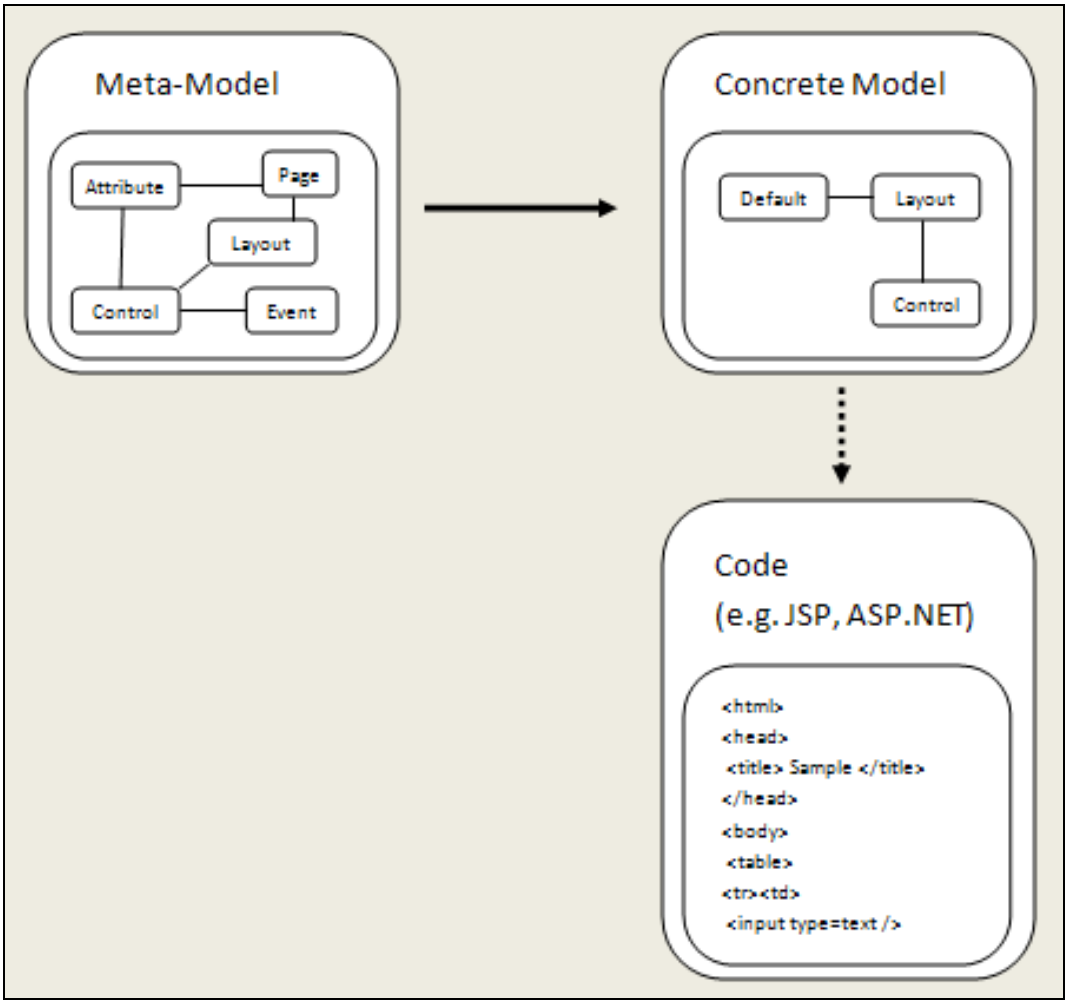


Figure 21: Model Driven Development – Fundamentals

Referring to Cook et al. [8] Domain-Specific Development offers the following benefits. A DSL enables developers to migrate a solution from one technology to another (e.g. from Java to C#). In addition a DSL reduces error proneness which has its origin in writing code manually using general purpose languages. Furthermore a DSL hides details about the implementation technologies. For this reason the DSL's models are accessible to people without experience on these technologies. Another benefit of DSLs is that models offer immediate feedback and

suitability. A further key aspect is that the generated artifacts are not necessarily source code. A suitable model can be used to generate any possible artifact like artifacts for the software documentation. Furthermore a DSL can be used to generate artifacts for multiple technologies. For instance generated artifacts are Java and C# classes. In addition a DSL allows building models that can be used to generate other models.

As mentioned in [8] the benefit of using DSLs varies. DSLs basically can be seen as a more abstract form of source code. A DSL is generally applicable to any software development task. They can be used to configure applications and middleware as well as to initialize and populate databases. In some cases the level of abstraction can be adequate. On the other hand when mapping complex relationships the level of abstraction might be inadequate. A graphical DSL and visual understanding to data addresses this shortcoming to a certain extent. [8]

2.4.2 Domain Specific Language Tools

The *Microsoft Visual Studio DSL Tools* enable developers to build customized modeling tools on the .NET platform. This toolkit allows the definition and implementation of modeling languages. For instance, developers use the DSL Tools to create a specialized language which describes a business process or a user-interface. Consequently the defined language can be used to generate executable source code by writing custom *Text Templates*. *Text Templates* typically use a created model for the generation of source code and other artifacts. The DSL Tools contain the following typical features. [8]

- A graphical designer for building domain models.
- A set of code generators that generates code based on a defined domain model and a designer definition.
- A template-based generator that takes a concrete model, which is based on the defined DSL, and customized *Text Templates* as input to generate artifacts.

Figure 22 shows typical issues when developing applications using the DSL Tools.

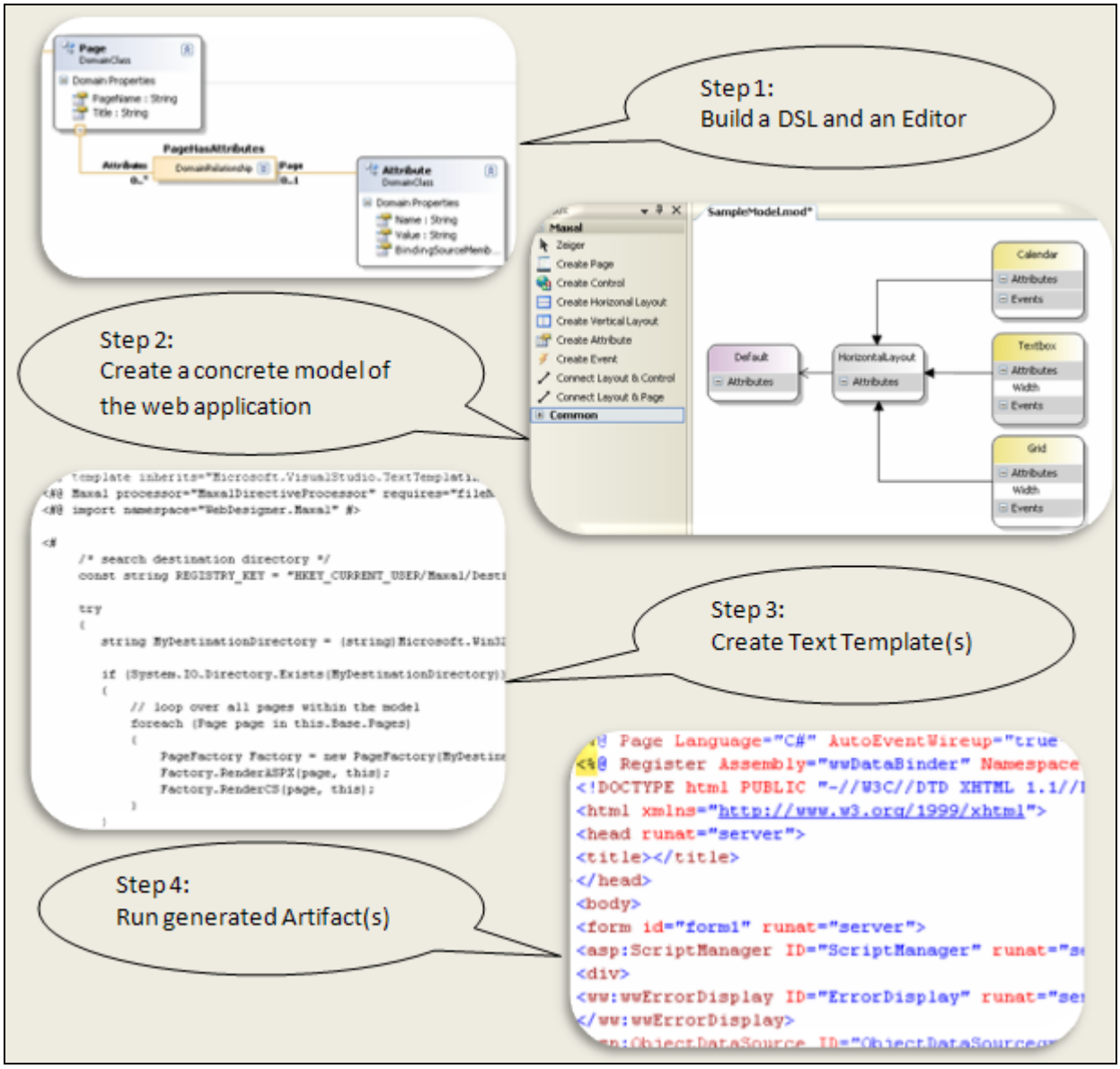


Figure 22: Development Process using the MS DSL Tools

Validation of Constraints

Referring to [8] constraints can be divided into two categories. *Hard constraints* are constraints that the tool prevents the user ever violating. For example, the *width* field of a shape element should allow only valid numbers. *Soft constraints* are constraints that the user is allowed to violate during editing. For example, all elements in a model should have unique identifiers. [8]

The MS DSL Tools allow the validation of *soft constraints*. Each constraint is defined as a method within a partial class. The partial class is for the class of the domain model object that should be validated. The constraint method uses the properties, the relationships, and the role

names that are defined in the domain model. The MS DSL Tools additionally support customization of the validation of constraints. Developers can define how the validation is launched and where messages are directed. [8]

Generating Artifacts

Referring to [8] a key task of Domain Specific Languages is to generate code and other artifacts. For this purpose the MS DSL Tools offer a parameterized template-based approach to generate artifacts. Writing *Text Templates* can be done by writing code using C# or Visual Basic.
Figure 23 shows a typical *Text Template*.

```
01 <#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.
02     ModelingTextTransformation" #>
03 <#@ ClassDiagramSample processor="ClassDiagramSampleDirectiveProcessor"
04     requires="fileName='Sample.mod'" #>
05 <#@ import namespace = "System.Text.RegularExpressions" #>
06 <#@ include file = "C:\Helper.txt"#>
07 <#@ output extension=".txt" #>
08 <#@ assembly name="…\CustomAssembly.dll" #>
09 <#
10    foreach(ModelType type in this.ModelRoot.Types)
11    {
12 #>
13    <#= type.Name#>
14 <#
15    }
16 #>
17 <#+
18  private static string CreateValidName(string typeName)
29  {
20    Regex.Replace(typeName,"[^a-zA-Z]","");
21    return fixedName;
22  }
23 #>
24 . . .
```

Figure 23: A Typical Text Template

From line 1 to line 8 so called *Built-in Directives* are included. *Built-in Directives* provide instructions to the *Text Template* transformation engine. Lines 1 and 2 specify which class should be used as the base class for the generated transformation class. To access a model from a *Text Template*, developers must call the generated directive processor which is provided in line 3 and 4. Calling the generated directive processor makes the classes in a model available to the *Text*

Template code. By using the *import* built-in directive as shown in line 5, developers can refer to .NET types in a text template without providing a fully qualified name. By specifying the *include* built-in directive as shown in line 6, external resources can be used within the Text Template. This feature allows splitting up the content of a *Text Template* into multiple source files. Line 7 specifies the file name extension of the generated output. Additionally Text Templates allow including custom assemblies by specifying the *assembly* built-in directive as shown in line 8.

Between line 9 and line 16 a so called *statement block* is provided. Statement blocks allow developers to structure *Text Templates* to generate code fragments conditionally, or to iterate over data to create code fragments repeatedly. Statement blocks typically contain control markers (<# and #>) to add structure and dynamic behavior [8]. While code outside of the control markers is rendered directly to the output, code between those markers is evaluated [8]. This feature is used within various web development programming techniques like Active Server Pages (ASP), Java Server Pages (JSP) or Hypertext Preprocessor (PHP). Furthermore *Text Templates* provide *Helper Functions* to encapsulate specific code fragments. For instance a *Helper Function* is shown between line 17 and 23.

Chapter 3

MAXAL – A Domain Specific Language for Building Web Applications

MAXAL (*M*odel-Driven *A*ja*x* *A*pplication *L*anguage) is a tool for both modeling of web applications and the generation of source code of internet applications, based on C#, ASP.NET 2.0 and ASP.NET AJAX. Developers can build their web applications using the MAXAL-Designer. Developing web applications using MAXAL is based on the model-driven approach. For this reason the major part of web application development is done by specifying models and their relationships. The designer's backbone is the meta-model shown in Figure 24.

3.1 The Meta-Model

This section introduces a common applicable meta-model for building web applications. The meta-model shown in Figure 24 enables developers designing concrete models of their web applications. Defining an unlimited amount of single web pages is supported. Concrete instances of the classes *Page, HorizontalLayout, VerticalLayout and Control* allow allocating user-defined attributes. The definition of an instance of the class *Layout*, which is a container for user-interface components, is obligatory for each single page. Therefore the DSL offers two different types – *HorizontalLayout* and *VerticalLayout*. The abstract class *Layout* encapsulates properties for the use within concrete instances of its subtypes. This concept allows individual extensions to the meta-model if necessary without causing conflicts with existing layouts. Each concrete instance of the class *Layout* requires the definition of at least one instance of the class *Control*. Furthermore the assignment of instances of the class *Event* to instances of the class *Control* is supported.

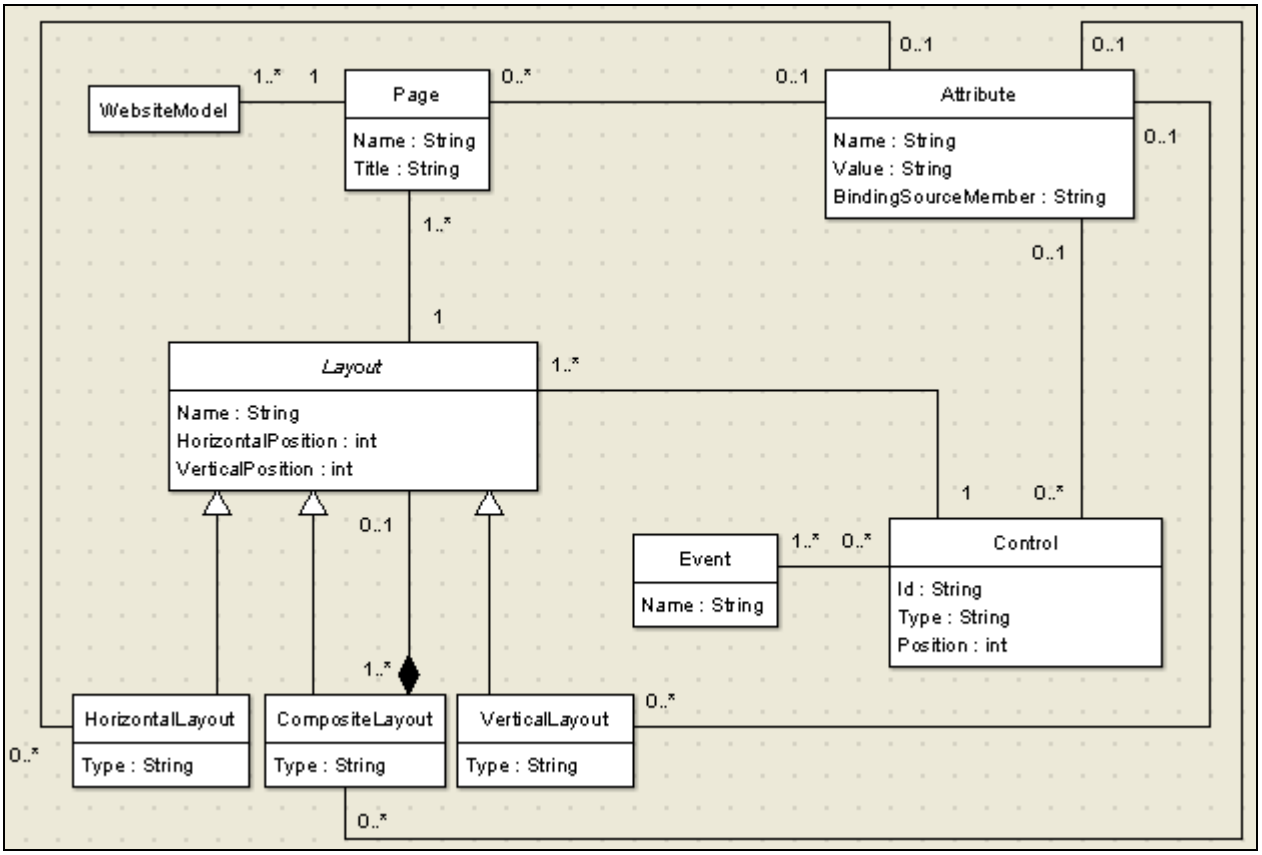


Figure 24: A Meta-Model for Developing Web Applications

An example of a concrete model based on the meta-model in Figure 24 is shown in Figure 25. The shown model of a web application contains one page with two user-interface controls. Basically the web page displays a label and a textbox horizontally. Figure 26 shows the web page after transforming this concrete model into source code, rendering it on a web server and showing it in a browser. The web page's HTML code is rendered automatically after the client receives the response from the server. The Figures 27 – 32 illustrate typical rendered HTML code.

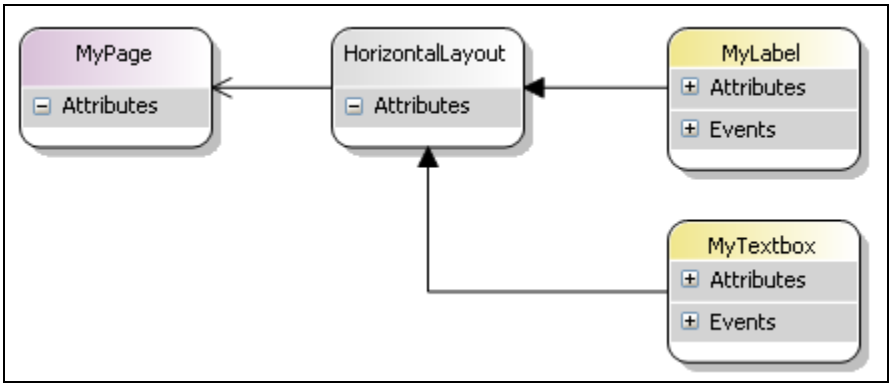


Figure 25: The Model of a Simple Web Page

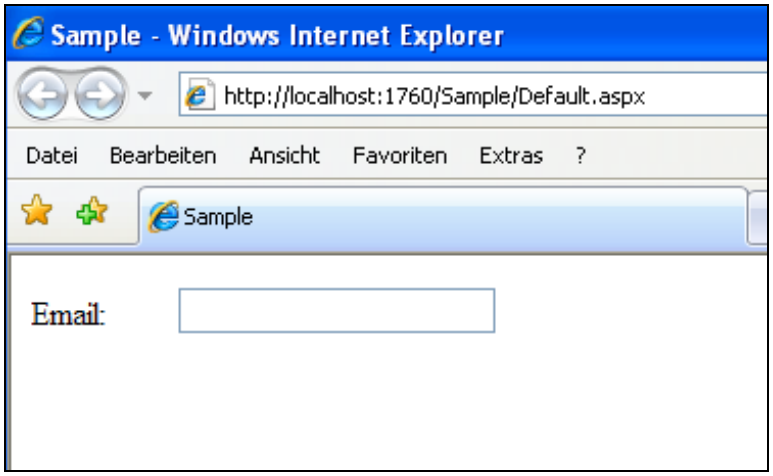


Figure 26: The Browser View of the Simple Web Page

WebsiteModel

The element *WebsiteModel* specifies the root of the meta-model. All other elements within the meta-model are child components and belong to the element *WebsiteModel*.

Page

The element *Page* specifies a web page. An instance of the element *Page* is a single web page which is identified by its attribute *Name*. The predefined attribute *Title* describes the title of a web page. After generating the web application the specified title will be inserted as child element of the HTML *head* tag of the web page. Furthermore additional attributes can be optionally assigned to an instance of a *Page*. Figure 27 shows a possible example of a *Page* element.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title> Page </title>
 </head>
 <body>
 </body>
</html>
```

Figure 27: DSL – A Typical *Page* Implementation

Attribute

Attribute specifies a property value for various instances within a model. Instances of the class *Attribute* can be assigned to instances of the classes *Page*, *HorizontalLayout*, *VerticalLayout* and *Control*. Some classes within the meta-model contain predefined, compulsory attributes, which do not belong to this class. By using instances of the class *Attribute* instances of the classes *Page*, *HorizontalLayout* and *VerticalLayout* are extensible to a certain extent. Figure 28 shows Figure 27 with three instances of the class Attribute. The example contains three additional HTML tags.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-15">
  <meta name="Author" content="Hannes Etl"
  <meta name="Description" content="A DSL for Building Web Applications">
  <title> Page </title>
 </head>
 <body>
 </body>
</html>
```

Figure 28: DSL – A Typical *Attribute* Implementation

Control

Control specifies a user-interface component. Instances of this class are for example input fields, labels, tables etc. For this reason an instance of the class *Control* is basically only a container of a specific user-interface component. An instance of the class *Control* is identified by its attribute *Id*. The attribute *Type* defines the actual desired user-interface component. The attribute *Position* allows developers to define the position of the control within an instance of the class *Layout*. The following example shown in Figure 29 illustrates an instance of the class *Control*.

```
<select>
   <option>Pizza</option>
   <option>Lasagne</option>
   <option>Tortellini</option>
   <option>Spaghetti</option>
</select>
```

Figure 29: DSL – A Typical *Control* Implementation

Event

Event specifies an event for instances of the class *Control*. Instances of this class are for example *onclick*, *onchange* or *onsubmit*. Figure 30 shows an example with a dropdown menu with an *onchange* event. In this case JavaScript is used to handle the event. Each time the selection in the menu changes the specified function *BrowserSelection()* within the JavaScript code fragment is invoked to display the updated selection within an input field.

```
<html>
<head>
<script type="text/javascript">
function BrowserSelection()
{
 var List=document.getElementById("List")
 document.getElementById("Browser").value=
 List.options[List.selectedIndex].text
}
</script>
</head>

<body>
 <form>
  <select id="List" onchange="BrowserSelection()">
   <option>Internet Explorer</option>
   <option>Netscape</option>
  </select>
  <p>Selected Browser: <input type="text" id="Browser"></p>
 </form>
</body>
</html>
```

Figure 30: A Typical *Event* Implementation

Layout

The abstract class *Layout* specifies a class of layout types. It defines the attributes *Name*, *HorizontalPosition*, *VerticalPosition and Type* which are available for instances of subtypes of the class *Layout*. The attributes *HorizontalPosition* and *VerticalPosition* allow developers of web applications to define the absolute position of a concrete layout on the screen. The attribute *Type* enables developers to define whether the layout should be an instance of the class *HorizontalLayout*, *VerticalLayout* or *CompositeLayout*.

HorizontalLayout

The *HorizontalLayout* inherits from the abstract type *Layout*. An instance of the class HorizontalLayout enables developers of web applications to position instances of the class *Control* on a single web page horizontally. The generated code of a *HorizontalLayout* might be realized using a HTML table combined with CSS. Figure 31 shows an example with three buttons which are positioned horizontally. Furthermore in this case the absolute position of the layout on the screen is set using CSS.

```
. . .
<table style="position: absolute; left: 100px; top= 100px">
 <tr>
  <td> <input type="button" name="ButtonLeft" /></td>
  <td> <input type="button" name="ButtonMiddle" /></td>
  <td> <input type="button" name="ButtonRight" /></td>
 </tr>
</table>
. . .
```

Figure 31: A Typical *HorizontalLayout* Implementation

VerticalLayout

The *VerticalLayout* inherits from the abstract type *Layout*. An instance of the class *VerticalLayout* enables developers of web applications to position instances of the class *Control* on a single web page vertically. The generated code of a *VerticalLayout* might be realized using a HTML table combined with CSS. Figure 32 shows an example with three buttons which are positioned vertically.

```
. . .
<table style="position: absolute; left: 100px; top= 100px">
 <tr>
  <td> <input type="button" name="ButtonTop" /></td>
 </tr>
 <tr>
  <td> <input type="button" name="ButtonCenter" /></td>
 </tr>
 <tr>
  <td> <input type="button" name="ButtonBottom" /></td>
 </tr>
</table>
. . .
```

Figure 32: A Typical *VerticalLayout* Implementation

CompositeLayout

A *CompositeLayout* allows nesting of layouts and subsequently of its controls. It is based on the *Composite*[10] design pattern. Other layouts like the previous introduced layouts *HorizontalLayout* and *VerticalLayout* might be assigned to an instance of the *CompositeLayout.* The content of all encapsulated layouts belong to the same update context. For the reason only the embedded content of a *CompositeLayout* is refreshed after a server request. User-interface controls which are not assigned to the *CompositeLayout* remain unchanged.

3.2 A Prototype Implementation of MAXAL

As the *MS Visual Studio DSL Tools*[11] are especially designed for developing DSLs and DSL-related applications on the .NET platform, this tool is a fundamental component for developing the prototype implementation.

As already mentioned in Section 2.4.1 a core feature of DSLs is that developers have the opportunity to choose any platform for realizing an implementation of a DSL. At the same time the generation of artifacts is not restricted to one platform. For this reason the .NET platform is not the only option. Developers might decide developing their DSLs and generating artifacts using Java or any other destination platform. For instance when developing on the Java platform, *openArchitecureWare*[12] is a promising framework for realizing Eclipse-based DSLs [7]. Java Server Pages (JSP) and Java Server Faces (JSF) based web applications might be generated.

Figure 33 shows basic steps for developing web applications using MAXAL. The first task is to build a concrete model of the web application within the MAXAL-Designer (see Section 3.2.1). For this purpose developers typically create numerous instances of web application components such as *Pages*, *Layouts* and *Controls*. A further task is to assign instances of *Attributes* and *Events*. Step 4 in Figure 33 illustrates how an abstract instance of the class *Control* becomes a specific user-interface control. As MAXAL provides a listbox for this mapping, developers need to choose the appropriate control from the list.

[10] http://www.dofactory.com/patterns/PatternComposite.aspx (8.November 2007)
[11] http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx (28th August 2007)
[12] http://www.openarchitectureware.org/ (28th August 2007)

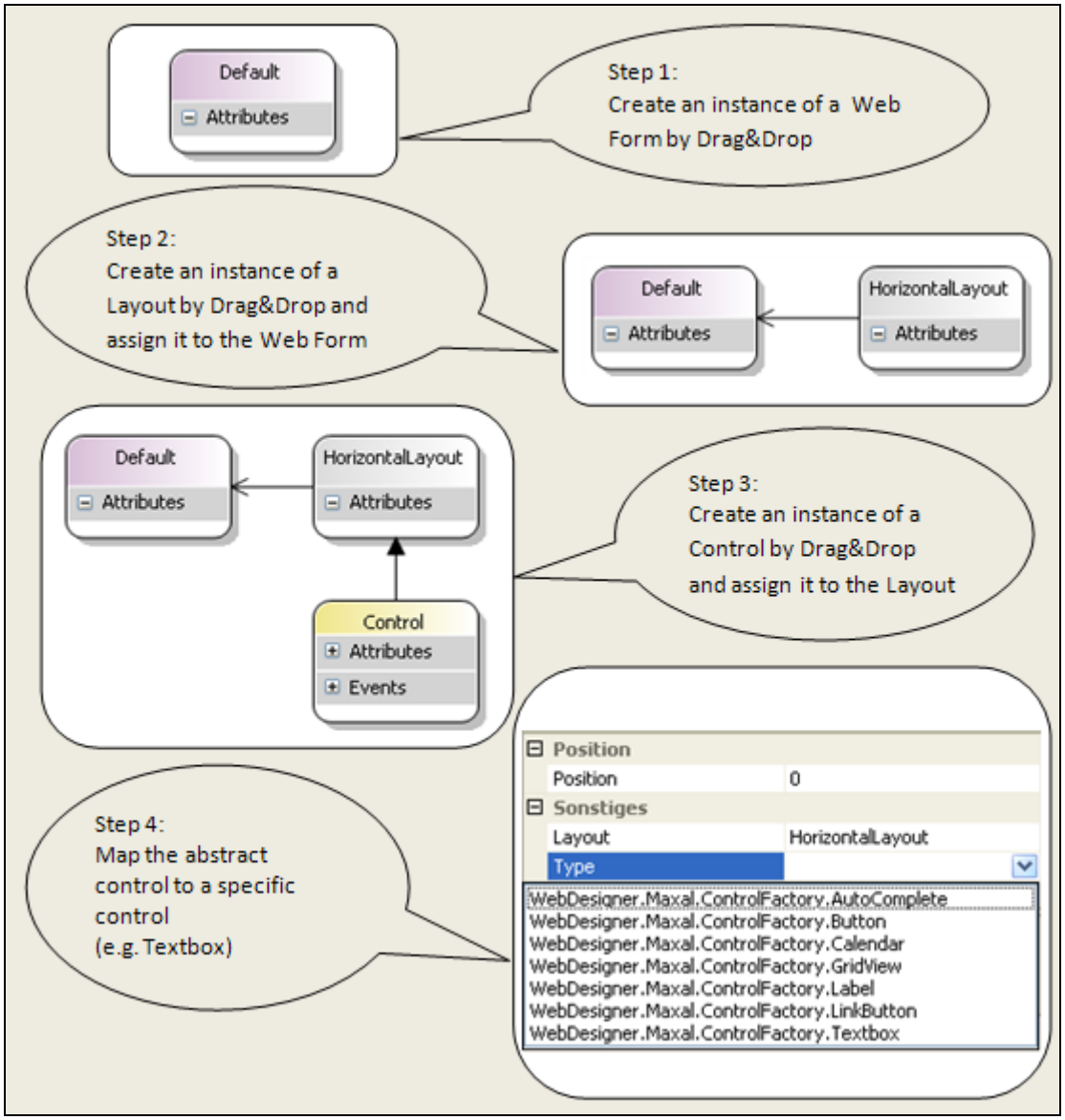


Figure 33: MAXAL Essentials

The following sections introduce the designer's characteristics, components, features and capabilities.

3.2.1 The MAXAL-Designer

The MAXAL-Designer is a visual development environment (see Figure 34) for developing AJAX-enabled web pages. Developers create their web applications using the MAXAL-Designer by specifying a model of their web application. A model contains instances of pages, controls and layouts. The MAXAL-Designer enables developers defining relationships between those

instances and assigning properties and events to controls and layouts. The following sections cover features, subcomponents and characteristics of the MAXAL-Designer.

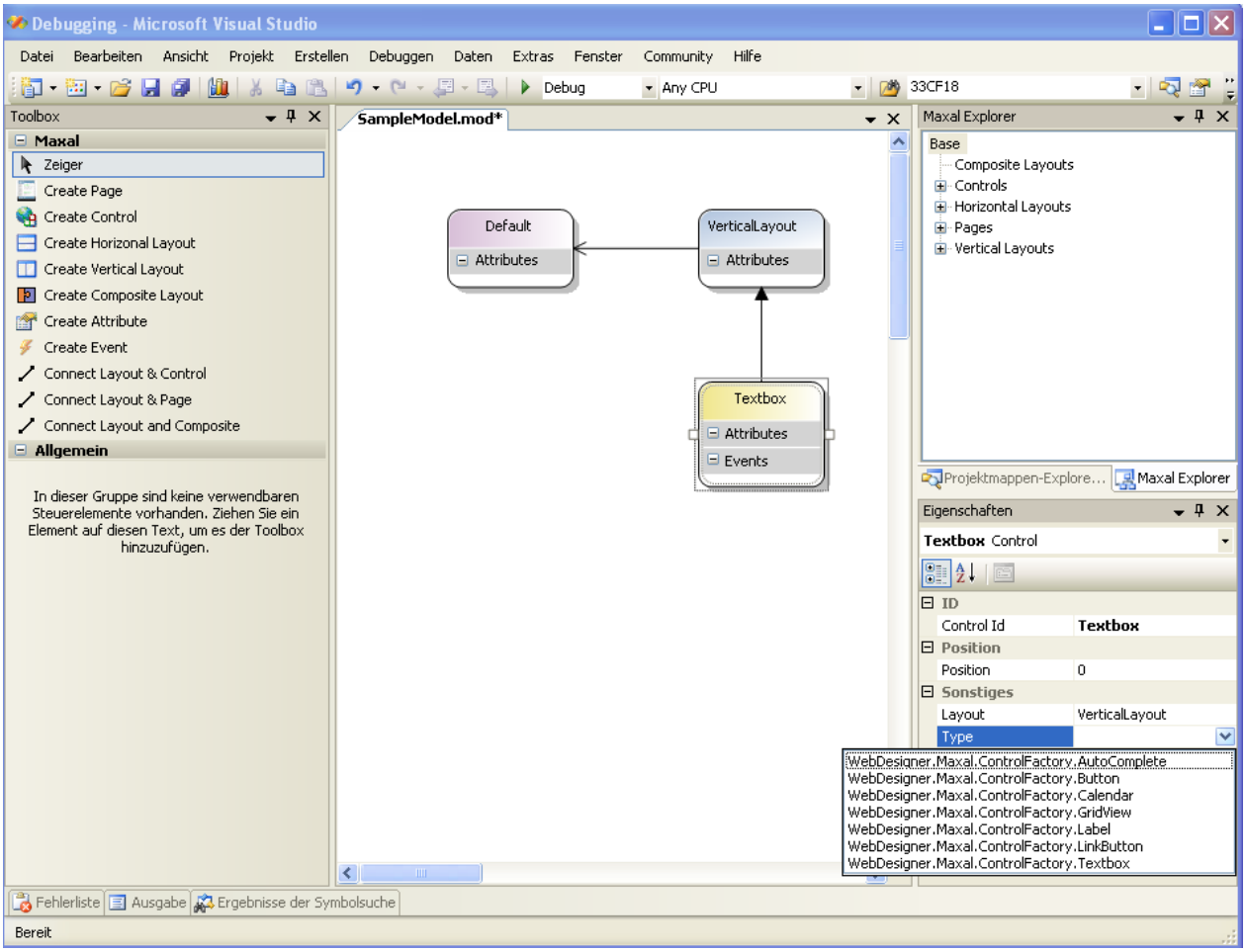


Figure 34: The MAXAL-Designer – A Visual Development Environment

3.2.2 Features

- Model-Driven Development

  This feature is a core feature of MAXAL. A good portion of the web application development can be done by specifying models and their relationships. These models are even accessible for people who are not familiar with the underlying technology. It offers a wide range of advantages for web application developers as mentioned in Section 2.4.1.

- Approved user-interface controls

  MAXAL provides a set of user-interface components that can be used by web application developers. The prototype implementation offers typical ASP.NET controls and ASP.NET AJAX components with their basic functionalities.

- AJAX

  This feature is also a core feature of MAXAL and a fundamental design concept. The basic idea is that all user-interface controls are AJAX-enabled. Developers of new user-interface controls do not have to pay attention on AJAX. This feature is integrated within MAXAL. When generating applications for ASP.NET all user-interface controls are embedded in instances of ASP.NET AJAX *UpdatePanels*. For this reason developers get rich AJAX-enabled user-interfaces without additional effort.

- Positioning

  This feature enables developers to build rich user-interfaces. It allows developers to define the position of their user-interface controls on the screen.

- Extensibility

  Extensibility is a fundamental design concept of MAXAL. Developers have the opportunity to implement their own user-interface controls and add those to MAXAL. Additionally MAXAL contains predefined layouts to simplify developing user-interfaces. These predefined layouts - the *HorizontalLayout* and the *VerticalLayout* - can be modified and adapted to developers requirements at any time with reasonable effort. For this purposes Section 3.2.11 introduces a mechanism for extending MAXAL with new user-interface components.

- Data Binding

  Most user-interface controls like an ASP.NET TextBox or an ASP.NET GridView are typically used to display data to the user or to retrieve data from the user. For this reason MAXAL supports two different mechanisms for binding data to user-interface controls. Details on these mechanisms are described in Section 3.2.5.

3.2.3 Built-In Controls

The prototype implementation of MAXAL provides a set of ASP.NET and ASP.NET AJAX based user-interface components with their basic functionalities that can be used, adapted and modified by developers for their own web applications. The following list shows all *built-in* controls. Most of them are typical ASP.NET controls. The available ASP.NET AJAX AutoComplete user-interface component is part of the ASP.NET AJAX Control Toolkit.

- ASP.NET TextBox

  This control allows users to input and read text. Binding data to this control is fully supported.

- ASP.NET GridView

  The ASP.NET GridView is typically used to display comprehensive data volumes that are retrieved from any data sources (e.g. a database). Binding data to this control is supported.

- ASP.NET Calendar

  The ASP.NET Calendar is a typical server-side control to allow a user to choose a date. This control is enhanced insofar as it is AJAX-enabled. After selecting a date no page interruption occurs.

- ASP.NET AJAX AutoComplete

  The ASP.NET AJAX AutoComplete control extends an ASP.NET TextBox. A popup panel displays words that begin with the prefix typed into the ASP.NET TextBox. This control retrieves its data from an ASP.NET Web Service. For this purpose the control provides appropriate properties to configure the URL to the ASP.NET Web Service.

- ASP.NET Label

  The ASP.NET Label control is typically used to display specific data for descriptions. Binding data to this control is fully supported.

- ASP.NET Button

  The ASP.NET Button control is used to display a push button to submit a request. As the ASP.NET Button is usually not used to display data, binding data to this control is not supported.

- ASP.NET LinkButton

  The ASP.NET LinkButton control looks like a HyperLink control but has the same functionality as the ASP.NET Button control. As the ASP.NET LinkButton is usually not used to display data, binding data to this control is not supported.

3.2.4 The Generation Process of Artifacts

For generating artifacts *Text Templates* are typically used. MAXAL contains initial invocations of methods for the generation process – within MAXAL the code generation is initiated within the file *Maxal.tt*. The actual code generation implementation of MAXAL is encapsulated within various .NET assemblies. The core of the Text Template *Maxal.tt* is shown in Figure 35. Within this code fragment an instance of the class *PageFactory* is created and the generation of the modeled web application is initialized by invoking two methods. While the first method is responsible for generating the web form, the second method initializes the generation of the Code-Behind file.

A set of custom classes is responsible for the code generation process. These classes are integrated within the implementation of MAXAL. User-interface components provide the only exception. User-interface controls and layouts are loaded into the MAXAL-Designer during the execution of the code generation process. Controls and layouts are encapsulated within separate assemblies.

```
. . .
// loop over all pages within the model of the web application
foreach (Page page in this.WebsiteModel.Pages)
{
     PageFactory Factory = new PageFactory(MyDestinationDirectory, this);
     Factory.RenderASPX(page, this);
     Factory.RenderCS(page, this);
}
. . .
```

Figure 35: The Text Template *Maxal.tt*

The original approach when using the MS DSL Tools for generating code is that each *Text Template* generates exactly one artifact. As MAXAL uses only one *Text Template* it differs from the original approach. The MAXAL-Designer offers developers to create their web applications with an unlimited amount of single web pages. MAXAL generates executable *Web Forms* and *Code-Behind* files for each single instance of the class *Page* within the designed model of the web application. For this reason the generated artifacts are stored in a specified location.

3.2.5 Bind Data to User-Interface Controls

Web applications typically contain data-centric user-interface controls like for example a drop down menu or a table. For this purpose the .NET platform as other technologies provides a mechanism to dynamically display and modify the data of a web application – the *Data-Binding*.

Within MAXAL each Web Form contains exactly one Code-Behind files which typically contains numerous method stubs and variables. Within the Code-Behind file developers return a data source with a method stub for specific user-interface controls which support *SimpleDataBinding*. For controls that support *ComplexDataBinding* a separate *Object* class is auto-generated. Within this class the method stub *CreateDataSource* is auto-generated. Developers use this method stub to return a data source.

As most user-interface controls display or retrieve data from the user MAXAL supports two different facilities to bind data to user-interface controls. The following sections describe both mechanisms with its characteristics and the utilization when implementing a custom control.

- **SimpleDataBinding**

  The basic idea of *SimpleDataBinding*, described in Section 3.2.5, is to bind data to simple user-interface controls like ASP.NET TextBoxes, ASP.NET Labels or ASP.NET DropDownLists. Microsoft offers the custom control *wwDataBinder*[13] to simplify data binding. This control is used within MAXAL to realize the *SimpleDataBinding* mechanism. Figure 36 and Figure 37 show how to bind data to the property *Text* of an ASP.NET TextBox using the *wwDataBinder* control. Figure 37 is a code fragment of the corresponding *Code-Behind* file to the Web Form shown in Figure 36.

[13] http://msdn.microsoft.com/msdnmag/issues/06/12/ExtendASPNET/Default.aspx?loc=en (30th August 2007)

```
. . .
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<ww:wwDataBinder ID="DataBinder1" runat="server">
     <DataBindingItems>
           <ww:wwDataBindingItem runat="server"
           BindingMode="OneWay"
           BindingSource="BindingSource1.Current"
           BindingSourceMember="Text"
           ControlId="TextBox1"
           BindingProperty="Text">
           </ww:wwDataBindingItem>
     </DataBindingItems>
</ww:wwDataBinder>
. . .
```

Figure 36: The Implementation of *SimpleDataBinding* – Web Form

```
. . .
public partial class Page1 : System.Web.UI.Page
{

 BindingSource BindingSource1 = new BindingSource();

 protected void Page_Load(object sender, EventArgs e)
 {
  // Assign datasource & datasource index(default: 0)
  this.BindingSource1.DataSource = CreateBindingSource1();
  this.BindingSource1.Position = 0;
  this.DataBinder1.DataBind();
 }

 private object CreateBindingSource1()
 {
  // TODO: Implement a DataSource
  return new NotImplementedException();
 }
. . .
```

Figure 37: The Implementation of *SimpleDataBinding* – Code-Behind Class

Developers who intend to build their own user-interface controls using the *SimpleDataBinding* mechanism may address the following coding guidelines.

- To simplify the development of custom controls the complete source code within the *Code-Behind* file is generated automatically by MAXAL. For this purpose MAXAL provides unique identifiers and variables. Details about this facility are described in Section 3.2.7.

- To inform MAXAL about the control's implementation of the *SimpleDataBinding* mechanism, the method *IsSimpleDataBindingSupported* may return the value *true*. Note: The method *IsSimpleDataBindingSupported* is defined within the interface *IControl*. For this reason this method may be implemented within a custom control.

- Developers may provide the *wwDataBinder* elements with its sub-elements within the .aspx file for each property of the user-interface control that should retrieve data from a data source. For this purpose developers of web applications define within the MAXAL-Designer when building the model of the web application which properties should retrieve their data from data sources. All required attributes like the *BindingSource*, *BindingSourceMember*, *ContolId*, *BindingProperty* and the *ID* of the *wwDataBinder* control are provided by the model of the web application during the generation process.

- **ComplexDataBinding**

  The basic idea of *ComplexDataBinding* is to provide data for data-centric user-interface controls like the *ASP.NET GridView* control. The *ComplexDataBinding* mechanism is based on the *ASP.NET ObjectDataSource* component. This mechanism provides a specific object with a method where developers specify their data source. MAXAL creates the object including its methods and member variables. For this purpose MAXAL generates unique identifiers for the object's methods and member variables and stores those into the model of the web application. During the execution of the generation process MAXAL retrieves all data of the model including the generated identifiers and creates appropriate artifacts. Details about generating unique identifiers and variables are described in Section 3.2.7. For user-interface controls that support *ComplexDataBinding*, MAXAL generates one additional file, which contains the implementation of the data object and a specific method stub for specifying a custom data source. Developers return a data source with this stub. Figure 38 and Figure 39 show the implementation of the ASP.NET GridView control with the *ComplexDataBinding* mechanism.

```
. . .
<asp:ObjectDataSource ID="ObjectDataSource1"
                      runat="server"
                      SelectMethod="CreateDataSource"
                      TypeName="GridViewData"
                      DataObjectTypeName="GridViewData">
</asp:ObjectDataSource>
 . . .
<asp:GridView ID="GridView1"
              runat="server"
              DataSourceID="ObjectDataSource1">
</asp:GridView>
. . .
```

Figure 38: The Implementation of *ComplexDataBinding* – Web Form

```
public class GridViewData
{
     // default constructor
     public GridViewData() { }

     // method stub
     public object CreateDataSource()
     {
          // TODO: provide a data source
          throw new NotImplementedException();
     }
}
```

Figure 39: The Implementation of *ComplexDataBinding* - Object

The ASP.NET GridView control provides the property *DataSourceID*. The value of this property references the provided ASP.NET ObjectDataSource. The declaration of the ASP.NET ObjectDataSource provides the property *SelectMethod* which points to the method *CreateDataSource* which provides the actual data source.

Developers who intend to build their own user-interface controls using the *ComplexDataBinding* mechanism may address the following coding guidelines.

- To simplify the development of custom controls the complete source code for the *ComplexDataBinding* mechanism is generated automatically by MAXAL. For this purpose MAXAL provides unique identifiers and member variables. Details about this facility are described in Section 3.2.7.

- To inform MAXAL about the control's implementation of the *ComplexDataBinding* mechanism, the custom control returns the value *true* when MAXAL invokes the method *IsComplexDataBindingSupported*.

3.2.6 Generated Artifacts

Figure 40 shows all components and their relationships for all generated artifacts.

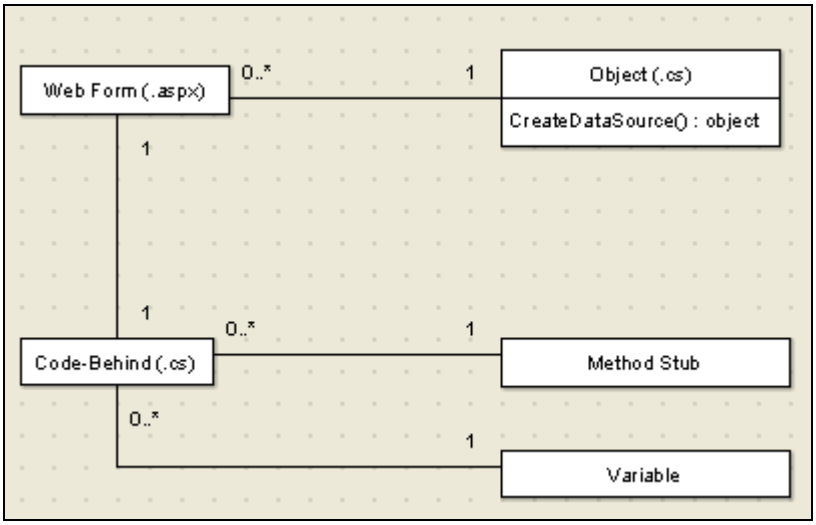


Figure 40: A UML Diagram of Generated Artifacts and Components

3.2.7 Generating unique Identifiers

Figure 40 shows that generally each *Code-Behind* file contains numerous variables and method stubs. For this purpose MAXAL has to generate unique identifiers for all variables and method stubs. This facility is realized by accessing the model of the web application programmatically to store all generated unique identifiers. Accessing the model requires modifying the *DSL Tools In-Memory Store*. This is generally done by initializing a transaction within a partial class. The *DSL Tools In-Memory Store* is a set of APIs which provides a set of basic facilities to access a DSL like creation, manipulation and deletion of model elements and links [8]. Figure 41 illustrates how MAXAL implements this functionality to add generated unique identifiers to the model. The custom class *ValidationHelper* provides methods that generate these unique identifiers.

```
using Microsoft.VisualStudio.Modeling;
using System.Collections.ObjectModel;
using Microsoft.VisualStudio.Modeling.Validation;

namespace WebDesigner.Maxal
{
 public partial class Control
 {
  private void CreateIdentifier(ValidationContext context)
  {
   using (Transaction GenerateIdentifier =
   Store.TransactionManager.BeginTransaction("Create unique identifiers"))
   {
    // create instance of helper class, which generates unique IDs
    ValidationHelper Helper = ValidationHelper.GetInstance();

    // assign unique IDs to properties of the model
    this.Store.DomainDataDirectory.FindDomainProperty(
        Control.BindingSourceIdDomainPropertyId).SetValue(this,
        Helper.GetUniqueBindingSourceId(this.WebsiteModel.Controls.Count));

    this.Store.DomainDataDirectory.FindDomainProperty(
        Control.DataBinderIdDomainPropertyId).SetValue(this,
        Helper.GetUniqueDataBinderId(this. WebsiteModel.Controls.Count));

    // assign unique ID to property ObjectDataSourceId
    this.Store.DomainDataDirectory.FindDomainProperty(
        Control.ObjectDataSourceIdDomainPropertyId).SetValue(this,
        string.Format("{0}{1}", "ObjectDataSource", this.ControlId);

    // Commit the transaction and store modifications in the model
    GenerateIdentifier.Commit();
   }
  }
```

Figure 41: Generate Identifiers for Variables and Method Stubs

3.2.8 Displaying Custom Attributes

As mentioned in at the beginning of Section 3.2 developers need to map abstract user-interface controls to specific user-interface controls. Within the MAXAL-Designer developers perform the assignment of specific user-interface controls by setting the attribute *Type* which is a predefined property of the class *Control*. Figure 42 illustrates this mapping.

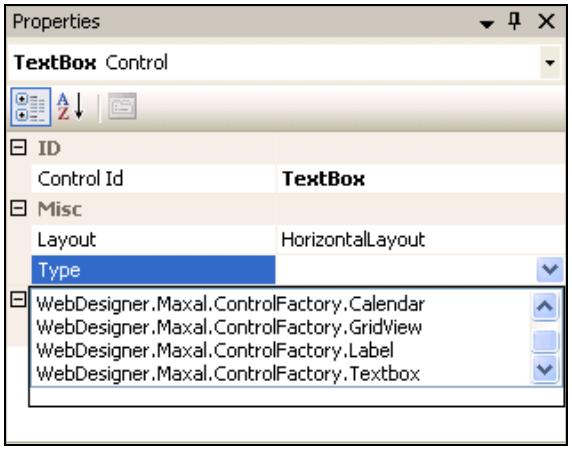


Figure 42: Mapping an Abstract Instance of the Class Control to a Specific UI Control

To enhance user-acceptance the MAXAL-Designer displays a list with all available user-interface controls. Front-end controls typically contain a comprehensive set of properties and events to enable developers to define their appearance and behavior. To simplify the assignment of properties and events MAXAL supports a selection of all properties and events that are available for a specific user-interface control. The implementation of this feature is for instance realized within the source file *PropertyControlTypeNameEditor.cs*. This class inherits from the class *UITypeEditor* and allows accessing the user-interface of the MAXAL-Designer. Its source code is shown in Appendix B.

3.2.9 Rendering

The last step within the development process of web applications using MAXAL is to generate source code which is specified by the developer's model within the MAXAL-Designer. For this purpose MAXAL initializes numerous processes for generating the required code files. As the focus is primarily on ASP.NET Web Forms and Code-Behind files are generated. Figure 43 and the following sections describe the rendering of both steps in detail. The following list describes step by step each stage of the generation process of the Web Form (.aspx file). The generation of the Code Behind file (.cs file) uses the same principle.

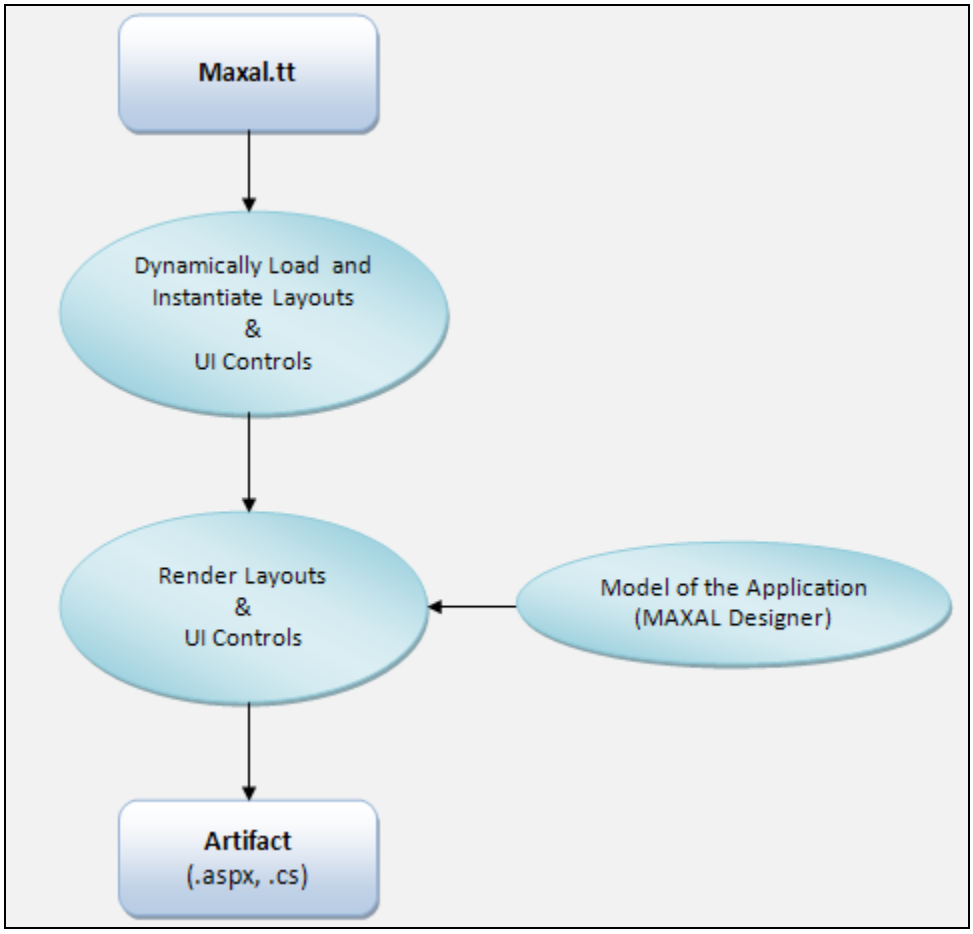


Figure 43: The MAXAL Rendering Process

- An instance of the class *PageFactory* is created within the Text Template *Maxal.tt* and its method *RenderASPX* is invoked. In the background the class *PageFactory* loads all available layouts (HorizontalLayout, VerticalLayout) into the memory.

- By using the .NET Reflection API and the interface *ILayout*, which is introduced in Section 3.2.11, the method *Render* of a specific layout is invoked.

- An instance of the class *ControlFactory* is created. In the background all available user-interface controls are loaded into the memory.

- By using the .NET Reflection API and the interface *IControl*, which is introduced in Section 3.2.11, the method *RenderControl* of a specific user-interface control is invoked. Within this method the actual code generation of the control is performed.

- The generated code is written to a specific file.

In addition to get a valid executable *Web Form,* between each stage specific parts of the generated file like the Header, the Footer, etc. are written to the destination file.

3.2.10 Validation of Models

MAXAL implements a comprehensive set of validation methods to prevent the user to build invalid models of their web applications. MAXAL provides constraints on instances of the classes *Page*, *VerticalLayout*, *HorizontalLayout*, *Attribute*, *Event* and *Control*. Furthermore the class *ValidationHelper.cs* is part of the validation mechanism. As all validation methods basically do not consider overlapping relationships between various instances of the model, the class *ValidationHelper.cs* is available. This class is based on the *Singleton Pattern*[14] and provides methods for constraint validation especially between various instances of all model shapes.
Table 2 shows all implemented constraints.

Constraint	Component	Description of the Constraint
UniqueId	Page	All instances that are assigned to an instance of the class *Page* must have unique identifiers.
PageLayout	Page	Each instance of the class *Page* must have at least one instance of the class *Layout*. *(HorizontalLayout* or *VerticalLayout)*
ControlAttribute	Control	Instances of the class *Attribute* that are assigned to an instance of the class *Control* must have unique identifiers.
ControlEvent	Control	Instances of the class *Event* that are assigned to an instance of the class *Control* must have unique identifiers.
ControlMap	Control	Each instance of the class *Control* has to be mapped to a specific user-interface control by setting the predefined attribute *Type*.

Table 2: Constraints of MAXAL Components

Figure 44 and Figure 45 illustrates the pseudo code of constraints implementations on instances of the class *Control*. All other constrains have similar implementations. Figure 44 validates the name of each instance of the class *Control* within the model.

[14] http://msdn2.microsoft.com/en-us/library/ms954629.aspx (1st September 2007)

```
ControlNameConstraint(CONTEXT context)
{
   if (Name IS EMPTY)
   {
     context.PRINT "Name is not specified.";
   }
}
```

Figure 44: The Constraint Validation of the Control Name

Figure 45 shows how to check that each instance of the class *Attribute* is identified by its *Name*. Firstly each instance of the class Attribute may provide the attribute *Name*. Secondly the shown method verifies if duplicates are defined within the model.

```
ControlAttributeConstraint(CONTEXT context)
{
   foreach (ATTRIBUTE Attribute in AttributeList)
   {
      if (Attribute.Name IS EMPTY)
      {
        context.PRINT "Name is not specified.";
      }
   }

   DICTIONARY Errors;

   foreach (ATTRIBUTE AttributeCompared in AttributeList)
   {
      foreach (ATTRIBUTE CurrentAttribute in AttributeList)
      {
         if (CurrentAttribute != AttributeCompared AND
            CurrentAttribute.Name == AttributeCompared.Name)
         {
            Errors.ADD(CurrentAttribute.Name, CurrentControl.Name);
         }
      }
   }

   foreach (ERROR Error in Errors)
   {
      context.PRINT Error.Attribute + "is not unique in" + Error.Control;
   }
}
```

Figure 45: The Constraint Validation of the Control Attribute

3.2.11 Extensibility

This section deals with MAXAL and its extensibility. As already mentioned in previous sections extensibility is a key design concept of MAXAL. The idea is to allow developers to build their own user-interface controls and add those to the MAXAL-Designer without modifying the existing implementation of MAXAL. For this purpose MAXAL uses the .NET Reflection API to load assemblies which contain the implementations of user-interface components. Therefore MAXAL retrieves all assemblies from a source directory which has been configured before.

All user-interface controls that implement the interface *IControl* are treated as MAXAL-compatible controls and will be available within the MAXAL-Designer when developers build their models of their web applications. Section 3.2.11 deals with details of the interface *IControl*. Figure 46 shows pseudo code of the class *ControlFactory*. The pseudo code illustrates how available user-interface controls are loaded into the MAXAL-Designer.

The basic idea of the interfaces *IControl* and *ILayout* is that custom controls which are built by developers, implement so called *renderers*. *Renderers* are methods within custom classes that a responsible for the code generation of the actual user-interface components. *Renderers* specify the appearance and behavior of the available controls. When transforming the model of the web application to source code, MAXAL invokes these *renderers* using the .NET reflection API.

As already mentioned in previous sections MAXAL provides two different layouts – the *HorizontalLayout* and *VerticalLayout*. The implementations of both layouts are encapsulated within the assembly *WebDesigner.Maxal.LayoutFactory.dll* which has to be placed within a specific directory which contains all required assemblies. As all user-interface controls both layouts are loaded into MAXAL using the .NET Reflection API. For this purpose MAXAL provides the interface *ILayout* which is described in Section 3.2.11 in detail. *ILayout* basically defines a set of methods that are invoked by MAXAL during the development of a model of a web application and during the code generation process.

Both user-interface controls and layouts are external components of MAXAL. Insofar developers have the opportunity to modify existing controls and layouts and add at least new user-interface controls to MAXAL without modifying source code of MAXAL.

```
LIST Renderers;

foreach (File in All_Files_In_Directory)
{
   if File is ASSEMBLY
   {
     TYPE [] Types = LOAD_ASSEMBLIES(File)
      foreach Type in Types
     {
        if Type IMPLEMENTS_INTERFACE IControl
        {
           Renderers.ADD(CREATE_INSTANCE(Type))
        }
     }
   }
}
```

Figure 46: Loading Controls using the .NET Reflection API

Interface ILayout

MAXAL provides layouts to simplify positioning of user-interface controls on the screen. For this reason MAXAL defines the interface *ILayout* as shown in Figure 47. Each valid MAXAL layout has to implement *ILayout*. It consists of two methods – *Render* and *GetLayoutProperties*.

```
using System.IO;
using WebDesigner.Maxal;
using Microsoft.VisualStudio.TextTemplating;

namespace WebDesigner.Maxal
{
    public interface ILayout
    {
        void Render(Layout l, StreamWriter s, TextTransformation t);
        object[] GetLayoutProperties();
    }
}
```

Figure 47: Definition of the Interface *ILayout*

- **Method *Render***

The method *Render* generates code of a specific layout within a web form. The signature of *Render* contains three parameters. The first parameter contains an instance of the class *Layout* defined inside the *MAXAL*-Designer with its controls. This parameter is used to get all required

information of a specific layout to generate the desired output. The second parameter is a stream to a specific file. This is the output file where the generated code will be appended. For this reason the method's return value is void. The third parameter *TextTransformation* is used to print errors or warnings to the error pane of the hosting environment in case of invalid model specifications in the MAXAL-Designer.

- **Method *GetLayoutProperties***

The MAXAL-Designer invokes the method *GetLayoutProperties* to retrieve all available properties of a specific layout. This method allows users to choose supported properties of used layouts from a *Listbox* control. Currently neither *HorizontalLayout* nor *VerticalLayout* provide properties. The motive of this method is extensibility. If developers need additional features for existing layouts, properties can optionally be added to change the layouts' appearance.

Interface IControl

IControl defines a specific interface for valid MAXAL controls. Its implementation is shown in Figure 48. Each control used by MAXAL has to implement the interface *IControl*. It consists of six methods – *RenderASPX*, *RenderCS*, *GetControlProperties*, *GetControlEvents*, *IsSimpleDataBindingSupported* and *IsComplexDataBindingSupported*. During the development of a web application and the source code generation process MAXAL invokes these methods to perform both retrieving available attributes, events and supported data binding mechanisms as well as rendering the control. The following sections deal with each method in detail.

```
using System.IO;
using WebDesigner.Maxal;
using Microsoft.VisualStudio.TextTemplating;

namespace WebDesigner.Maxal
{
    public interface IControl
    {
        void RenderASPX(Control c, StreamWriter s, TextTransformation t);
        void RenderCS(Control c, StreamWriter s, TextTransformation t);
        bool IsSimpleDataBindingSupported();
        bool IsComplexDataBindingSupported();
        object[] GetControlProperties();
        object[] GetControlEvents();
    }
}
```

Figure 48: Definition of the Interface *IControl*

- **Method *RenderASPX***

The method *RenderASPX* generates code of a specific user-interface control within a *Web Form* (.aspx file). The first parameter *Control* contains an instance of the class *Control* defined inside the *MAXAL-Designer* with its attributes and events. This parameter is used to get all required information of the control to generate the desired output. The second parameter *StreamWriter* is a stream to a specific file. This is the output file where the generated code will be appended. For this reason the method's return value is void. The third parameter *TextTransformation* is used to print errors or warnings to the error pane of the hosting environment in case of invalid model specifications of the control in the MAXAL-Designer.

- **Method *RenderCS***

The method *RenderCS* generates code of a specific user-interface control within the *Code-Behind-File*. The signature of *RenderASPX* contains three parameters. The first parameter Control contains an instance of class *Control* defined inside the MAXAL- Designer with its attributes and events. Especially the control's events are required to realize the method's stubs for events of the control. The second parameter *StreamWriter* contains the path to the output file. The generated code will be appended to this file. For this reason the method's return value is void. The third parameter *TextTransformation* is used to print errors or warnings to the error pane of

the hosting environment in case of invalid model specifications of the control in the MAXAL-Designer.

- **Method *IsSimpleDataBindingSupported***

The MAXAL-Designer invokes the method *IsSimpleDataBindingSupported* to get information about the control's DataBinding mechanisms. This method indicates whether DataBinding via *wwDataBinder-Control* is implemented or not. If this feature is part of the control the MAXAL-Designer appends required code inside the *Code-Behind-File* automatically. For this reason developers decide on their own if their controls support DataBinding via *wwDataBinder-Control*.

- **Method *IsComplexDataBindingSupported***

The MAXAL-Designer invokes the method *IsComplexDataBindingSupported* to get information about the control's DataBinding mechanisms. This method indicates whether DataBinding via *ASP.NET ObjectDataSource-Control* is implemented or not. If this feature is part of the control the MAXAL-Designer appends required code inside the code-behind class automatically. For this reason developers decide on their own if their controls support DataBinding via *ASP.NET ObjectDataSource-Control*.

- **Method *GetControlProperties***

The MAXAL-Designer invokes the method *GetControlProperties* to retrieve all available properties of a specific control. Developers of controls have to provide an array with all supported properties. This method allows user of MAXAL-Designer to choose supported properties of used controls from a selection.

- **Method *GetControlEvents***

The MAXAL-Designer invokes the method *GetControlEvents* to retrieve all available events of a specific control. Developers of controls have to provide an array with all supported events. This

method allows user of MAXAL-Designer to choose supported events of used controls from a selection.

3.3 Setting up a Demo Application

Before developing a new web application using the generated artifacts of MAXAL some essential configuration settings are required. Appendix D and Appendix E deal with two possible alternatives for setting up the environment. While the *Standard Setup*, shown in Appendix D, allows developers with little experience in ASP.NET to setup the environment, the *Advanced Setup*, shown in Appendix E, requires some coding. The key benefit of the *Advanced Setup* is to add generated artifacts of MAXAL to existing projects.

Chapter 4

Evaluation and Future Work

This section provides an evaluation of the prototype implementation of MAXAL. The basic idea is to illustrate the opportunities MAXAL provides for web application developers. The introduced sample retrieves data from a database by specifying special search criteria. The database stores data about students, courses, lectures and their relationships. The main task when using the application is to retrieve data by editing the registration date, the title or number of a course or the lecturer who is responsible for a special course. After the user defines special search criteria and pushes a command button a search-operation is executed and the result is displayed within a HTML table.

For this purpose a Microsoft SQL Server 2005 Express Edition serves as data storage. A simple database schema is designed. The database consists of three entities – *Student*, *Course* and *Lecturer*. Each of these entities contains a minimal set of columns. The database is filled with a small set of data. The schema including its relationships is shown in the EER diagram shown in Figure 49.

As MAXAL provides a set of user-interface controls these components are used to realize the user-interface of the web application. All currently available implementations of user-interface components like the VerticalLayout, the HorizontalLayout, the command button, the label, the link button, the HTML table and the control for performing auto-completion are included within the sample application to evaluate its applicability and utilization.

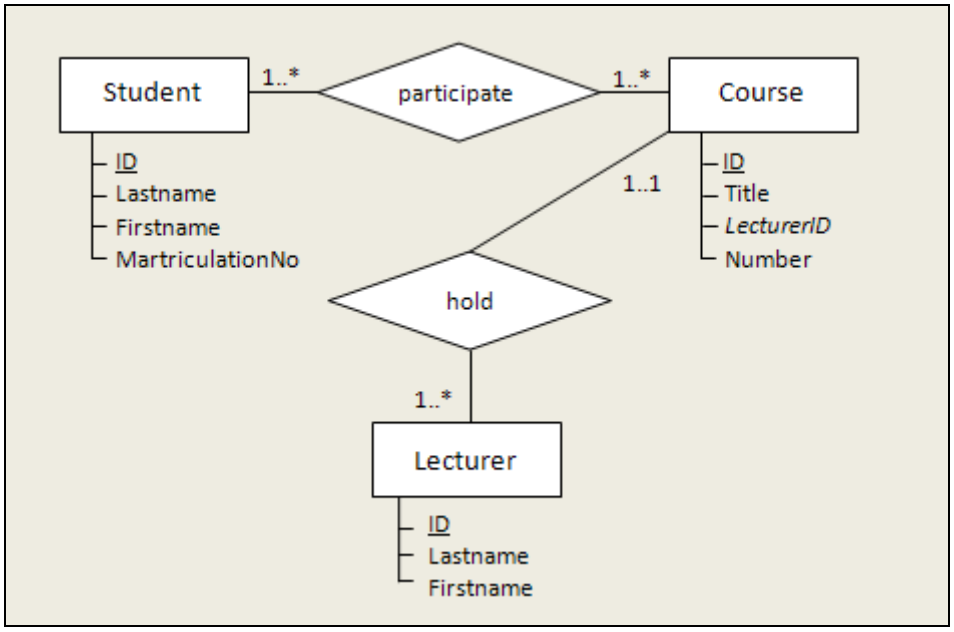


Figure 49: The Database Schema of the Sample Application

Figure 50 shows the model of the web application. It consists of a single web page named *Search*. Additionally 7 layouts and 11 user-interface components are used. Each instance of the class *Layout* is connected to the provided instance of the class *Page*. Each instance of the class *Control* is associated with a specific instance of the class *Layout*. Instances of the classes *Layout* and *Control* contain detailed information about their appearance, position on the screen and behavior within their attributes. All used instances of the class *Control* are mapped to specific available user-interface controls. Based on this model MAXAL generates three different artifacts to the specified destination directory. These artifacts are *Search.aspx*, *Search.aspx.cs* and *grdStudentData.cs*.

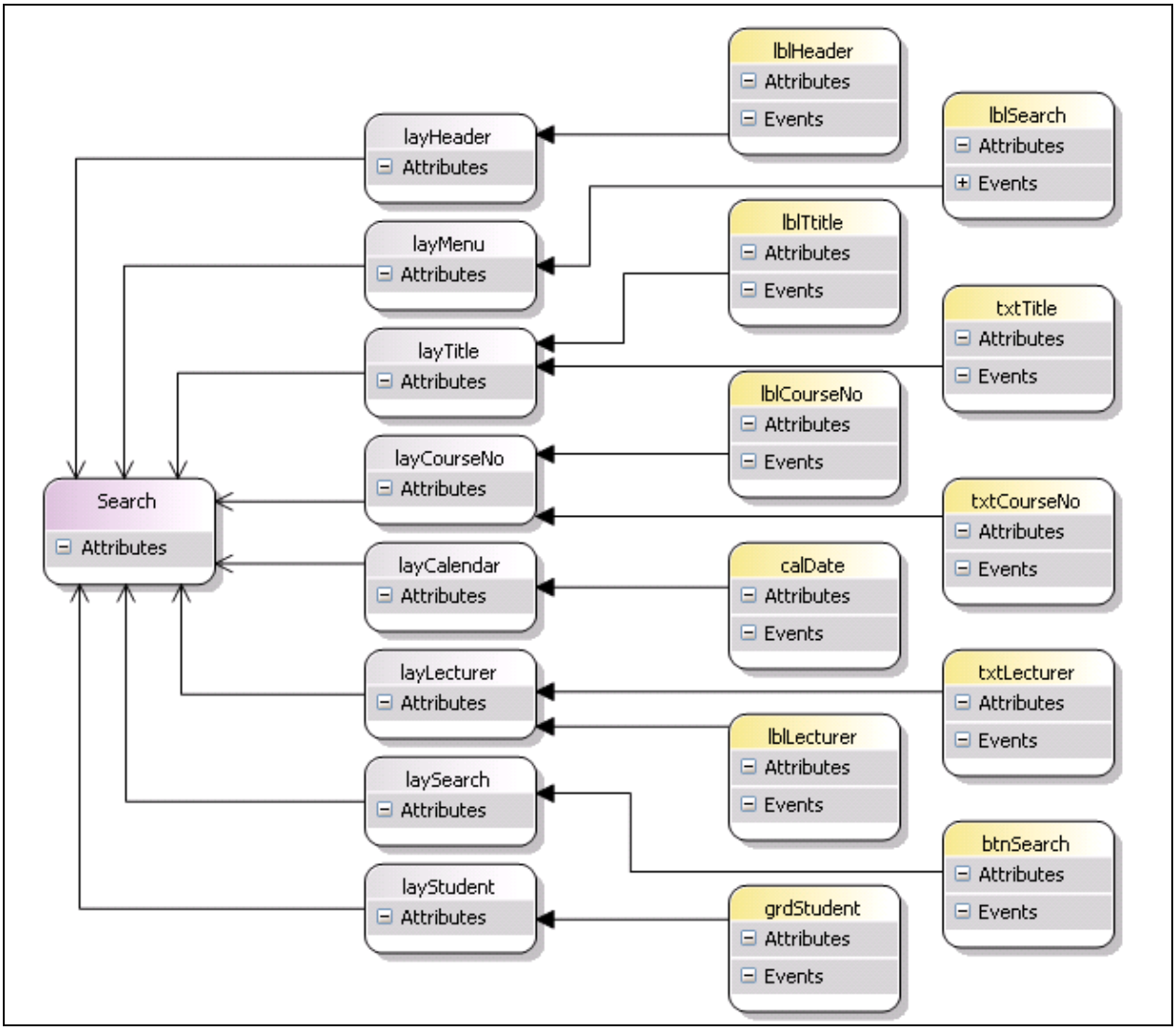


Figure 50: Model of the Sample Application[15]

After setting up the environment for a web application the generated artifacts have to be copied into appropriate directories within a web application. Details on the setup of the web application are described in detail in Appendix D and Appendix E. Before executing the application, generated stub methods may be implemented manually. MAXAL creates stub methods for controls' events and for binding data to specific controls. In the introduced web application two stub methods may be completed. Firstly the *click*-event has to be handled when a user pushes the *Search*-Button and secondly the connection for retrieving data from the database has to be completed manually. Generally the concrete implementation of both stub methods varies and can be adapted individually. The sample application for instance invokes a *Stored Procedure* to

[15] *lay*…Layout, *lbl*…Label, *txt*…Textbox, *grd*…GridView, *cal*…Calendar, *btn*…Button

perform the search-operation on the database. Furthermore the sample provides an ASP.NET Textbox with the *AutoComplete* functionality. A Web Service is implemented manually to receive available data. The URL of the Web Service and the invoked method are specified within the model of the web application shown in Figure 50. Figure 51 shows a screenshot of the web application.

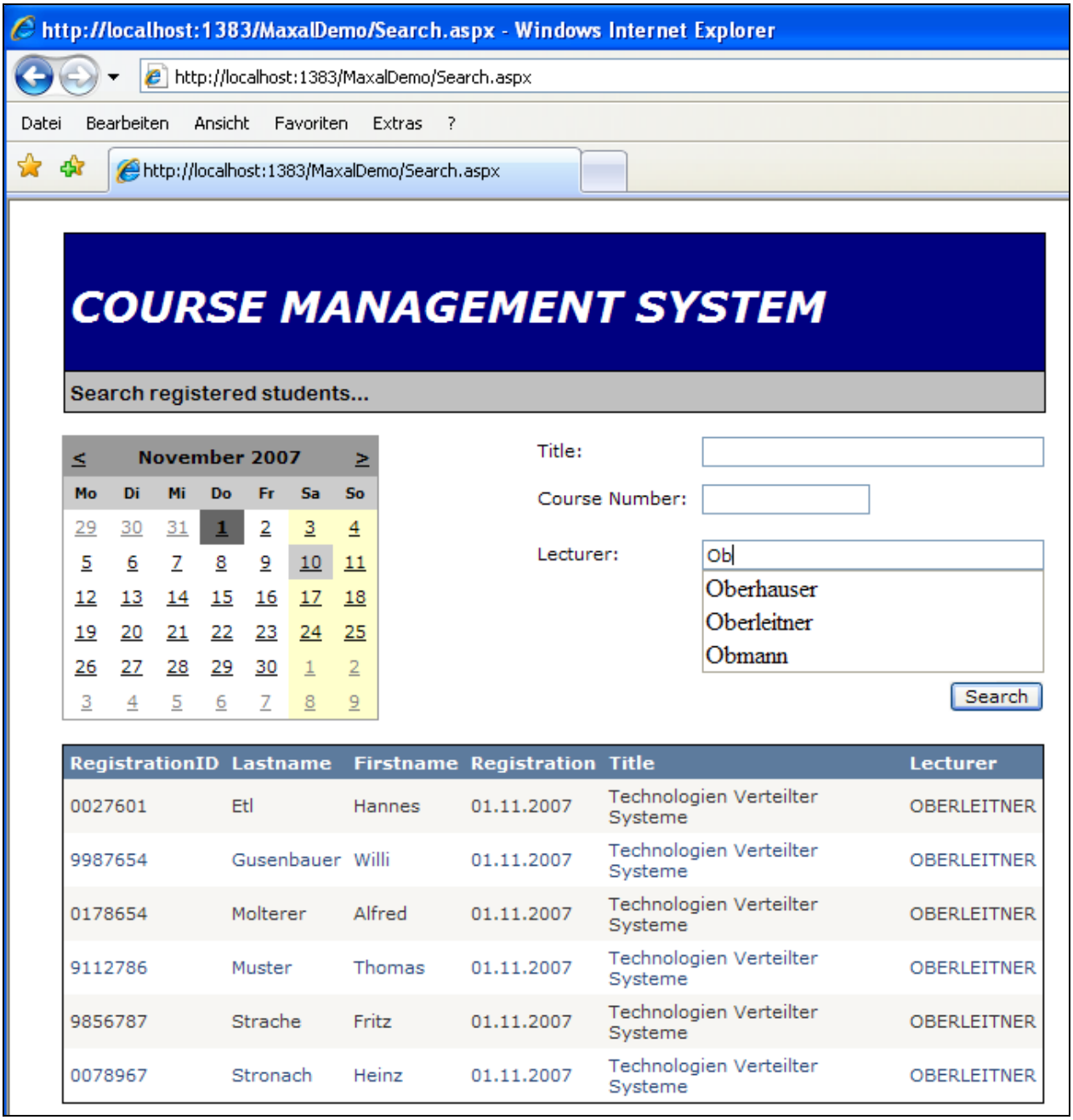


Figure 51: The Browser View of the Sample Application

Table 3 shows an evaluation of the code generation process using MAXAL. A major part of the application is generated by MAXAL. If the Web Service is not taken into account the percentage of generated lines increases to a high level. Detailed data are shown in Table 3.

File (Source Code)	Generated LOC[16]/ Total LOC	Description
Search.aspx	149/149	Web Form
Search.aspx.cs	147/160	Code-Behind File
grdStudentData.cs	25/84	Object for the ASP.NET ObjectDataSource. (ComplexDataBinding)
ParameterHelper.cs	0/62	Custom class to pass parameters retrieved from the user-interface for executing the SQL query.
Stored Procedure	0/11	SQL query to retrieve data from the database

Table 3: Evaluation of the Sample

Advantages

Designing web applications using MAXAL give developers a basic overview and understanding of the web application. The graphical model allows developers and laymen to communicate about a web application without technical knowledge. It allows developers to build a skeletal structure of the application. Small optical modifications within the interface might be easier and faster to adapt within the Visual Studio IDE.

As MAXAL is a prototype implementation it consists of only a small set of available user-interface components which prevents developers to build rich user-interfaces, but as MAXAL is extensible and provides developers mechanisms to include custom controls MAXAL provides an appropriate solution for this task.

Additionally a GUI (Graphical User-Interface) generator should provide rich user-interface components with pre-defined styles. It is for example not appropriate to force a developer to set all possible properties of the ASP.NET GridView control. For this reason MAXAL provides for example the ASP.NET GridView control with a predefined style. It might be advantageous to offer a set of various different styles.

[16] LOC…Lines Of Code

MAXAL allows developers to simplify and accelerate the backup process of a web application including all configuration files similar to Ruby[17]. As the web application can be generated at any time, invalid modifications have no problematic effects.

Disadvantages

Insufficient techniques for layouts make developing *pretty* user-interfaces difficult. As MAXAL currently does not support nested layouts and relative positioning the appearance of the user-interface of web application is problematic to a certain extent. Developers have to verify the output of the generator within the actual web application project.

Figure 50 illustrates that the model of a single web page reaches a complex level rapidly. Generally developing the whole web application with all its single web pages within one model becomes problematic, but MAXAL offers a solution for this problem. For this purpose MAXAL allows developers to build complex custom user-interface controls. For example having a control consisting of a configurable amount of textboxes and labels would reduce the complexity of the model to a high extent.

4.1 Future Work

Although the prototype implementation of MAXAL provides a powerful skeletal structure, there are still numerous subjects for further improvements. The list below provides desirable features which are not implemented yet.

- Relative Positioning of Layouts
  The prototype implementation of MAXAL supports absolute positioning of instances of the *HorizonalLayout* and *VerticalLayout*. Relative Positioning is not implemented yet. This feature would enhance the user-interface to a high degree.

- Comprehensive set of User-Interface Controls
  Developing professional user-interfaces requires the availability of a comprehensive set of user-interface controls. Currently MAXAL consists of only a small set of components

[17] http://www.ruby-lang.org/en/ (13th October 2007)

which allows only a limited development of web applications. Additionally it is advantageous to provide complex custom user-interface components with a set of predefined properties to reduce the complexity of the model of the web application and simplify the development. For instance, a custom control with a configurable amount of ASP.NET Textboxes and (or) ASP.NET Labels is useful in some cases. On the one hand this approach limits the flexibility and configurability of user-interface controls but on the other hand developers do not have to worry about specific concerns like style concerns.

- Complete Data-Binding Mechanisms

  The prototype implementation of MAXAL provides stub methods for binding data to user-interface controls. For this reason developers need to write a certain extent of source code manually to get a fully functional web application. A desirable feature is to enable binding data to user-interface controls through the model of the web application. The idea is to specify specific database parameters within the model and the MAXAL-Designer generates the source code.

- Caching and dynamic Loading of new Versions

  MAXAL currently provides a mechanism to load available user-interface controls into memory. A desirable feature is to cache loaded user-interface controls to enhance the performance of the generation process and to refresh the cache if a newer version of a specific user-interface control is deployed.

Chapter 5

Related Work

This section deals with alternative approaches for building and generating web applications, especially user-interfaces, using the model-driven design principle. As MAXAL all below mentioned approaches use source code generation facilities for the development process.

Andrade et al. [1] present a document-based prototype to generate web applications called *wView*. This tool is based on various XML-related techniques like XML documents, the XMI (XML Metadata Interchange) format, XML Schema, XSLT and the web publishing framework *Cocoon*[18]. Basically the code generator consists of a set of XML and XSLT documents which allow defining the content of web applications and the transformation rules to generate the web application. Firstly a XMI-based representation of the web application is created. For this purpose a UML (Unified Modeling Language) class diagram of the web application is designed and afterwards transformed to the XMI format. Based on this representation numerous transformation steps, which are based on XSLT and Cocoon, generate a deployable web application. Referring to the authors of this approach this tool is generally extensible by adding or modifying XML and XSLT documents. Furthermore the output is basically adaptable to other web technologies such as JSP, other publishing frameworks, storage types (XML databases) or presentation formats. On the other hand the authors encountered problems using XSLT.

Referring to MAXAL the prototype implementation for generating web applications introduced above differs in numerous aspects. Although the authors of the document-based approach argue that their implementation is generally extensible, interventions and extended features might cause fatal negative effects. As the authors use Cocoon for all transformation steps and Cocoon provides a pipeline mechanism for this purpose, an invalid modification like an additional feature might cause a crash of the complete transformation process. Further *wView* provides no specific environment to simplify the definition of models.

Nunes and Schwabe [24] introduce the *HyperDe* environment which can be used to build complete web applications based on Domain Specific Languages. A web application is designed

[18] http://cocoon.apache.org/ (14th September 2007)

using the SHDM (Semantic Hypermedia Design Method) method which consists of five steps – Requirements Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation. Furthermore SHDM provides a meta-model which allows developing concrete models of web applications. The tool which consists of meta-models, defined navigational models, as well as application instance data is completely stored as RDF (Resource Description Framework) data. Additionally *HyperDe* uses extended HTML templates to define the appearance of the front-end of the web application.

SHDM .NET, introduced by Ricci and Schwabe [26], is another model-driven approach for developing web applications based on the SHDM method. Analog to MAXAL this code generation tool is based on Visual Studio 2005 and the MS DSL Tools for developing meta models. Although *SHDM .NET* and MAXAL are based on the same technology *SHDM .NET* differs in some fundamental aspects. While MAXAL provides one DSL, *SHDM .NET* consists of a set of meta-models. *SHDM .NET* distinguishes between *Navigational context diagram meta-models*, *abstract* and *concrete interface diagram meta-models*. Secondly *SHDM .NET* focuses explicitly on the navigational context of the web application. For this reason *SHDM .NET* generates classes. As MAXAL *SHDM .NET* uses a late binding option to allow assigning concrete user-interface components like Buttons, Labels, etc. to abstract instances of front-end components. Furthermore SHDM .NET generates ASP.NET based web applications using VB (Visual Basic).

As MAXAL the *HyperDe* environment is a model-driven approach using DSLs. Within *HyperDe* data are stored within RDF databases. MAXAL basically generates user-interfaces but provides method stubs to enable developers to add data sources. At the same time MAXAL does not restrict the type of the data source used. Developers might bind data for example from a relational database (MS SQL, MS Access, etc.) or from an XML database. Developers using *HyperDe* are forced to use a RDF database. Addressing the front-end, both approaches use similar techniques to generate user-interfaces. While MAXAL uses Microsoft-specific Text Templates, *HyperDe* uses extended HTML templates. Both techniques provide similar syntactical rules.

Milosavljevid et al. [22] introduce a simple approach for generating database-oriented web applications. This tool uses Java and Java-related techniques like JavaBeans, Servlets and JSP (Java Server Pages) to build web applications. Their approach assumes that the web application retrieves its data from a database. The first step is to map the database schema to an XML

document. Based on the XML document appropriate classes, JavaBeans, Servlets and JSP pages are generated using XSLT. The generation of the front-end is based on a set of predefined web pages. Basically the concept of this tool is similar to the code generator introduced in [1]. MAXAL generally follows a different approach. As MAXAL provides a visual environment for specifying models and their relationships, the development of web applications is accessible on a more conceptional layer. The XML-based representation of web applications requires expert knowledge. Furthermore using the approach in [22] designing user-interfaces is limited to a set of predefined templates.

Molina [23] presents the ONME (OlivaNova Model Execution System) for generating user-interfaces for various platforms. ONME consists of a tool suite which contains a component for defining specifications of a software system within a visual editor. Furthermore it provides facilities to define abstract user-interface specifications and mappings from abstract user-interface specifications to concrete implementations not only for web applications but also for any device. Unlike ONME, MAXAL is designed to generate ASP.NET based web applications and MAXAL offers developers facilities to bind data to user-interface components.

WARP (Web Application Rapid Prototyping) [3] offers a set of tools for building web applications using model-driven techniques. This approach is based on HDM2000, which is a notation for the specification of the structure navigation and presentation semantics. *WARP* supports the whole development cycle beginning with the analysis of requirements until the deployment of the complete system. Designing the presentation layer is done within the tool *WPD* (*Warp Presentation Designer*). Furthermore *WARP* contains tools like *WFeeder* which allows managing the content of the prototype of the web application, *WEngine* and *Generator* to publish the generated artifacts of the generation process. Data exchange between those tools is realized with XML-based files. The current version of WARP is implemented using MS Visual Studio and the MS SQL DBMS (Database Management System). The online environment is realized with VB.NET (Visual Basic) and JavaScript.

Compared to MAXAL the generation tool of Bochicchio and Fiore [3] addresses the whole development cycle. While MAXAL supports the generation of especially front-end artifacts, *WARP* considers for example aspects like requirements and deployment. *WARP* generally supports all steps within the development process. On the one hand *WARP* offers a wide range of tools with rich capabilities but on the other hand a comprehensive knowledge of all included

tools is required to build web applications. Both MAXAL and *WARP* are based on the .NET framework and ASP.NET.

Another CASE tool for modeling and generating web applications is introduced in [6]. This tool allows the specification and deployment of web applications and web services. The code generator is implemented using the CASE tool WebRatio[19] which enables developers to visually specifying web interfaces and web service interfaces. WebRatio is based on WebML [12, 5] which extends UML with web application facilities. The generator deploys the specified model in the J2EE platform. WebRatio provides rich capabilities like for example data design, hypertext design, data mapping, presentation design, code generation. Essential tasks within WebRatio like conceptual modeling and code generation are discussed [4]. The code generation process allows generating running web applications for Java2EE, Struts and the .NET platform. Furthermore WebRatio checks semantic correctness, automatic production of project documentation, project version management and database direct and reverse engineering. WebRatio additionally offers a plug-in architecture which allows developers to extend the tool by XML descriptors and XSL rules.

Both WebRatio and MAXAL provide rich capabilities such as extensibility. While WebRatio therefore uses XML documents MAXAL offers developers to write their own features using C#. While controls within MAXAL are generally AJAX-enabled, WebRatio has not integrated this feature as default. WebRatio provides the plug-in *WebRatio AJAX Extension* for building RIAs (Rich Internet Application).

WAgen is a web application generator toolkit introduced in [13] which allows the development of complete and ready-to-use internet applications. Similar to the above mentioned tool *WebRatio WAgen* distinguishes between a *Content Model*, a *Composition* & *Navigational Model* and a *Presentation Model*. These terms are explained in [4]. *WAgen* introduces one additional model – the *Operational Model* [14]. The idea of this model is to express complex operations that access or modify the content of web applications. Referring to the authors of *WAgen* the J2EE platform and ZOPE are two possible implementation environments. *WAgen* uses three main generator components which use XML definitions as input. The *Presentation Model* is used to determine the front-end of web applications. This model uses CSS to allow positioning of elements within the user-interface that are specified within the *Navigational Model*.

[19] http://www.webratio.com (20th September 2007)

Compared to MAXAL the above mentioned approach differs in some cases. *WAgen* provides no environment for the definition of its models. Although the authors mention that the usage of CASE tools supports the development of models and eliminates these limitations, developing web applications still remains as task for experts. Furthermore *WAgen* is not extensible. Extensions or modifications require the adaption of the used code generators.

In summary, a multitude of generation tools are currently available with more or less features. Nearly all approaches are based on XML and XML-related techniques. Although a certain portion of presented tools are extensible, MAXAL is the only prototype that offers developers to build their own features using a popular imperative programming language like C#. Furthermore AJAX is a main feature within MAXAL and insofar all generated web applications are AJAX-enabled as default. This characteristic is unique and not supported by any other introduced generation tool.

Maximilien et al. [20] introduce the *Swashup* platform which is a programming model that facilitates and accelerates the development and deployment of mashups[20] of diverse services. It is implemented using Ruby and the RoR (Ruby on Rails) framework. Additionally the platform provides a DSL to unify the most common service models.

Similar to MAXAL *Swashup* is based on a DSL for the development of web applications. While the *Swashup* platform typically retrieves its data from various services, MAXAL does not restrict its data sources. Data might be stored within relational databases, web services or any other store. Although *Swashup* focuses on services it does not offer advantageous capabilities compared to MAXAL. Furthermore *Swashup* enables developers to create concrete models of the DSL only text-based. MAXAL however provides a visual development environment for this task. Another advantage of MAXAL is that the front end is completely generated. Using *Swashup* the generation of web pages is limited. For this reason views need to be customized by the user. Both approaches MAXAL and *Swashup* enable the development of AJAX-enabled user-interfaces.

[20] A *mashup* is a web application that aggregates multiple services to achieve a new purpose. [20]

Chapter 6

Summary and Conclusion

This thesis introduced a model-driven approach for building AJAX-enabled web applications. The recent version of the prototype implementation, named MAXAL (Model-Driven Ajax Application Language), allows developing AJAX-enabled web applications by specifying models and their relationships within a visual development environment.

Both AJAX and Model-Driven Development are promising techniques for building rich internet applications. On the one hand this thesis has highlighted the background of AJAX, its key components, capabilities, advantages and disadvantages. Furthermore this thesis introduced basic implementation details for developing AJAX-enabled web applications generally as well as approved AJAX frameworks like for example MS ASP.NET AJAX. Additionally this thesis focused on Model-Driven Development. As current existing techniques for building professional web applications through 3GLs are generally error-prone, cost intensive and time-consuming, model-driven development provides promising capabilities to handle these problematic aspects [15]. For this reason this thesis discussed main ideas of the model-driven approach with its benefits and shortcomings.

The introduced prototype implementation of MAXAL offers developers a set of components to design concrete models of web applications *Out-Of-The-Box* with less effort. Furthermore MAXAL enables the generation of various artifacts of web applications. The tool itself was developed within the .NET platform. As the core of MAXAL is a DSL (Domain Specific Language), a meta-model for building internet applications platform-independently was presented. Its structure, elements, characteristics and capabilities were highlighted in detail. The DSL was designed using the MS Visual Studio DSL Tools. Furthermore a set of approved ASP.NET based user-interface controls is provided to build professional web applications rapidly. The integrated, template-based source code generator allows the generation and deployment of numerous artifacts like for instance web forms.

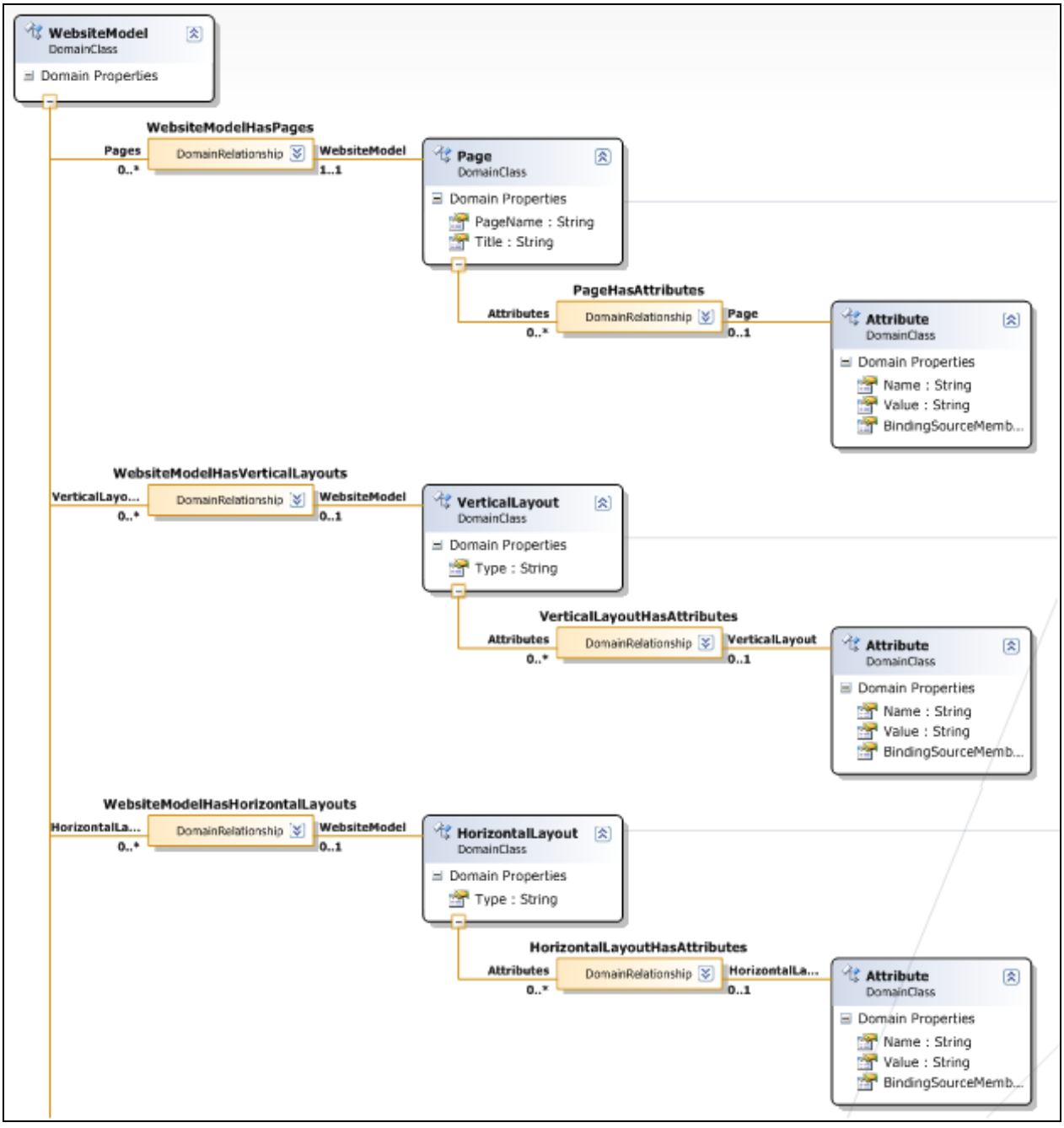
The main features of the introduced approach are extensibility through its plug-in architecture, a set of AJAX-enabled user-interface components, the development of custom user-

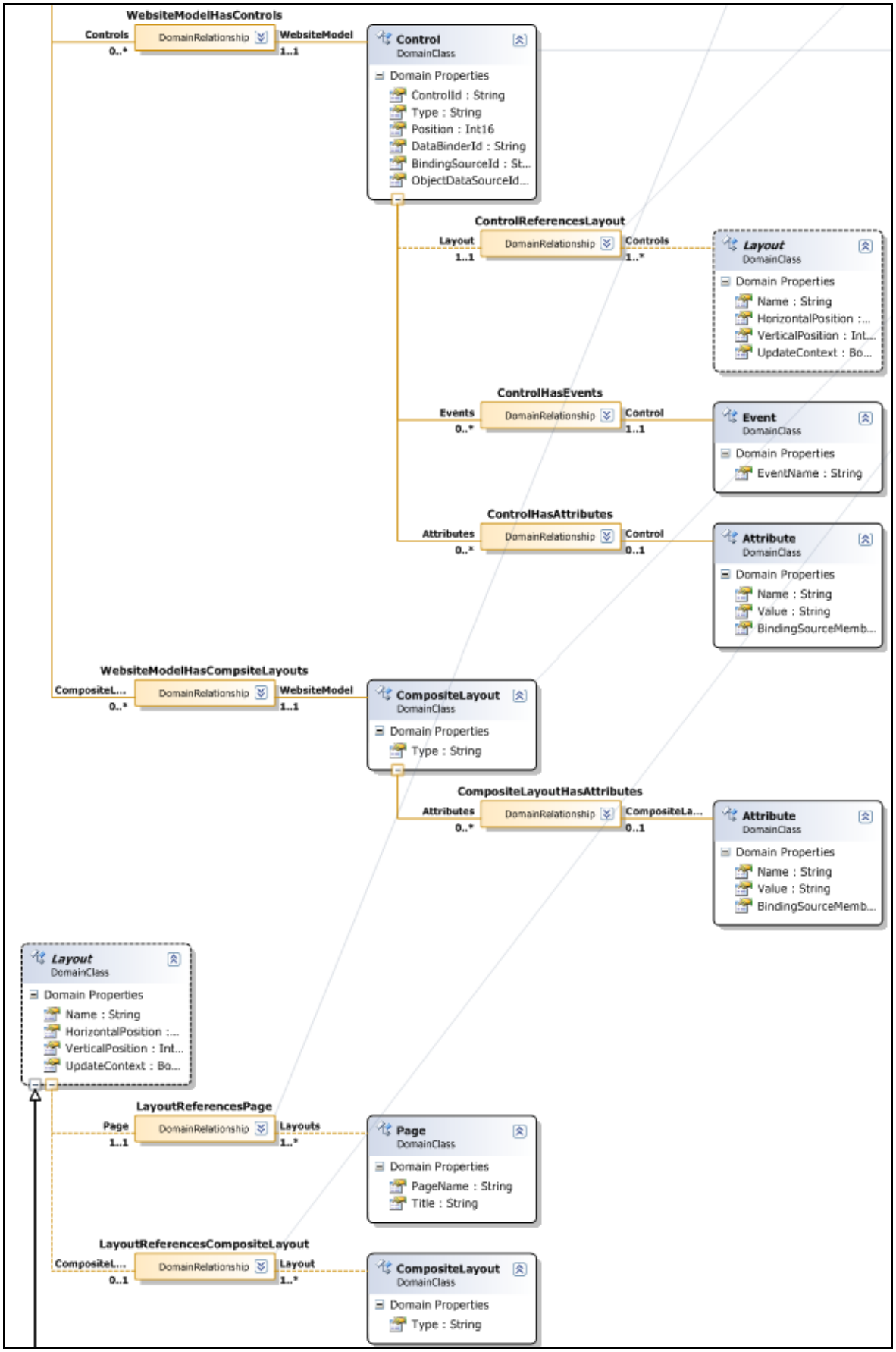
interface controls, the visual development environment and mechanisms for binding data to front-end components. The plug-in architecture allows modifying and extending the recent version of MAXAL without accessing its source code. Developers might use existing front-end components or develop their own controls. Provided data binding mechanisms offers developers rich capabilities to bind data sources to user-interface components with a reasonable effort.

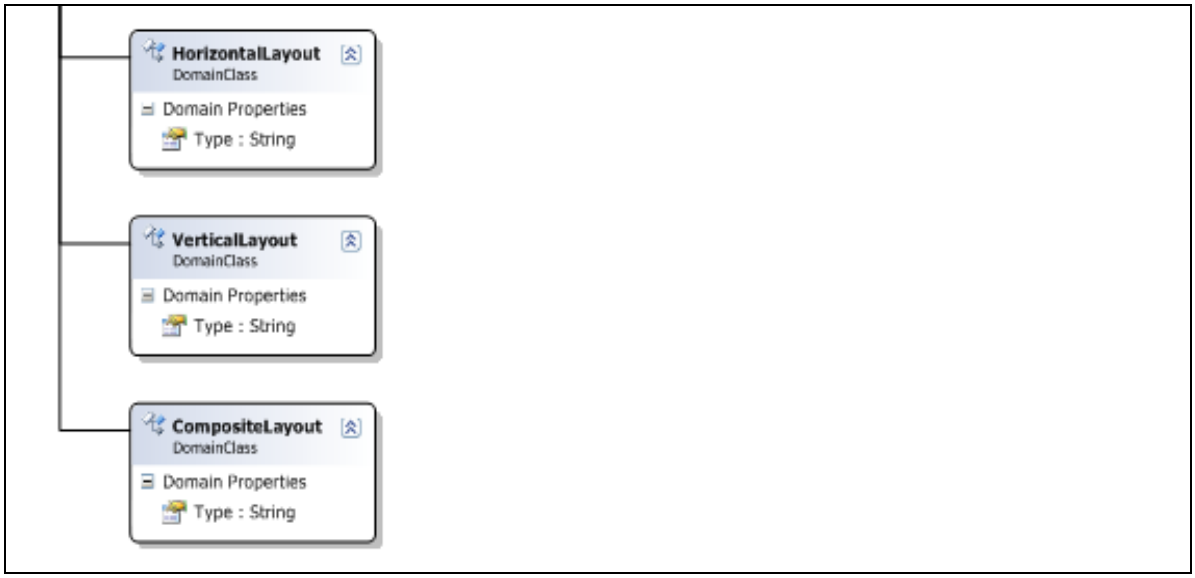
Further studies in evaluating the utilization and applicability of the prototype implementation are required. An empirical analysis might illustrate the quality of the generation process. Furthermore implementations of the meta-model and the web design environment on different platforms are indispensible. The quantity of available user-interface components and automatically generated connections to data sources, e.g. to a relational database, through the model are subject for further research.

Appendix

Appendix A




WebsiteModelHasControls
Controls
0..*
DomainRelationship
WebsiteModel
1..1
Control
DomainClass
Domain Properties
ControlId : String
Type : String
Position : Int16
DataBinderId : String
BindingSourceId : St...
ObjectDataSourceId...
ControlReferencesLayout
Layout
1..1
DomainRelationship
Controls
1..*
Layout
DomainClass
Domain Properties
Name : String
HorizontalPosition :...
VerticalPosition : Int...
UpdateContext : Bo...
ControlHasEvents
Events
0..*
DomainRelationship
Control
1..1
Event
DomainClass
Domain Properties
EventName : String
ControlHasAttributes
Attributes
0..*
DomainRelationship
Control
0..1
Attribute
DomainClass
Domain Properties
Name : String
Value : String
BindingSourceMemb...
WebsiteModelHasCompsiteLayouts
CompositeL...
0..*
DomainRelationship
WebsiteModel
1..1
CompositeLayout
DomainClass
Domain Properties
Type : String
CompositeLayoutHasAttributes
Attributes
0..*
DomainRelationship
CompositeLa...
0..1
Attribute
DomainClass
Domain Properties
Name : String
Value : String
BindingSourceMemb...
Layout
DomainClass
Domain Properties
Name : String
HorizontalPosition :...
VerticalPosition : Int...
UpdateContext : Bo...
LayoutReferencesPage
Page
1..1
DomainRelationship
Layouts
1..*
Page
DomainClass
Domain Properties
PageName : String
Title : String
LayoutReferencesCompositeLayout
CompositeL...
0..1
DomainRelationship
Layout
1..*
CompositeLayout
DomainClass
Domain Properties
Type : String


HorizontalLayout
DomainClass
Domain Properties
Type : String
VerticalLayout
DomainClass
Domain Properties
Type : String
CompositeLayout
DomainClass
Domain Properties
Type : String

Appendix B

```
using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Diagnostics;
using System.Drawing.Design;
using System.Windows.Forms;
using System.Windows.Forms.Design;
using Microsoft.VisualStudio.Modeling.Design;

namespace WebDesigner.Maxal
{
 public class PropertyControlTypeNameEditor : UITypeEditor
 {
  private IWindowsFormsEditorService FormEditorService;

  public override UITypeEditorEditStyle GetEditStyle(ITypeDescriptorContext context)
  {
   if (context == null)
   {
    return base.GetEditStyle(context);
   }
   return UITypeEditorEditStyle.DropDown;
  }

  protected void List_Click(object pSender, EventArgs pArgs)
  {
   if (FormEditorService != null)
   {
    FormEditorService.CloseDropDown();
   }
  }

  public override object EditValue(ITypeDescriptorContext context,
                                   IServiceProvider provider, object value)
  {
   if ((context == null) || (provider == null) || (context.PropertyDescriptor == null))
   {
    return base.EditValue(context, provider, value);
   }

   FormEditorService =
  (IWindowsFormsEditorService)provider.GetService(typeof(IWindowsFormsEditorService));
   ListBox lbxAvailableControlType = new ListBox();
   lbxAvailableControlType.Click += new EventHandler(List_Click);
   ControlFactory ControlFactory = new ControlFactory();
   lbxAvailableControlType.Items.AddRange(ControlFactory.GetControlTypes());
   FormEditorService.DropDownControl(lbxAvailableControlType);
   return lbxAvailableControlType.SelectedItem;
  }

  private DialogResult ShowForm(IServiceProvider provider, Form form)
  {
   IUIService service = (IUIService)provider.GetService(typeof(IUIService));
   if (service != null)
   {
     return service.ShowDialog(form);
   }
    return form.ShowDialog();
  }
 }
}
```

Appendix C

```
<?xml version="1.0"?>
<configuration>
<configSections>
 <sectionGroup name="system.web.extensions"
  type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
  System.Web.Extensions, Version=1.0.61025.0”>
 <sectionGroup name="scripting" type="System.Web.Configuration.ScriptingSectionGroup,
  System.Web.Extensions, Version=1.0.61025.0, Culture=neutral">
  <section name="scriptResourceHandler"
   type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
   System.Web.Extensions, Version=1.0.61025.0" allowDefinition="MachineToApplication"/>
   <sectionGroup name="webServices"
   type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
   System.Web.Extensions, Version=1.0.61025.0, Culture=neutral">
  <section name="jsonSerialization"
   type="System.Web.Configuration.ScriptingJsonSerializationSection,
   System.Web.Extensions, Version=1.0.61025.0" allowDefinition="Everywhere"/>
  <section name="profileService"
   type="System.Web.Configuration.ScriptingProfileServiceSection,
   System.Web.Extensions, Version=1.0.61025.0, Culture=neutral"
   allowDefinition="MachineToApplication"/>
  <section name="authenticationService"
   type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
   System.Web.Extensions, Version=1.0.61025.0" allowDefinition="MachineToApplication"/>
 </sectionGroup>
 </sectionGroup>
 </sectionGroup>
 </configSections>
<appSettings/>
<connectionStrings/>
<system.web>
 <pages>
  <controls>
   <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions,
    Version=1.0.61025.0"/>
   <add namespace="AjaxControlToolkit" assembly="AjaxControlToolkit"
    tagPrefix="ajaxToolkit"/>
  </controls>
 </pages>
<compilation debug="true">
 <assemblies>
  <add assembly="System.Web.Extensions, Version=1.0.61025.0"/>
  <add assembly="System.Design, Version=2.0.0.0"/>
  <add assembly="System.Windows.Forms, Version=2.0.0.0"/></assemblies>
 </compilation>
<httpHandlers>
 <remove verb="*" path="*.asmx"/>
 <add verb="*" path="*.asmx" validate="false"
  type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
  Version=1.0.61025.0"/>
 <add verb="*" path="*_AppService.axd" validate="false"
  type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
  Version=1.0.61025.0"/>
 <add verb="GET,HEAD" path="ScriptResource.axd"
  type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
  Version=1.0.61025.0" validate="false"/>
</httpHandlers>
```

```
<httpModules>
 <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
  System.Web.Extensions, Version=1.0.61025.0 "/>
</httpModules>
<authentication mode="Windows"/>
</system.web>
<system.webServer>
<validation validateIntegratedModeConfiguration="false"/>
<modules>
<add name="ScriptModule" preCondition="integratedMode"
 type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=1.0.61025.0"/>
</modules>
<handlers>
 <remove name="WebServiceHandlerFactory-Integrated"/>
 <add name="ScriptHandlerFactory" verb="*" path="*.asmx" preCondition="integratedMode"
  type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
  Version=1.0.61025.0″/>
 <add name="ScriptHandlerFactoryAppServices" verb="*" path="*_AppService.axd"
  preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
  System.Web.Extensions, Version=1.0.61025.0"/>
 <add name="ScriptResource" preCondition="integratedMode" verb="GET,HEAD"
  path="ScriptResource.axd" type="System.Web.Handlers.ScriptResourceHandler,
  System.Web.Extensions, Version=1.0.61025.0"/>
</handlers>
</system.webServer>
</configuration>
```

Appendix D (Standard Setup)

a) Download & Installation of *ASP.NET 2.0 AJAX Extensions v1.0*[21]

This download installs Microsoft's framework for developing and running AJAX-enabled web applications with either server-centric or client-centric development models.

b) Download & Installation of *ASP.NET AJAX Control Toolkit*[21]

The ASP.NET AJAX Control Toolkit is a shared-source community project. It contains a collection of samples and components to simplify the work with AJAX-enabled controls and extenders. Additionally this toolkit provides a Software Development Toolkit (SDK) to develop custom ASP.NET AJAX controls. As MAXAL generates artifacts which are based on the *ASP.NET AJAX Control Toolkit* this component may be installed.

c) Create a new *Web Site* using the Visual Studio template *ASP.NET AJAX Control Project*

After installing the *ASP.NET AJAX Control Toolkit* Visual Studio offers a specific project template called *ASP.NET AJAX Control Project* to develop AJAX-enabled web

[21] http://asp.net/ajax/downloads (13th August 2007)

applications with controls of the *ASP.NET AJAX Control Toolkit (*Figure 52*)*. Using this predefined template all required entries within the XML-based configuration file *Web.Config* are generated automatically. Developers who decide to setup the environment following the *Advanced Setup* have to add those entries manually.

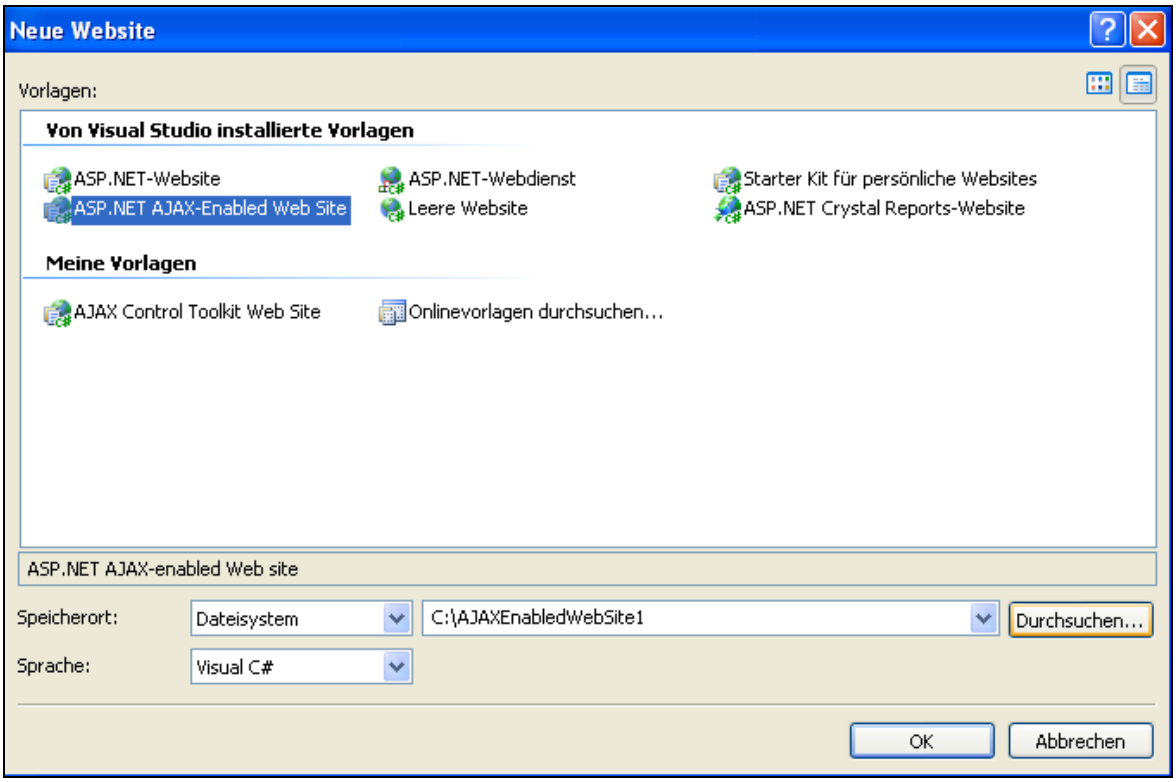


Figure 52: ASP.NET AJAX Control Toolkit Template

d) Add references to the project

The MAXAL-Designer generates controls with specific data binding mechanisms if controls provide a support. For this reason the following two assemblies have to be added to the project.

- wwDataBinder.dll[22]
- System.Windows.Forms.dll (built-in assembly of the .NET Framework 2.0)

e) Copy generated artifacts into the project

The last step before executing the web application is to copy all generated artifacts into the appropriate folder(s) of the web application.

[22] http://msdn.microsoft.com/msdnmag/issues/06/12/ExtendASPNET/default.aspx?loc=en (13th August 2007)

APPENDIX E (Advanced Setup)

a) Create a new Project *Web Site* or use an existing Project

The first step is creating a new project using the built-in template *Web Site*…

Alternatively an existing project can be used.

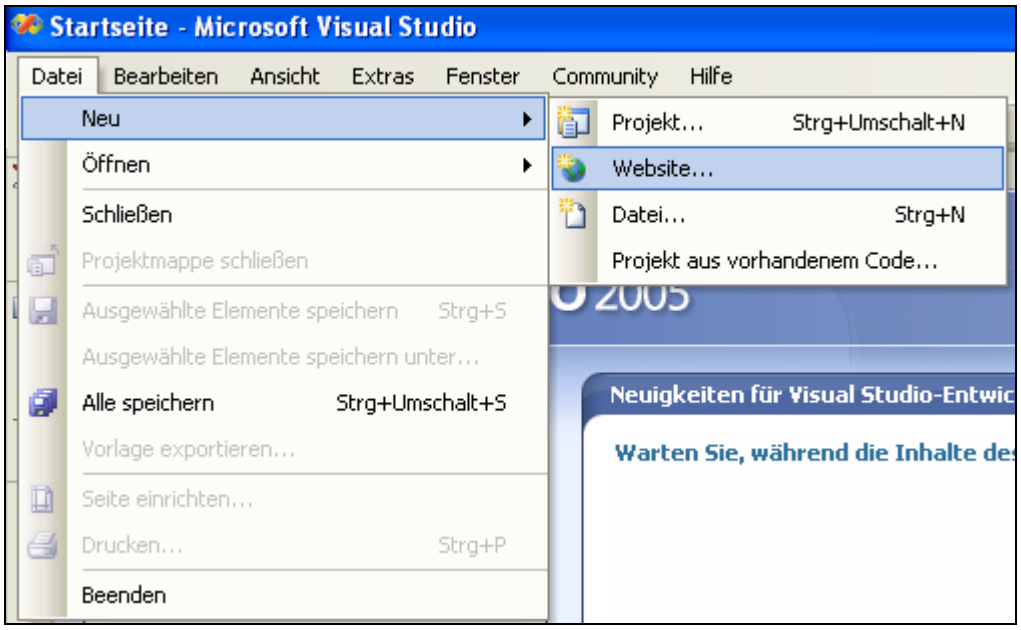


Figure 53: Advanced Setup

b) Add references to the project

The MAXAL-Designer generates AJAX-enabled controls with data binding mechanisms. For this reason the following assemblies have to be added to the project.

- wwDataBinder.dll[23]
- System.Windows.Forms.dll (built-in assembly of the .NET Framework 2.0)
- System.Web.Extensions.dll
- AjaxControlToolkit.dll

[23] http://msdn.microsoft.com/msdnmag/issues/06/12/ExtendASPNET/default.aspx?loc=en (13th August 2007)

c) Adapt Web.Config

The above included references require some additional configuration settings. For this reason the XML-based file *Web.Config* which is generated by Microsoft Visual Studio 2005 IDE automatically has to be modified. Appendix C shows a ready-to use *Web.Config*. All required sections within the *Web.Config* are highlighted.

d) Copy generated artifacts into the project

The last step before executing the web application is to copy all generated artifacts into the appropriate folder(s) of the web application.

References

[1] **Andrea R. de Andrade, Ethan V. Munson et al.** *A Document based Approach to the Generation of Web Applications*, Proceedings of the 2004 ACM symposium on Document Engineering, Pages: 45 – 47, ACM, Oct. 2004, ACM

[2] **Krishnakumar Balasubramanian, Aniruddha Gokhale et al.** *Developing Applications Using Model-Driven Design Environments*, Computer, Volume 39, Issue 2, Feb. 2006, Pages: 33 – 40, IEEE

[3] **Mario Bochicchio, Nicola Fiore.** *WARP: Web Application Rapid Prototyping*, Proceedings of the 2004 ACM symposium on Applied computing, Pages: 1670 – 1676, Mar. 2004, ACM

[4] **Allessandro Bozzon, Sara Comai et al.** *Conceptual Modeling and Code Generation for Rich Internet Applications*, Proceedings of the 6th international conference on Web engineering, Pages: 353 - 360, July 2006, ACM

[5] **Marco Brambilla.** *Generation of WebML web application models from business process specifications*, Proceedings of the 6th international conference on Web engineering, Pages: 85 – 86, Jul. 2006, ACM

[6] **Marco Brambilla, Stefano Ceri et al.** *A CASE tool for modeling and automatically generating web service-enabled applications*, International Journal of Web Engineering and Technology 2006 - Vol. 2, No.4, Pages: 354 – 372, (http://www.webml.org/)

[7] **R.Ian Bull.** *Integrating dynamic Views Using Model Driven Development*, IBM Centre for Advanced Studies Conference, Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, Article No. 17, Oct. 2006, ACM

[8] **Steve Cook, Gareth Jones, et al.** *Domain Specific Development with Visual Studio DSL Tools*, June 2007, Addison-Wesley

[9] **CSS Specification.** http://www.w3.org/TR/REC-CSS1 (June 28th, 2007)

[10] **DOM Specification.** http://www.w3.org/TR/REC-DOM-Level-1 (June 28th, 2007)

[11] **Davis S. Frankel.** *Model Driven Architecture – Applying MDA to Enterprise Computing*, Jan. 2003, Wiley

[12] **S. Comai, P. Fraternali.** *A Semantic Model for Specifying Data-Intensive Web Applications Using WebML*, Semantic Web Workshop, Stanford, USA, Jul. 2001, (http://www.webml.org/)

[13] **Mihály Jakob, Holger Schwarz et al.** *Modeling and generating application logic for data- intensive web applications*, Proceedings of the 6th international conference on Web engineering, Pages: 77 – 84, Jul. 2006, ACM

[14] **Mihály Jakob, Holger Schwarz et al.** *Towards an operation model for generated web applications*, Workshop proceedings of the sixth international conference on Web engineering, Article No. 7, Jul. 2006, ACM

[15] **Nora Koch.** *Transformation Techniques in the Model-Driven Development Process of UWE* ACM International Conference Proceeding Series; Vol. 155, Workshop proceedings of the 6th international conference on Web engineering, Article No. 3, 2006, ACM

[16] **Nora Koch, Gefei Zhang et al.** *Model Transformation from Requirements to Web System Design*, ACM International Conference Proceeding Series, Proceedings of the 6th international conference on Web engineering, Pages: 281 – 288, 2006, ACM

[17] **Guido Krüger.** *Handbuch der Java Programmierung*, 4.Auflage, 2006, Addison-Wesley

[18] **Ivan Kurtev, Jeadn Bezivin et al.** *Model-based DSL Frameworks*, Conference on Object Oriented Programming Systems Languages and Applications, Pages: 602 – 616, Oct. 2006, ACM

[19] **Matthew MacDonald, Mario Szpuszta.** *Pro ASP.NET 2.0 in C# 2005 - Create next-generation web applications with the latest version of Microsoft's revolutionary technology*, Sept. 2005, Apress

[20] **E. Michael Maximilien et al.** *A Domain-Specific Language for Web APIs and Services Mashups*, ICSOC 2007, Springer

[21] **Ali Mesbah and Arie van Deursen.** *An Architectural Style for AJAX*, Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture, Jan. 2007, Page(s): 9 – 9, IEEE

[22] **Branko Milosavljevid, Milan Vidakovid et al.** *Automatic Code Generation for Database-oriented Web Applications*, ACM International Conference Proceeding Series; Vol. 25, Pages: 59 – 64, Jun. 2002, ACM

[23] **Pedro J. Molina.** *User Interface Generation with OlivaNova Model Execution System*, Proceedings of the 9th international conference on Intelligent user interfaces, Pages: 358 – 359, Jan. 2004, ACM

[24] **Dementrius Arraes Nunes, Daniel Schwabe**. *Rapid Prototyping of Web Applications Combining Domain Specific Languages and Model Driven Design*, Proceedings of the 6th international conference on Web engineering ICWE '06, Jul. 2006, Pages: 153 – 160, ACM

[25] **Linda Dailey Paulson.** *Building Rich Web Applications with AJAX*, Computer, Volume 38, Issue 10, Oct. 2005, Pages: 14 – 17, IEEE

[26] **Luiz A. Ricci and Daniel Schwabe.** *An Authoring Environment for Mode-Driven Web Applications*, ACM International Conference Proceeding Series; Vol. 192, Proceedings of the 12th Brazilian symposium on Multimedia and the web, Pages: 11 – 19, Oct. 2006, ACM

[27] **James F. Ryan and Blair L. Reid.** *Usable Encryption Enabled by AJAX*, Proceedings of the International conference on Networking and Services ICN'06, Page(s):116 – 116, IEEE

[28] **Jan Schulz-Hofen, Silvan Golega.** *Generating Web Applications from Process Models*, ACM International Conference Proceeding Series; Vol. 155, Workshop proceedings of the sixth international conference on Web engineering, Article No. 6, Jul. 2006, ACM

[29] **Cynthia C. Shelly, Georg Young.** *Accessibility for Simple to Moderate-Complexity DHTML Web Sites*, ACM International Conference Proceeding Series; Vol. 225, Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A), Pages: 65 – 73, May 2007, ACM

[30] **Keith Smith.** *Simplifying AJAX-Style Web Development*, Computer, Volume 39, Issue 5, May 2006, Pages: 98 – 101, IEEE

[31] **Michael Sonntag.** *AJAX Security in Groupware*, Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications EUROMICRO '06, Aug. 2006, Pages: 472 – 479, IEEE

[32] **John Stamey and Trent Richardson.** *Middleware Development with AJAX*, Journal of Computing Sciences in Colleges, Volume 22, Issue 2, Dec. 2006, Pages: 281 -287, ACM

[33] **Ralph Steyer and Joachim Fuchs.** *AJAX mit ASP.NET und ATLAS -Der Einstieg in die hochperformante Webentwicklung*, Sept. 2006, Addison-Wesley

[34] **XML Specification.** http://www.w3.org/TR/REC-xml (June 28th, 2007)