Internetanwendungstechnik – IAT

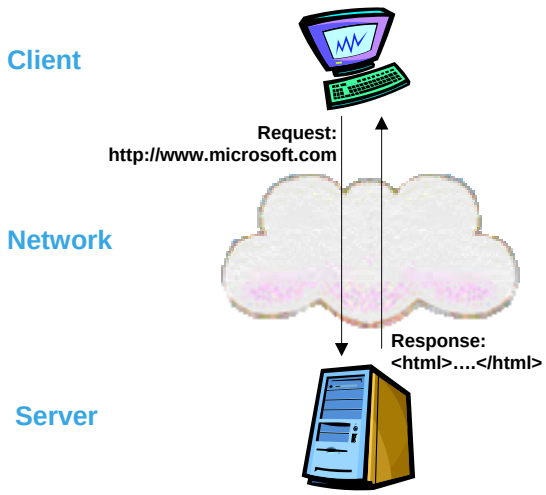
Web Applications with ASP.NET

Helge Parzyjegla

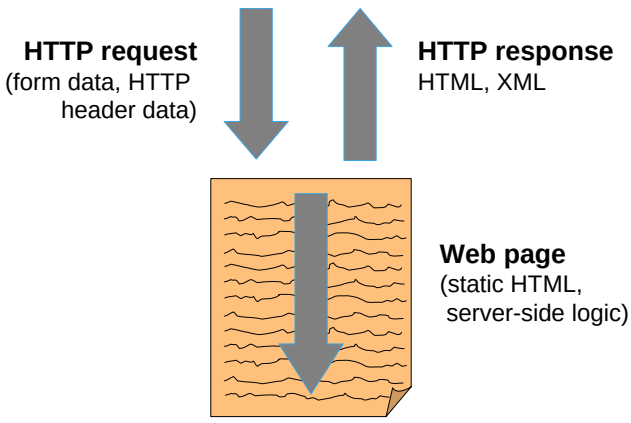
Kommunikations- und Betriebssysteme (KBS)
Fakultät IV – Elektrotechnik und Informatik
TU Berlin

Outline

Web Architecture



Flow of Action



Web Development Technologies

- Client-side technologies
  - HTML, DHTML, JavaScript
  - Applets, ActiveX® Controls
- Server-side technologies
  - CGI, PHP, Servlets
  - JSP, ASP
- ASP.NET is the next generation of ASP

Common Problems in Web Development

- Coding overhead
- Code readability
  - too complex
  - code and UI intermingled
- Reuse is difficult
- Supporting many types of browsers is difficult
- Session state scalability and availability
- Limited support for caching, tracing, debugging, etc.
- Performance and safety limitations of script

Goals of ASP.NET

- Keep the good parts of ASP and improve the rest
- Simplify: less code, easier to create and maintain
- Multiple, compiled languages
- Fast and scalable
- Manageable
- Available
- Customizable and extensible
- Secure
- Tool support

ASP.Net Overview

- ASP.NET is a server-side technology
- ASP.NET provides services for creation (i.e. debugging), deployment and execution of
  - Web applications
  - Web services
- Web applications are built using Web Forms
  - Creation should be as easy as building Visual Basic applications

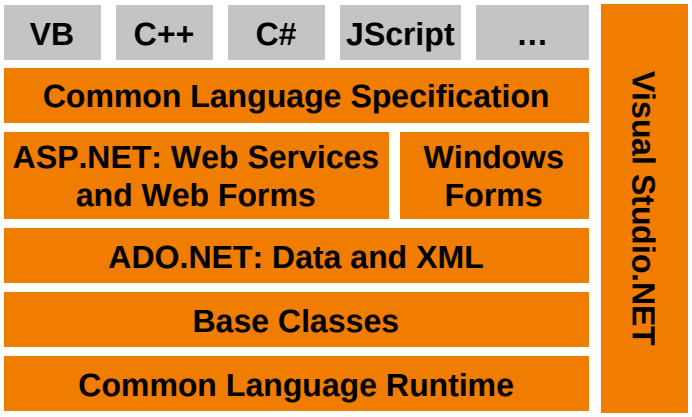
Key Features of ASP.NET

>Web Forms
>Web Services
>Built on .NET Framework
>Simple programming model
>Complete object model
>Maintains page state
>Multibrowser support
>XCOPY deployment
>XML configuration
>Session management
>Caching
>Debugging
>Extensibility
>Separation of code and UI
>Security
>ASPX, ASP side by side
>Simplified form validation
>Cookieless sessions

Architecture

- ASP.NET is built upon
  - .NET Framework
  - Microsoft Internet Information Server (IIS)
- MONO builds a Apache binding
  - ASP.NET 1.0 & 1.1 are completely supported
  - Some ASP.NET 2.0 features are already supported

.NET Framework



Internet Information Server (IIS)

- IIS MMC Snap-In (Internet Services Manager)
  - Tool to manage IIS
- Virtual Directories
  - Provides a mapping between URL and file path
  - E.g., the URL:
    `http://localhost/group1`
    maps to the file path:
    `C:\inetbub\wwwroot\lv\group1`

Example: HelloWorld.aspx

```
<html>
<%@ Page language="c#" %>
<head></head>
<script runat="server">
public void B_Click(object sender, System.EventArgs e)
{
   Label1.Text = "Hello, the time is " + DateTime.Now;
}
</script>
<body>
   <form method="post" runat="server">
      <asp:Button onclick="B_Click" Text="Push Me“
         runat="server" /> <p>
      <asp:Label id=Label1 runat="server" />
   </form>
</body>
</html>
```


Outline

> Background and Overview

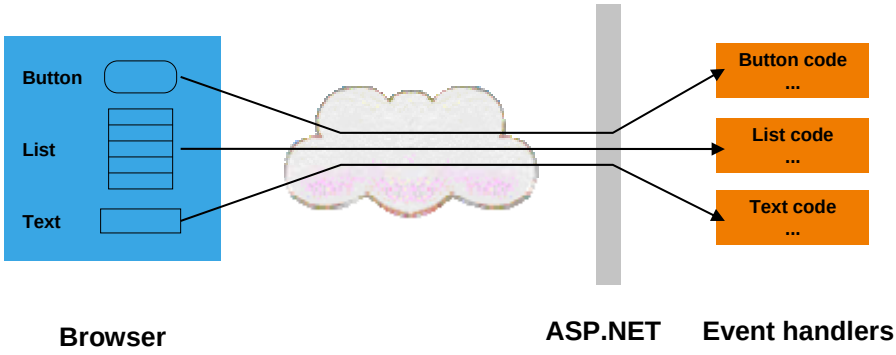
> **Programming Model**

> Programming Basics

Controls and Events I

- Server-side programming model
- Based on controls and events
  - Just like Visual Basic
  - Not “data in, HTML out”
- Higher level of abstraction than ASP
- Requires less code
- More modular, readable, and maintainable

Controls and Events II



ASP.NET Object Model

- Code executes on the web server in page or control event handlers
- Controls are objects, available in server-side code
  - Derived from **`System.Web.UI.Control`**
  - Similar to **`System.Windows.Forms.Control`**
- The web page is an object, too
  - Derived from **`System.Web.UI.Page`** which is a descendant of **`System.Web.UI.Control`**
  - A page can have methods, properties, etc.

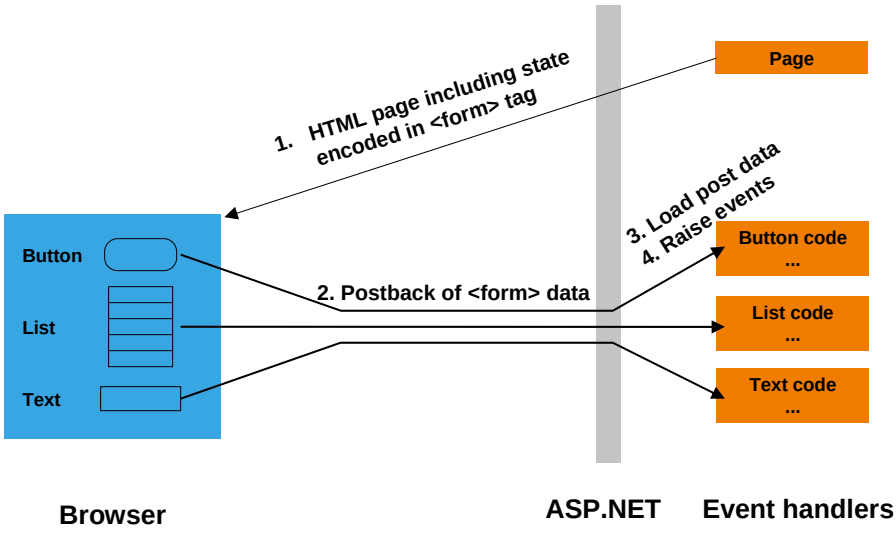
Postback I

- **`Page`** object generates an HTML form
- Upon user action (e.g. button pressed) the event is posted back to the server
  - Not all possible events are posted back
  - For example, mouse move or key press are not posted back because of performance issues
- In ASP and other server-side technologies the state of the page is lost upon postback...
  - Unless you explicitly write code to maintain state
  - This is tedious, bulky and error-prone

Postback II

- By default, ASP.NET maintains the state of all server-side controls during a postback
  - Can use **`method="post"`** or **`method="get"`**
- Server-side control objects are automatically populated during postback
  - No state stored on server
  - Works with all browsers

Postback III



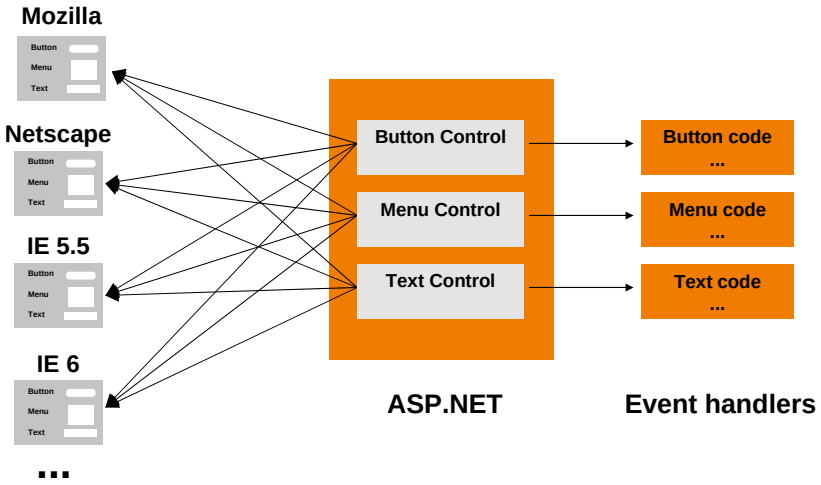
Server Side Controls

- Multiple sources to obtain controls
  - Built-in
  - 3rd party
  - User-defined
- Controls range in complexity and power
  - Button
  - Text
  - Drop down
  - Calendar
  - Data grid
  - …
- Can be populated via data binding

Automatic Browser Compatibility I

- Controls can provide automatic browser compatibility
- Can target uplevel or downlevel browsers
  - Uplevel browsers support additional functionality, such as JavaScript and DHTML
  - Downlevel browsers support HTML 3.2

Automatic Browser Compatibility II



Code-behind Pages

- Two styles of creating ASP.NET pages
  - UI and code in .aspx file
  - UI in .aspx file, code in code-behind page
    - Supported in Visual Studio.NET
- Code-behind pages allow you to separate the user interface design from the code
  - Allows programmers and designers to work independently

```
<%@ Codebehind="WebForm1.cs"
Inherits=WebApplication1.WebForm1" %>
```

Automatic Compilation I

> Just edit the code and hit the page (edit, save, and run)

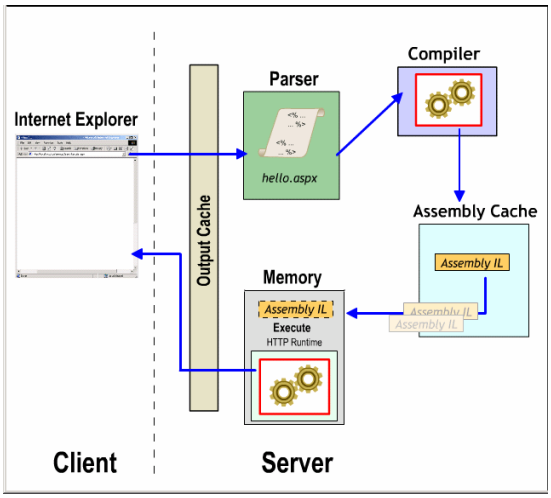
> ASP.NET will automatically compile the code into an assembly

> Compiled code is cached in the CLR Assembly Cache

> Subsequent page hits use compiled assembly

> If the text of the page changes then the code is recompiled

Automatic Compilation II



Outline

- Background and Overview
- Programming Model
- **Programming Basics**
  - **Page Object**
  - Controls
  - Events

Page Syntax

- The most basic page is just static text
  - Any HTML page can be renamed **`.aspx`**
- Pages may contain:
  - Directives: **`<%@ Page Language="C#" %>`**
  - Server controls: **`<asp:Button runat="server">`**
  - Code blocks: **`<script runat="server">…</script>`**
  - Server side comments: **`<%-- --%>`**
  - Render code: **`<%= %>`**
    - Use is discouraged; use **`<script runat=server>`** with code in event handlers instead

Page Directive

> **<%@ Page Language="C#" … %>**

> Lets you specify page-specific attributes, e.g.
  > **AspCompat**: Compatibility with ASP
  > **Buffer**: Controls page output buffering
  > **CodePage**: Code page for this .aspx page
  > **ContentType**: MIME type of the response
  > **ErrorPage**: URL if unhandled error occurs
  > **Inherits:** Base class of Page object
  > **Language**: Programming language
  > **Trace**: Enables tracing for this page
  > **Transaction**: COM+ transaction setting

> Only one page directive per .aspx file

Server Code Blocks

> Server code lives in a script block marked **runat="server"**

```
<script language="C#" runat="server">
<script language="VB" runat="server">
<script language="Jscript" runat="server">
```

> Script blocks can contain
  > Variables, methods, event handlers, properties
  > They become members of a custom **Page** object

Page Import Directive

- Adds code namespace reference to page
  - Avoids having to fully qualify .NET types and class names
  - Equivalent to the C# **`using`** directive

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>
```


Page Class

- The **`Page`** object is always available when handling server-side events
- Provides a large set of useful properties and methods, including:
  - **`Application`**, **`Cache`**, **`Controls`**, **`EnableViewState`**, **`ErrorPage`**, **`IsPostBack`**, **`IsValid`**, **`Request`**, **`Response`**, **`Server`**, **`Session`**, **`Trace`**, **`User`**, **`Validators`**
  - **`DataBind()`**, **`LoadControl()`**, **`MapPath()`**, **`Validate()`**

Outline

- Background and Overview
- Programming Model
- **Programming Basics**
  - **Page** Object
  - **Controls**
  - Events

Server Control Syntax

- Controls are declared as HTML tags with **`runat="server"`** attribute

```
<input type=text id=text2 runat="server" />
<asp:calendar id=myCal runat="server" />
```

- Tag identifies which type of control to create
  - Control is implemented as an ASP.NET class
- The **`id`** attribute provides programmatic identifier
  - It names the instance available during postback
  - Just like Dynamic HTML

Server Control Properties

> Tag attributes map to control properties

```
<asp:button id="c1" Text="Foo" runat="server">
<asp:ListBox id="c2" Rows="5" runat="server">
```

> Tags and attributes are case-insensitive

> Control properties can be set programmatically

```
c1.Text = "Foo";
c2.Rows = 5;
```


Maintaining State

> Controls maintain their state across multiple postback requests by default
  > Implemented using a hidden HTML field: **__VIEWSTATE**
  > Works for controls with input data (e.g. **TextBox**, **CheckBox**), non-input controls (e.g. **Label**, **DataGrid**), and hybrids (e.g. **DropDownList**, **ListBox**)

> Can be disabled per control or entire page
  > Set **EnableViewState="false"**
  > Lets you minimize size of **__VIEWSTATE**

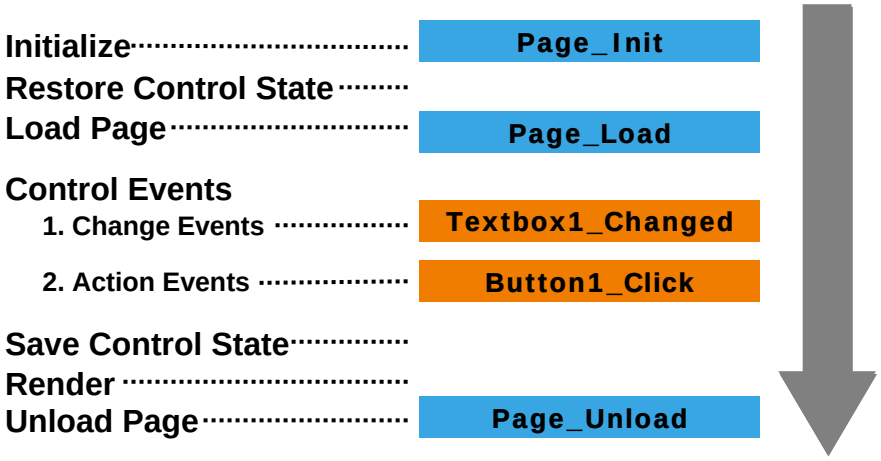
Outline

- Background and Overview
- Programming Model
- **Programming Basics**
  - **`Page`** Object
  - Controls
  - **Events**

Events

- Controls reacts to events
  - Enables clean code organization
  - Avoids the “Monster IF” statement
  - Less complex than ASP pages
- Code can respond to page events
  - e.g. **`Page_Load`**, **`Page_Unload`**
- Code can respond to control events
  - **`Button1_Click`**
  - **`Textbox1_Changed`**

Event Lifecycle



Page Loading I

> **Page_Load** fires at beginning of request
  - > Controls are already initialized
  - > Input control values already populated

```
protected void Page_Load(Object s, EventArgs e)
{
  message.Text = textbox1.Text;
}
```

Page Loading II

- **Page_Load** fires on every request
  - Use **Page.IsPostBack** to execute conditional logic
  - If a Page/Control is maintaining state then only initialize it when **IsPostBack** is false

```
protected void Page_Load(Object s, EventArgs e)
{
 if ( !Page.IsPostBack )
 {
    // Executes only on initial page load
    Message.Text = "initial value";
 }
 // Rest of procedure executes on every request
}
```


Server Control Events

- Change Events
  - By default, these execute only on next action event
  - E.g. **OnTextChanged, OnCheckedChanged**
  - Change events fire in random order
- Action Events
  - Cause an immediate postback to server
  - E.g. **OnClick**
- Works with any browser
  - No client script required, no applets, no ActiveX® Controls

How Postback works on Client Side

```
<form name="Form1" method="post" action="WebForm1.aspx"
                                                    id="Form1">
<a href="javascript:__doPostBack('Firstcontrol1','dec')">
   Decrease Number
</a>
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />

<script language="javascript">
<!--
   function __doPostBack(eventTarget, eventArgument) {
      var theform;
      theform = document.Form1;
      theform.__EVENTTARGET.value = eventTarget;
      theform.__EVENTARGUMENT.value = eventArgument;
      theform.submit();
   }
// -->
</script></form>
```


Wiring Up Control Events

> Control event handlers are identified by the tag

```
<asp:button onclick="btn1_click" runat=server>
<asp:textbox onchanged="text1_changed" runat=server>
```

> Event handler code

```
protected void btn1_Click(Object s, EventArgs e)
{
     Message.Text = "Button1 clicked";
}
```

Event Arguments

- Events pass two arguments:
  - The sender, declared as type **`object`**
    - Usually the object representing the control that generated the event
    - Allows you to use the same event handler for multiple controls
  - Arguments, declared as type **`EventArgs`**
    - Provides additional data specific to the event
    - **`EventArgs`** itself contains no data; a class derived from **`EventArgs`** will be passed
    - Read the fine manual to find out which subclass of **`EventArgs`** you have to expect

Change Events & AutoPostBack I

- Usually only action events trigger a postback
- Some controls can send postbacks for change events
  - CheckBox
  - ListControl
  - TextBox
- Postback is sent after the user
  - 1. changed a value
  - 2. tabbed out of the control
- Example
  - Automatically sum up values entered in text boxes

Change Events & AutoPostBack II

```
<%@ Page Language="C#" AutoEventWireup="True" %>
<html><head>
<script runat="server">

   protected void Page_Load(Object sender,EventArgs e)
   {
      int Answer = Convert.ToInt32(Value1.Text) +
                   Convert.ToInt32(Value2.Text);
      AnswerMessage.Text = Answer.ToString();
   }
</script></head>

<body><form runat="server">
   <asp:TextBox ID="Value1" AutoPostBack="True"
                runat="server"/>
   <asp:TextBox ID="Value2" AutoPostBack="True"
                runat="server"/>
   <asp:Label ID="AnswerMessage" runat="server"/>
</form></body>
</html>
```


Page Unloading

- **`Page_Unload`** fires after the page is rendered
  - Don't try to add to output
- Useful for logging and clean up

```
protected void Page_Unload(Object s, EventArgs e)
{
   MyApp.LogPageComplete();
}
```

Bibliography

> Matthias Lohrer. *Einstieg in ASP.NET*. Galileo Computing, 2002. http://www.galileocomputing.de/openbook/asp/

> Eric Gunnerson. *C# – Tutorial und Referenz*. Galileo Computing, 2002. http://www.galileocomputing.de/openbook/csharp/