

ASP.NET WebUserControls

Die Datenbindung von verschachtelten WebUserControls in der Praxis

André Achtermeier, Daniel Saager

Der Einsatz von WebUserControls (benutzerdefinierte Steuerelemente für modulare Software) gehört zu den täglichen Handgriffen des modernen ASP.NET Software-Engineers. Wenn jedoch WebUserControls verschachtelt zum Einsatz kommen, ist der Entwicklungsaufwand zumeist höher als gewünscht, da Daten ggf. hierarchisch angebunden werden müssen. Um in solchen Situationen das Rad nicht immer neu zu erfinden, werde ich in diesem Artikel beispielhaft UserControls verschachteln und ein paar Anregungen zur Implementierung geben.

„Wir wollen möglichst austauschbare Software-Komponenten realisieren, um unsere ASP.NET-Applikation einem breiten Markt zur Verfügung zu stellen“. Diesem Satz kann man ohne weitere Überlegungen in der Regel zustimmen. Wenn er allerdings zu oft fällt, hat die Applikation schnell ein paar verschachtelte WebUserControls im Einsatz. Der Entwickler steht nun vor dem Problem die Daten der einzelnen Hierarchiestufen bis zur obersten Stufe durchzureichen.

Die Anforderung im Blick

Wir sollen für ein Gasversorgungsunternehmen eine Kommunikationslösung implementieren. In einer Weboberfläche müssen verschiedene Nachrichtendefinitionen verwaltet werden. Das entsprechende UML-Diagramm sieht wie folgt aus:

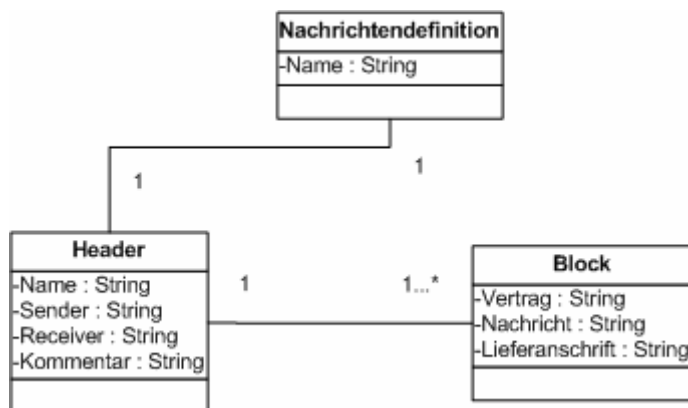


Abbildung 1: UML – Diagramm

Wie aus dem Diagramm ersichtlich, gibt es zu einer Nachrichtendefinition einen Header und mehrere Blöcke. (Die Attribute sind erst einmal zu vernachlässigen). Unser Kunde möchte gerne eine Nachrichtendefinition auf einer ASP.NET-Seite verwalten. Wir entschließen uns sowohl den Header, als auch den Block als WebUserControl zu realisieren, da wir für einen anderen Kunden ein ähnliches Gerüst benötigen. Die WebUserControls geben uns die notwendige Flexibilität.

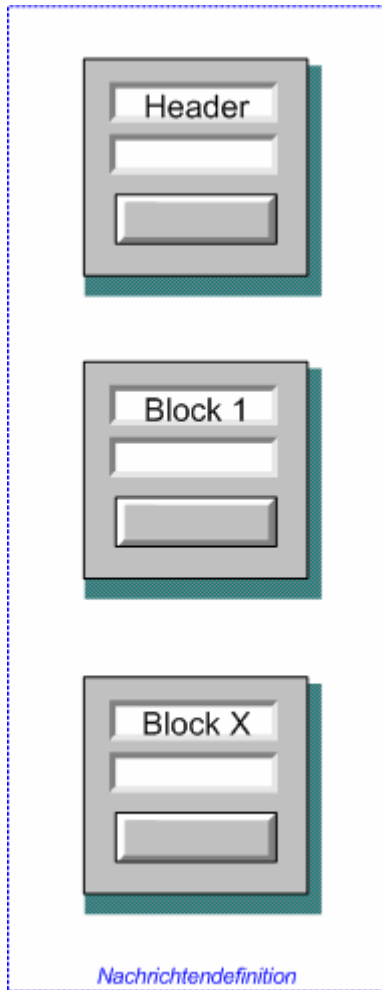


Abbildung 2: Seitenstruktur

Die Struktur der Seite wird in Abbildung 2 deutlicher. Hinzu kommt noch, dass es möglich sein muss mit einem Button-Click die kompletten Eingaben des Headers, als auch aller Blöcke zu speichern. Des Weiteren soll der Übersichtlichkeit genüge getan werden, indem jeder Block auf- und zuklappbar ist. Es muss demnach eine klassische Toggle-Funktionalität implementiert werden.

Die WebUserControls ‚Header‘ und ‚Block‘

Die grafische Gestaltung der beiden WebUserControls wird in den folgenden zwei Abbildungen dargestellt.

Das Screenshot zeigt das WebUserControl 'Header' mit folgenden Feldern:

- Sender***: Dropdown-Menü mit dem Wert 'shipper'. Rechts daneben: 'Identifizier' [shipper [ZEW Kennung]] und 'Rolle' [Broker or Sales office].
- Empfänger***: Dropdown-Menü mit dem Wert 'shipper'. Rechts daneben: 'Identifizier' [shipper [ZEW Kennung]] und 'Rolle' [Broker or Sales office].
- Vertrag**: Ein leeres Textfeld.
- Einspeisestation [ID]**: Textfeld, gefolgt von '[CA]:' und einem Dropdown-Menü mit dem Wert 'Keine Angabe'.
- Ausspeisestation [ID]**: Textfeld, gefolgt von '[CA]:' und einem Dropdown-Menü mit dem Wert 'Keine Angabe'.
- Speicherstation [ID]**: Textfeld, gefolgt von '[CA]:' und einem Dropdown-Menü mit dem Wert 'Keine Angabe'.
- Kommentar**: Ein Textfeld mit vertikalen Scrollbalken.

Abbildung 3: Der Header

Block: : N.N.

Shipper1 Identifier Rolle

Shipper2 Identifier Rolle

Vertrag

Einspeisestation [ID] [CA]:

Ausspeisestation [ID] [CA]:

Speicherstation [ID] [CA]:

Infopunkt Name Parameter Bezeichnung Faktor

Wertart

Kennzeichen

Einheit

Intervall

Zusatzwerte:

Abbildung 4: Der Block

Wie in Abbildung 4 ersichtlich wurde in der Kopfzeile des Blocks sowohl ein ImageButton für das Togglen, als auch ein ImageButton für das Löschen des Blocks implementiert.



[Angaben mit * sind Pflicht] [m_HIDDENHEADERTIMESTAMP][m_HIDDENHEADERUSER]

Header

Unbound Identifier Unbound Rolle

Kontakt Unbound

Unbound Identifier Unbound Rolle

Kontakt Unbound

Vertrag

Einspeisestation [ID] [CA]:

Ausspeisestation [ID] [CA]:

Speicherstation [ID] [CA]:

Kommentar



[Placeholder "m_PHBlocks"]

Abbildung 5: Der Aufbau des Headers im VS.NET

Abbildung 5 zeigt den Aufbau des HeaderControls im VS.NET und dient dem Verständnis. Wie zu sehen beherbergt der Header ein ASP.NET-PlaceholderControl für die (sich wiederholenden) Blöcke. In diesem Konstrukt findet sich die 1:n-Beziehung zwischen Header und Block wieder.

Technische Umsetzung

Für die technische Umsetzung setzen wir auf Clientseite ein ‚DataTransferObject‘ (DTO) nach dem Prinzip des DataTransferObject-Patterns ein (näheres dazu siehe [1]). Als Implementierungssprache entscheiden wir uns für C#. Die Quellcodebeispiele dienen lediglich der Verdeutlichung und sind nicht bis ins kleinste Detail abgebildet.

Seifenblasen

Das .NET-Framework liefert von Haus aus eine Technik, die es erlaubt Events von Child-Controls dem übergeordneten Parent-Control zu übergeben. In der MSDN[2] wird diese Vorgehensweise mit „Bubbling an Event“ über- und beschrieben. Die Technik ermöglicht es die komplette Hierarchie von verschachtelten WebUserControls zu durchlaufen und Informationen durchzureichen.

Die Implementierung dieser Technik ist nicht wirklich kompliziert. Im Child-Control wird an gewünschter Stelle die Funktion `RaiseBubbleEvent(...)` aufgerufen. Die Informationen landen im Parent-Control in der zu überschreibenden Funktion `OnBubbleEvent(...)`. Die MSDN gibt weiterführende Informationen und Beispiele.

Für unser Beispiel benötigen wir dieses Wissen um dem Header mitzuteilen, dass ein neuer Block hinzugefügt, gelöscht oder geändert wurde. Dies ist notwendig, da der Header die Daten der Blöcke in seinem Placeholder-Control neu anbinden muss. Die Informationen werden demnach vom Block zum Header ‚gebubblt‘.

Block anlegen

Beginnen wir unser Fallbeispiel mit einem leeren PlaceholderControl im Header. Der User entschliesst sich zuerst einen Block anzulegen. Um das entsprechende WebUserControl für einen Block zu laden, implementieren wir folgende Funktionalität im Event des ‚AddBlock‘-Buttons:

- Neues DTOBlock erzeugen
- Block der Liste hinzufügen
- ‚BindBlocksData()‘ aufrufen (um einen ‚teuren‘ Refresh der Page zu verhindern)

```

DTOBlock block = new DTOBlock();

//Negative ID (vom Typ LONG) setzen
block.MyID = UCHeader.UCBlockId;
//Static ID runterzählen
UCEditMessage.UCBlockId--;
//Als nicht persistent markieren
block.IsPersistent = false;

//Neuer Block der Liste hinzufügen
UCHeader.blocks.Add(block);

//Hierzu mehr im nächsten Abschnitt
this.BindBlocksData();

```

Wir erreichen die Eindeutigkeit eines Blocks über die Verwaltung einer statischen, negativen ID (CustomProperty ‚MyID‘). Mit dieser Basis können wir später die Toggle-Funktionalität implementieren.

Datenbindung

Im vorigen Abschnitt wurde bereits die Methode `BindBlocksData()` eingeführt. Schauen wir uns die Methode einmal im Detail an. Die Aufgaben dieser Methode sind

- WebUserControls vom Typ Block in den Placeholder laden
- Die einzelnen Controls der Blöcke mit ‚SetControls()‘ initialisieren

Wir erreichen diese Anforderung mit folgendem Code:

```

private void BindBlocksData()
{
    //Placeholder zurücksetzen
    this.m_PlaceholderBlocks.Controls.Clear();

    if(UCHeader.blocks != null)
    {
        foreach(DTOBlock dto in UCHeader.blocks)
        {
            //Laden eines WebUserControls vom Typ Block
            this.LoadBlockControl(dto);
        }
    }
}

```

```

foreach(Control con in this.m_PlaceholderBlocks.Controls)
{
    if(con.GetType().BaseType == typeof(UCBlock))
    {
        UCBlock uc = (UCBlock)con;
        //Datenbindung der Controls
        uc.SetControls();
    }
}

private void LoadBlockControl(DTOBlock dto)
{
    UCBlock blockControl = (UCBlock)LoadControl("UCBlock.ascx");
    blockControl.ID = dto.Id.ToString();
    blockControl.MyID = dto.Id;
    this.m_PlaceholderBlocks.Controls.Add(blockControl);
}

//Diese Methode ist im WebUserControl Block implementiert
public void SetControls()
{
    ...
    this.m_TextboxContract.Text = this.m_dtoBlock.Contract;
    ...
}

```

Durch diese Vorgehensweise erreichen wir, dass sowohl persistente Objekte, als auch nicht persistente Objekte mittels DTO beim Client „bekanntgemacht“ werden. Das DTO des jeweiligen BlockControls sorgt für die Verfügbarkeit der gemachten Eingaben in den Controls auch bei weiteren, gefeuerten Events.

Ein weiterer wichtiger Punkt in diesem Kontext ist, dass die WebUserControls für die Blöcke bereits im OnInit(...) der Seite geladen werden müssen. Dies ist notwendig, da die Control-Inhalte aus dem Viewstate zwischen OnInit(...) und PageLoad(...) zurückgesetzt werden (vgl. ControlExecutionLifetime [3]). Wenn in der „Load-View-State-Phase“ die Controls noch nicht da sind, können die Inhalte üblicherweise auch nicht gesetzt werden. Deshalb nehmen wir ein kleines Refactoring vor und lagern einen Teil des Codes aus BindBlocksData() in eine Methode BindBlocks() aus.

```

private void BindBlocks()
{
    //Ausgelagerter Teil
    this.m_PlaceholderBlocks.Controls.Clear();

    if(UCHeader.blocks != null)
    {
        foreach(DTOBlock dto in UCHeader.blocks)
        {
            this.LoadBlockControl(dto);
        }
    }
}

```

Dieses Code-Fragment sorgt dafür, dass die WebUserControls für die Blöcke rechtzeitig zur Verfügung gestellt werden. Ändern sich Daten in einem Block, so wird der Header mit dieser Vorgehensweise darüber informiert und ist in der Lage für eine korrekte Darstellung seiner Blöcke zu sorgen.

Block auf, Block zu

Wie eingangs beschrieben wollen wir nur einzelne Blöcke der Übersichtlichkeit wegen auf- und zuklappen. Wir müssen demnach jede Instanz eines WebUserControls ‚UCBlock.ascx‘ getrennt ansprechen können. Aus diesem Grund haben wir für jeden Block (egal ob persistent oder nicht) eine eindeutige ID vergeben. Im WebUserControl selber platzieren wir einen statischen Hashtable, der nach aussen sichtbar ist:

```

/// <summary>
/// Der Hashtable enthält KEY:BLOCKID und VALUE:CONTENTISVISIBLE
/// </summary>
public static Hashtable m_VisibilityHashtable = new Hashtable();

```

Im Click-Event des Toggle-Buttons für den Block finden wir folgenden Code wieder:

```

this.CheckToggle();
bool isVisible = (bool)UCBlock.m_VisibilityHashtable[this.MyID];
UCBlock.m_VisibilityHashtable.Remove(this.MyID);

```

```

UCBlock.m_VisibilityHashtable.Add(this.MyID,!isVisible);
this.HandleHashtable();

private void CheckToggle()
{
    if(!UCBlock.m_VisibilityHashtable.ContainsKey(this.MyID)) //Block nicht vorhanden
    {
        UCBlock.m_VisibilityHashtable.Add(this.MyID,false);
    }
}

private void HandleHashtable()
{
    if((bool)UCBlock.m_VisibilityHashtable[this.MyID])
    {
        this.SetBlockInVisibility();
    }
    else
    {
        this.SetBlockVisibility();
    }
}

```

Die Methoden ‚SetBlockVisibility()‘ und ‚SetBlockInVisibility()‘ setzen lediglich das Property ‚visible‘ der Container-Tabelle für den Inhalt auf ‚true‘ oder ‚false‘.

Wir können nun neue Blöcke hinzufügen und jeden einzelnen Block auf- oder zuklappen.

Block weg

Bleibt noch die Frage offen, wie man einen Block aus der Liste löscht. Hierbei halten wir uns an die empfohlene Vorgehensweise in der MSDN [4]. Im Click-Event des Löschen-Buttons eines Blocks schreiben wir folgenden Code:

```

//Identifizier für das BubbleEvent des UCHeaders
HttpContext.Current.Session["ISDELETEBLOCK"] = "TRUE";
this.RaiseBubbleEvent(sender,e);

```

Um die Datenbindung der Blöcke zu kontrollieren, setzen wir eine Session-Variable, die im Bubble-Event des UCHeaders abgefragt wird. Ist diese Variable vorhanden, wird folgender Code im WebUserControl für den Header ausgeführt:

```

protected override bool OnBubbleEvent(object source, EventArgs args)
{
    //Auslöser war das DeleteBlockEvent des UCBlocks
    if(HttpContext.Current.Session["ISDELETEBLOCK"]!=null)
    {
        // this.BindBlocks() wird in OnInit ausgeführt (siehe voriger Abschnitt)
        this.BindHeaderData(); //analoge Vorgehensweise wie bei den Blöcken
        this.BindBlocksData();

        HttpContext.Current.Session.Remove("ISDELETEBLOCK"); //Wieder zurücksetzen
    }
    return base.OnBubbleEvent (source, args);
}

```

Fazit

Der Artikel hat Möglichkeiten aufgezeigt, verschiedene Instanzen von einem WebUserControl anzusprechen und Übersichtlichkeit mit ‚Toggle‘-Funktionalität zu realisieren. Des Weiteren wurde der Leser angeregt den Einsatz von verschachtelten WebUserControls mit Hilfe des DTO-Patterns zu vereinfachen. Auch wurde auf die Problematik der Datenbindung bei n-Instanzen eines verschachtelten WebUserControls eingegangen und ein Vorschlag zur Lösung gegeben. Natürlich erhebt dieser Artikel keinen Anspruch auf „den einzig richtigen Weg“. Auch bei diesem Teil des Software-Engineering gilt: „Viele Wege führen nach Rom.“

Literatur

- [1] Martin Fowler: Patterns of Enterprise Application Architecture; Addison Wesley 2003
- [2] MSDN: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfssystemwebuicontrolclassraisebubbleeventtopic.asp>
- [3] MSDN: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcontrolexecutionlifecycle.asp>

[4] MSDN: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconbubblingcommandevent.asp>

[5] <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconwebusercontrolsvscustomwebcontrols.asp>

[6] ASP.NET Professional: Wrox Press Ltd. 2001 (Einführung in die Thematik)

>>Kasten Beginn<<

DTO-Pattern in Kürze

Das DTO-Pattern geht von einer service-basierten Architektur aus. Der Client arbeitet nur mit sog. DataTransferObjects. Diese Objekte sind Container für Daten der Business Objekte. Das DTO wird gelesen und die Controls werden mit diesen Daten angebinden. Zum Schreiben wird das gefüllte/upgedatete DTO dem Service wieder übergeben. Ein Assembler auf Serverseite assembliert das DTO in die entsprechenden Business Objekte und führt mit den Business Objekten Transaktionen aus. Dieses Vorgehen hat den Vorteil, dass der Client keine Kenntnisse über Methodenimplementierung und Objekt-Modell haben muss und nur „dumm“ mit dem DTO arbeitet.

>>Kasten Ende<<

>>Kasten Beginn<<

WebUserControls vs. WebCustomControls

Die MSDN [6] liefert einen guten Überblick, wann es sinnvoll ist ein WebUserControl oder ein WebCustomControl zu benutzen. WebCustomControls stehen nach der Fertigstellung in kompiliertem Code zur Verfügung und sind schwieriger zu gestalten, während WebUserControl auch nur aus HTML bestehen können und damit gut im VS.NET gestaltet werden können. Der Vorteil von WebCustomControls liegt in der Wiederverwendbarkeit. So kann bspw. ein solches Control über die Toolbar im VS.NET zur Verfügung gestellt werden. Es ist auch nur eine Registrierung im GAC (Global Assembly Cache) notwendig, während eine Kopie des WebUserControls in jeder Applikation vorhanden sein muss.

>>Kasten Ende<<