Återanvändbara gränssnittskomponenter i ASP.NET

Jonas Paro

9 april 2008
Examensarbete i Datavetenskap, 20 poäng
Handledare vid CS-UmU: Jurgen Burstler
Examinator: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 Umeå

Sammanfattning

Denna rapport undersöker hur man kan underlätta återanvändning av presentationslagret i en n-skiktad lösning i ramverket ASP.NET. Rapporten jämför designmönstren Model-View-Controller och Model-View-Presenter som grund för presentationslagret. Slutsatsen är att Model-View-Presenter lämpar sig bättre för presentationslagret i ASP.NET. Rapporten visar hur presentationslagret i Acino AB:s Internetkontor kan utformas enligt Model-View-Presenter för att underlätta återanvändning. Ett antal gränssnittskomponenter implementeras med hjälp av Custom-Server-Controls, och rapporten visar hur dessa passar in i Model-View-Presenter. Målsättningen med dessa komponenter är att de skall kunna återanvändas i Acinos framtida applikatoner.

Abstract

Reusable user interface components in ASP.NET

This paper examines how to facilitate reuse in the presentation-tier of an n-tier solution in the ASP.NET framework. The paper examines the design patterns Model-View-Controller and Model-View-Presenter as a basis for the presentation-tier. The results show that Model-View-Presenter is more suitable as a basis for the presentation-tier in the ASP.NET framework. The paper shows how the presentation-tier of Acino AB:s Internetkontoret can be organized using Model-View-Presenter to facilitate reusability. A number of user interface components are implemented using Custom Server Controls and it is shown how these components fit into the Model-View-Presenter design pattern. The goal of these components is that they should be reusable in Acinos future applications.

Innehåll

1 Introduktion **1**

2 Problembeskrivning **3**

2.1 Problemformulering . . . 3

2.2 Mål och Syfte . . . 3

2.3 Metod . . . 3

3 Litteraturstudie och teknisk fördjupning **5**

3.1 Designmönster för presentationslagret . . . 5

3.2 GoF-mönster . . . 6

3.3 Decorator . . . 6

3.3.1 Decorator i ASP.NET . . . 7

3.4 Strategy . . . 8

3.4.1 Strategy i ASP.NET . . . 9

3.5 Observer . . . 9

3.5.1 Observer i ASP.NET . . . 9

3.6 Arkitektiska mönster . . . 11

3.7 Model-View-Controller . . . 11

3.7.1 Model . . . 11

3.7.2 View . . . 12

3.7.3 Controller . . . 12

3.7.4 Model-View-Controller i ASP.NET . . . 12

3.8 Model-View-Presenter . . . 16

3.9 N-skiktade lösningar och MVC/MVP . . . 18

3.10 Tekniker för återanvändning i ASP.NET . . . 19

3.10.1 User Controls . . . 19

3.10.2 Custom Server Controls . . . 20

4 Arbetsprocess **21**
4.1 Beskrivning av Internetkontoret 21
4.2 Arbetsprocess 21

5 Resultat **27**
5.1 ValidatingTextBox 27
5.2 SelectionBox 27
5.3 SearchBox 28
5.4 Användning i Acinos ramverk och Internetkontoret 28

6 Sammanfattning **31**
6.1 Begränsningar 31
6.2 Framtida arbete 32

7 Tack **33**

Referenser **35**

Figurer

3.1 Decorator 6
3.2 Strategy 8
3.3 Observer 9
3.4 Model-View-Controller 12
3.5 Implementationsexempel MVC 13
3.6 Model-View-Presenter 16
3.7 MVC och N-tier 19
3.8 User Control 20

4.1 Internetkontoret 22
4.2 Internetkontorets sökfunktion 23

5.1 Presentationslagret 29

Kodexempel

3.1 Decorator 7
3.2 Delegate+Event:Observer 10
3.3 MVC View 13
3.4 MVC Controller 14
3.5 MVP View 16
3.6 MVP Presenter 17
4.1 SearchBox i aspx-sidan 23
4.2 SearchBox i aspx.cs 23
4.3 SearchBox.cs 24
4.4 SearchBox v2 i aspx-sidan 24
4.5 Searchbox v2 i aspx.cs 24

Kapitel 1

Introduktion

Mjukvaruåteranvändning är ett koncept som introducerades 1968 av Douglas McIlroy [12]. Syftet med denna process är att skapa nya system genom att använda existerande mjukvara. Det finns en mängd olika tekniker för mjukvaruåteranvändning såsom komponenter, designmönster och ramverk. Till en början fokuserades mjukvaruåteranvändning kring att skapa komponenter [18]. En komponent är ett stycke paketerad mjukvara som erbjuder en lösning på ett väl avgränsat problem. Tanken med komponenter är att en utvecklare skall kunna använda sig av mekanismer istället för att utveckla dem [12]. Designmönster är en mer abstrakt form av återanvändning. Ett designmönster kan sägas vara ett recept på ett väl beprövat tillvägagångssätt för att lösa ett ofta förekommande problem [7]. Till skillnad från komponenter så innebär ett designmönster inte någon färdigt skriven kod som återanvänds, utan klassdiagram och andra modeller som illustrerar en bra lösningsstrategi för det givna problemet. Implementationen av designmönstret lämnas till utvecklaren. Ramverk är en mer avancerad form av återanvändning som kan sägas vara en kombination av komponenter och designmönster [11]. I takt med att teknikerna förbättras och förståelsen för mjukvaruåteranvändning har ökat har de mer avancerade formerna av återanvändning, såsom ramverk, vuxit fram och blivit en viktig del av mjukvaruutvecklarens verktyg [11]. ASP.NET är ett sådant ramverk. Detta ramverk erbjuder utvecklaren massvis med funktionalitet för att underlätta utvecklingen av dynamiska webbsidor. Exempelvis erbjuds färdiga komponenter som representerar vanligt förekommande grafiska gränssnittselement, såsom knappar, dropdownlistor och textfält. Dessa gränssnittskomponenter är en stor tillgång för utvecklaren men de flesta av dem är generella och tänkta för storskalig återanvändning inom alla typer av webbapplikationer. För en utvecklare eller ett företag som utvecklar flera liknande applikationer eller applikationer inom samma applikationsfamilj kan det löna sig att sätta samman egna, mer specifika gränssnittskomponenter som är skräddarsydda för den egna applikationsfamiljen och kan återanvändas i framtida applikationer. En sådan komponent skulle exempelvis kunna vara en knapp med ett för applikationsfamiljen unikt beteende och utseende. Denna rapport

handlar om att identifiera, konstruera och implementera sådana applikationsspecifika, återanvändningsbara gränssnittskomponenter.

Acino AB bygger och förvaltar produkter för intelligent interaktion över internet. Internetkontoret är en webbapplikation för att hantera försäkringsärenden över Internet. Applikationen är byggd av Acino AB och vilar på deras ramverk. Ramverket är byggt på Microsofts ASP.NET ramverk. Ansatsen med Acinos arkitektur är att en applikation som anpassats för en kund skall utgöra en instans av arkitekturen. Således kan Internetkontoret med en kundanpassning (exempelvis mot en viss pensionskassa) sägas vara en instans av Acinos arkitektur. Instanserna av Acinos arkitektur utgör så kallade treskiktade lösningar. Med detta innebär att dataåtkomst, affärslogik och presentationslogik återfinns i olika lager. En kundanpassning av en applikation utgörs normalt sett av presentationslagret, och möjligtvis även av affärslogikslagret. Acinos önskan är att kunna återanvända så stora delar som möjligt av presentationslagret mellan olika kundanpassningar och applikationer.

Kapitel 2 ger en detaljerad bild av problemet.

Kapitel 3 är en litteraturstudie som undersöker vilka designmönster som kan användas för att skapa ett presentationslager som kan innehålla återanvändbara komponenter. Studien ger också en översikt över ett antal tekniker och verktyg för återanvändning som finns tillgängliga i ASP.NET.

Kapitel 4 beskriver hur designmönstren använts för att implementera ett antal användargränssnittskomponenter.

Kapitel 5 ger en ingående beskrivning av de implementerade komponenterna.

Kapitel 6 sammanfattar arbetet och diskuterar framtida arbete.

Kapitel 7 innehåller tack.

Kapitel 2

Problembeskrivning

2.1 Problemformulering

Det är en stor utmaning att återanvända delar av presentationslagret i n-skiktade lösningar. I Acinos Internetkontor innebär varje ny kundanpassning att stora delar av presentationslagret antingen skrivs om eller klipps och klistras från en lösning till en annan. De delar av presentationslagret som kan återanvändas behöver en bättre förpackning. Problemet är att öka återanvändbarheten i presentationslagret utan att göra ingrepp i övriga lager i det ramverk som Acinos Internetkontor vilar på.

2.2 Mål och Syfte

Målet med denna rapport är att undersöka om det går att underlätta återanvändningen av presentationslagret i en n-skiktad lösning genom att skapa gränssnittskomponenter. En gränssnittskomponent är ett stycke återanvändbar kod med specialiserat beteende och utseende. Målet är att identifiera delar av presentationslagret som går att förpacka som återanvändbara komponenter, samt undersöka hur presentationslagret i Acinos arkitektur kan struktureras för att underlätta återanvändning. Gränssnittkomponenterna skall implementeras i Acinos Internetkontor.

2.3 Metod

Arbetet har delats upp i två delar, en litteraturstudie och en implementationsdel.

Litteraturstudien undersöker vilka designmönster som kan ligga till grund för ett presentationslager som innehåller återanvändbara komponenter. Litteraturstudien jämför hur olika designmönster kan implementeras i ASP.NET. Litteraturstudien innehåller också en teknisk fördjupning i de tekniker och verktyg som finns inom ASP.NET för att underlätta återanvändning och skapandet av komponenter.

Implementationsdelen implementerar ett antal komponenter som skall passa in i Acinos applikation Internetkontoret och kunna återanvändas i framtida kundanpassningar. Den teoretiska fördjupningen och de designmönster och implementationsstrategier som undersökts ligger till grund för implementationen av komponenterna.

Kapitel 3

Litteraturstudie och teknisk fördjupning

Hur skapar man bra, återanvändbar mjukvara? Svaret på den frågan har gäckat utvecklare ända sedan begreppet mjukvaruåteranvändning myntades. En metod för att underlätta mjukvaruåteranvändning, som fått starkt fotfäste är användandet av designmönster. Ett designmönster kan beskrivas som en lösningsstrategi till ett återkommande problem. Målet med ett designmönster är att dokumentera denna lösning så att den kan återanvändas i många olika projekt [7]. Det finns mängder av beprövade designmönster för att adressera olika typer av problem, inom olika domäner. Denna studie undersöker och jämför designmönster som är speciellt avsedda för presentationslagret. Syftet med studien är att finna vilket eller vilka designmönster som lämpar sig bäst för att konstruera ett presentationslager som innehåller återanvändbara komponenter. För att göra detta krävs en ingående diskussion om hur dessa designmönster kan implementeras i ASP.NET. Detta kapitel innehåller också en fördjupning i de tekniker som finns i ramverket ASP.NET för att skapa återanvändbara komponenter.

Designmönster för presentationslagret i kombination med befintliga tekniker i ASP.NET kan vara en god bit på vägen till att skapa bra, återanvändbar mjukvara [22].

3.1 Designmönster för presentationslagret

Inom presentationslagret finns flertalet unika problem, som inte uppstår i något av de andra lagren i en n-skiktad lösning. Presentationslagret ansvarar för applikationens flöde, validering av användarens indata, händelsehantering och rendering av gränssnittet. Att konstruera ett presentationslager för återanvändning innebär att samtliga av dessa problem måste adresseras.

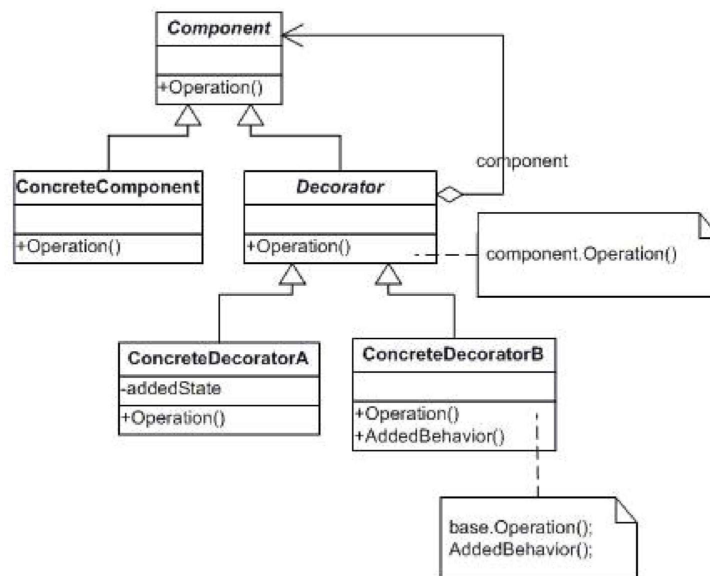
3.2 GoF-mönster

Av de välkända GoF-mönstren finns ett flertal som lämpar sig för användning i presentationslagret [7]. Nedan beskrivs ett urval av de mest intressanta mönstren samt hur de kan implementeras i ASP.NET.

3.3 Decorator

Decorator-mönstret kan användas när man vill addera beteende till ett objekt dynamiskt. Decorator är ett alternativ till att skapa sub-klasser som ger mer flexibilitet och därmed bättre återanvändbarhet.

Klassdiagram för Decorator visas i 3.1.



Figur 3.1: Decorator

Tanken med Decorator-mönstret är att innesluta objektet som skall dekoreras i ett annat objekt, som kallas Decorator. Decorator-objektets interface är likadant som det dekorerade objektet plus den funktionalitet som skall läggas till. Genom att skapa flera Decorators i varandra kan man addera flera lager av utseende och beteende till ett objekt. Decorator-mönstret är intressant för presentationslagret eftersom det erbjuder ett flexibelt sätt att påverka en komponents utseende.

3.3.1 Decorator i ASP.NET

Ramverket ASP.NET innehåller inget specifikt stöd för Decorator, men alla nödvändiga verktyg för att implementera det på ett smidigt sätt finns [4]. Ett exempel på hur Decorator kan implementeras i ASP.NET presenteras i 3.1. Exemplet bygger på ett exempel presenterat av [14]. I exemplet skapas ett enkelt dokument som kan ha sidhuvud, brödtext och sidfot. Istället för att ärva utseendet för sidhuvudet och sidfoten skapas dessa som decorators som dekorerar brödtexten.

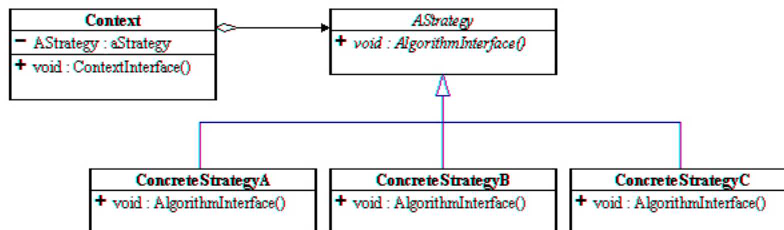
Kodexempel 3.1: Decorator

```
public interface IReport{
    void Print();
}

public class Report : IReport{
    public void IReport.Print(){
    HttpContext.Current.Response.Write
    ("Detta är brödtext.");
    }
}

public class ReportHeaderDecorator : IReport{
    private IReport _innerReport;
    public ReportHeaderDecorator(IReport innerReport){
    //referens till objektet som dekoreras
    _innerReport = innerReport;
    }

    public void IReport.Print(){
    //addera sidhuvud dekorering
    HttpContext.Current.Response.Write
    ("<h3>Detta är sidhuvudet</h3>");
    HttpContext.Current.Response.Write("<hr />");

    //Den dekorerade rapportens utskrift
    _innerReport.Print();
    }
}

public class ReportFooterDecorator : IReport{
    //referens till det dekorerade objektet
    private IReport _innerReport;
    public ReportFooterDecorator(IReport innerReport){
        _innerReport = innerReport;
    }

    public void IReport.Print(){
    //Den dekorerade rapportens utskrift
```

```
    _innerReport.Print();
    //Addera sidfot

    HttpContext.Current.Response.Write("<hr_/>");
    HttpContext.Current.Response.Write
    ("<h6>Detta_är_en_sidfot.</h6>");
    }
}

//implementera decorator
IReport myReport;

myReport = new Report();

//skapa basrapporten
myReport = new ReportHeaderDecorator(myReport);

//addera sidhuvud
myReport = new ReportFooterDecorator(myReport);

//addera sidfot
myReport.Print();
```

3.4 Strategy

Designmönstret Strategy kan användas när man vill att en komponent skall kunna använda sig av olika algoritmer beroende på vilken klient som använder den. Strategymönstret är intressant i presentationslagret eftersom det erbjuder ett flexibelt sätt att ändra en komponents beteende beroende på vem som är inloggad på applikationen.



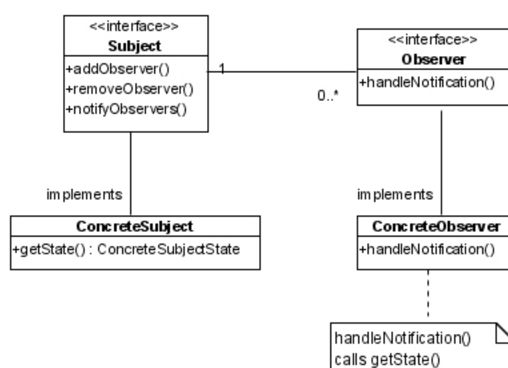
Figur 3.2: Strategy

3.4.1 Strategy i ASP.NET

Att implementera Strategy i ASP.NET är förhållandevis simpelt. Genom att definiera metoder som interface och delegera implementationen av metoderna till instanser av detta interface uppnås den önskade effekten. Strategy-mönstret diskuteras inte mer ingående i denna rapport, men det är värt att notera att strategy är en del av det sammansatta mönstret Provider som används i ASP.NET för att erbjuda ett sätt för användare att kontrollera beteendet hos välkända API:er.

3.5 Observer

Designmönstret Observer är ett mycket användbart mönster för applikationer med grafiska gränssnitt. Mönstret definierar en 1-N relation mellan objekt så att förändringar i ett objekt kommuniceras till flera andra objekt. Observer låter denna kommunikation ske utan att det observerade objektet behöver känna till de objekt som observerar det. Observer-mönstret är intressant för presentationslagret eftersom det är ett bra sätt att implementera knappar och andra liknande gränssnittsstrukturer som innefattar interaktion med användaren.



Figur 3.3: Observer

För att ett objekt skall kunna observera ett annat objekt enligt Observer-mönstret krävs att det ärver från den abstrakta klassen Subject. Observatören ärver från den abstrakta klassen Observer.

3.5.1 Observer i ASP.NET

Att implementera Observer i ASP.NET kan göras genom att skapa en struktur av klasser enligt 3.3 men detta är i själva verket onödigt. Ramverket innehåller nämligen redan en mycket enkelt struktur för detta mönster, genom den teknik som kallas Events och Delegates. Klassen som skall observeras definierar en

Delegate och ett Event. Denna Delegate kan sedan ersättas med en metod från observatören när det observerade objektet avfyrar sitt Event. Detta illustreras i 3.2. Exemplet illustrerar en tjänst som spelar ett musikalbum och en tjänst som tar betalt för varje spelat album. Betaltjänsten observerar musiktjänsten enliget Observer-mönstret genom Delegates och Events. Exemplet bygger på exempel presenterade av [15].

Kodexempel 3.2: Delegate+Event:Observer

```
public class Album {
    private String name;
    public delegate void PlayHandler(object sender);
    public event PlayHandler PlayEvent;

    public Album(String name){ this.name = name; }

    public void Play(){
        Notify();
        //kod för att spela album
    }

    private void Notify(){
        if(PlayEvent != null)
            PlayEvent(this);
    }

    public String Name{
        get { return name; }
    }
}

****

public class BillingService{
    public void Update(object subject){
        if(subject is Album)
            GenerateCharge((Album)subject);
    }

    private void GenerateCharge(Album theAlbum) {
        //kod för att ta betalt
    }
}

****

class Client{
    [STAThread]
```

```
    static void Main(string[] args){
        BillingService billing = new BillingService();
        Album album = new Album("Up");

        album.PlayEvent +=
        new Album.PlayHandler(billing.Update);
        album.Play();
    }
}
```

3.6 Arkitektiska mönster

Presentationslagret påminner i många avseenden om traditionell skrivbordsapplikation med ett grafiskt gränssnitt. Arkitektiska designmönster för att skapa applikationer med grafiska gränssnitt har funnits ändå sedan grafiska gränssnitt dök upp. Ett arkitektiskt designmönster är ett mönster som föreslår lösningar på en rad sammanhängande och ofta storskaliga problem [23]. Arkitektiska designmönster är ofta sammansatta designmönster som består av flera underliggande atomära designmönster. Ett atomärt designmönster kan inte delas upp i mindre delar på ett meningsfullt sätt [20]. Model-View-Controller är det mest använda av dessa arkitektiska gränssnitt för applikationer med grafiska gränssnitt [6]. Av den anledningen är MVC en naturlig utgångspunkt för att hitta ett arkitektiskt designmönster som kan ligga till grund för ett presentationslager som innehåller återanvändbara komponenter.

3.7 Model-View-Controller

Begreppet Model-View-Controller har sitt ursprung i en teknisk rapport skriven av Tryggve Reenskaug 1979 [19]. Det ursprungliga syftet med detta mönster var att presentera en bra strategi för att skapa system med grafiska användargränssnitt. Model-View-Controller är ett sammansatt, arkitektiskt designmönster som kan sägas innehålla både Decorator, Observer och Strategy [7]. Som visats ovan så går dessa mönster utmärkt att implementera i ASP.NET, eller finns redan inbyggda i ASP.NET och det känns därför som ett rimligt antagande att även MVC som helhet kan implementeras i ramverket.

Det råder en viss förvirring och många delade meningar kring vad MVC egentligen är och hur det kan användas [6]. Nedan följer en beskrivning av MVC så som det presenterades i den ursprungliga rapporten [19].

3.7.1 Model

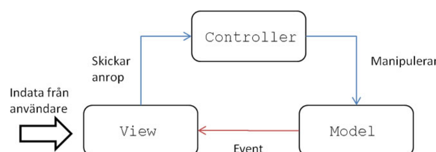
Model representerar en modell av identifierbara delar av ett problem eller en domän. Model existerar utan vetskap om View och kan stödja flera olika View-objekt.

3.7.2 View

View representerar en vy av Model. Det är ett presentationsfilter som visar vissa valda delar av modellen. View får sin data från Model genom att ställa frågor. För att kunna göra detta krävs det att View känner till Models semantik.

3.7.3 Controller

Controller arrangerar en eller flera Views på skärmen och bestämmer vad som skall visas för användaren. Controller tar hand om användarens output och förmedlar denna till View i form av meddelanden.



Figur 3.4: Model-View-Controller

Den ursprungliga definitionen av MVC innehöll inte så mycket mer än denna beskrivning. Men det bärande konceptet var så pass starkt att det levt vidare, implementeras, omtolkats och förändrats fram till idag.

När webbsidor utvecklades till att bli webbapplikationer anpassades MVC för att passa denna nya typ av applikationer och det finns idag mängder av ramverk för att bygga webbapplikationer som bygger på MVC [10]. Följande stycke undersöker hur MVC kan implementeras i ramverket ASP.NET.

3.7.4 Model-View-Controller i ASP.NET

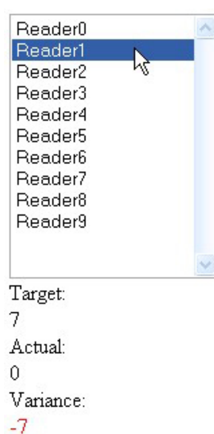
Detta avsnitt beskriver hur MVC kan implementeras i ASP.NET. Syftet är att se hur mönstret kan underlätta separation och fördela ansvar mellan olika klasser för att underlätta återanvändning. Delar av mönstret finns inbyggt i ramverket ASP.NET [22] [3].

Det huvudsakliga konceptet i MVC är att separera presentationslogik från domänlogik [19]. Ramverket ASP.NET har en struktur för att utföra just detta kallad code-behind-strukturen. Detta betyder att presentationskod, exempelvis HTML-taggar och liknande, placeras i en presentationsfil (med ändelsen .aspx) och domänlogik placeras i en separat fil (med ändelsen .aspx.cs). En tagg i presentationsfilen visar vilken fil som innehåller den övriga logiken. Code-behind-strukturen kan sägas vara ett sätt att implementera delar av MVC-mönstret [22]. View utgörs av presentationsfilen, och Controller och Model utgörs av code-behind-filen. För att ytterligare separera Model och Controller kan man skapa

klasser som representerar Model och hanterar all kontakt med databasen [22] [16].

MVC utgår från att tillståndet i View går att avläsa direkt från tillståndet i Model, vilket kan vara ett problematisk antagande [6]. Vilka knappar som har tryckts i ett gränssnitt, eller vilket element som är valt i en lista passar sig inte speciellt bra för att lagra i Model. För att lösa detta kan man använda sig av ett tillägg till MVC kallat Presentation-Model [6]. I Presentation-Model lagras viss information om View. Objekten i View observerar Presentation-Model istället för Model. I ASP.NET finns en variant av detta mönster implicit i ramverket genom den teknik som kallas ViewState. ViewState är i princip en dold lista där värden och fält på kontroller sparas mellan Post-backs.

Nedanstående exempel illustrerar hur ett enkelt användargränssnitt kan implementeras i ASP.NET enligt MVC. Exemplet bygger på exempel presenterade av [22] och [16]. Användargränssnittet som skall implementeras visas i 3.5.



Figur 3.5: Implementationsexempel MVC

Denna enkla sida presenterar ett antal givare. Användaren kan klicka på en vald givare för att se dess värde. Tre textfält presenteras givarens målvärde, dess faktiska värde och differensen mellan dessa två värden. Om givarens värde är under målvärdet presenteras det i röd text. Om givarens värde är lika med målvärdet presenteras det i grön text.

Följande stycken undersöker hur implementationens delar som svarar mot MVC-mönstrets delar. View utgörs av aspx-sidan och innehåller endast html och asp-taggar. Koden för View visas i 3.3.

Kodexempel 3.3: MVC View

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
```

```
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ListBox ID="ListBox1" runat="server"
        OnSelectedIndexChanged="ListBox1_SelectedIndexChanged"
        DataTextField="Name" DataValueField="Reading"
        Height="198px" Width="157px" AutoPostBack="true">
        </asp:ListBox>
        <br />
        Target:<br />
        <asp:Label ID="lbl_target" runat="server"
        Text="Target"></asp:Label><br />
        Actual:<br />
        <asp:Label ID="lbl_actual" runat="server"
        Text="Actual"></asp:Label><br />
        Variance:<br />
        <asp:Label ID="lbl_variance" runat="server"
        Text="Variance"></asp:Label></div>
    </form>
</body>
</html>
```

Controller utgörs av aspx-sidans code-behind och innehåller logiken för att visa en givares värde när givaren valts i listan. Controller innehåller också logiken som avgör vilken färg som resultatet skall presenteras i. Controller anger vilken datakälla som tillhör listboxen i View.

Kodexempel 3.4: MVC Controller

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack) {
        ListBox1.DataSource = Model.GetReaders();
        ListBox1.DataBind();
    }
}

protected void
ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    lbl_target.Text = "7";
    IReaderDTO rdto =
    Model.GetReader(ListBox1.SelectedValue.ToString());
    lbl_actual.Text = rdto.Reading.ToString();
    int variance =
    Convert.ToInt32(lbl_target.Text) -
    Convert.ToInt32(lbl_actual.Text);
    if (variance < 0) {
        lbl_variance.ForeColor = Color.Red;
```

```
    }else{
        lbl_variance.ForeColor = Color.Black;
    }
    lbl_variance.Text = variance.ToString();
}
```

Model utgörs av klassen Model som innehåller statiska metoder för att presentera data samt värdeobjekt, så kallade Data-Transfer-Objects (DTO). Modelklassen ansvarar för att kommunicera med databasen, förpacka datat i DTO:s och passera det till View via databindning.

Att implementera MVC i ASP.NET på detta sätt uppfyller några av de mål som MVC-mönstrets ursprungliga specifikation hade. View består enbart av presentationslogik. Separationen mellan Model och View är god. Model känner inte till vilken View som konsumerar dess data, och en Model kan stödja flera separata View-objekt, vilket är ett av huvudsyftena med MVC [19].

Det finns dock ett antal nackdelar med denna implementationsstrategi. För det första så innebär code-behind-strukturen en stark koppling mellan View och Controller, då varje aspx-sida (View) är knuten till en aspx.cs-sida (Controller). Det blir svårt att återanvända en Controller för olika Views [21].

För det andra så kan det finnas logik som tillhör View, exempelvis att sätta olika färger på olika textstycken eller göra knappar klickbara eller icke klickbara [6]. I den ovan beskrivna implementationsstrategin finns ingen plats för den typen av logik förutom i code-behind-filen, eller i Model. Det kan sluta med en övervuxen Controller som blandar presentationslogik med domänlogik. Ett test för att undersöka om en viss logik är presentationslogik är att föreställa sig att man skriver ett annat användargränssnitt. Om någon kod dupliceras mellan de två användargränssnitten kan den logiken förpassas från View [6].

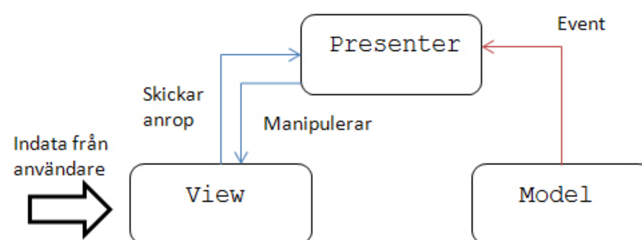
Kopplingen mellan View och Model, så som den beskrivs i den ursprungliga beskrivningen av MVC är otydlig i denna implementation. Den ursprungliga beskrivningen av MVC beskriver hur View observerar Model och frågar Model efter data. Detta förhållande kan till viss del sägas uppfyllas av databindningsfunktionen i ASP.NET. Problemet blir då återigen separationen mellan View och Controller, eftersom koden för databindningen är förlagd till code-behind-filen, som representerar Controller. Gränsen mellan View och Controller blir inte tydlig.

Ovan beskrivna strategi gör det inte tydligt var validering skall ske. Om man använder ASP.NET inbyggda valideringskontroller sker valideringen i View. Enligt den ursprungliga beskrivningen av MVC sker validering lämpligtvis i Controller [5].

Exemplet ovan illustrerar ett antal problem med implementation av MVC i ASP.NET. I skrivande stund är ett nytt ramverk kallat ASP.NET MVC under utveckling, som ämnar adressera problemen med MVC i ASP.NET. Detta faktum, kombinerat med ovanstående problem antyder att ASP.NET inte lämpar sig väl för MVC.

3.8 Model-View-Presenter

För att konstruera ett presentationslager innehållande återanvändbara komponenter i ASP.NET krävs ett mönster som passar bättre i ramverket än vad MVC gör. Model-View-Presenter (MVP) kan vara ett sådant mönster. MVP är en utveckling av MVC. Skillnaden mellan de två mönstren ligger främst i kommunikationen mellan de olika delarna [17]. MVP gör en tydligare separation mellan View och Model. Presenter i MVP fungerar som en mellanhand mellan View och Model och tar hand om all kommunikation dem emellan. Där View i MVC observerar Model och uppdateras vid förändringar sköts allt detta i MVP av Presenter [2].



Figur 3.6: Model-View-Presenter

Implementation av MVP i ASP.NET illustreras nedan. Exemplet bygger på exempel presenterade av [13] och [1]. I detta exempel konstrueras samma användargränssnitt som tidigare implementerades med hjälp av MVC.

View utgörs i denna implementationsstrategi av aspx-sidan och dess code-behind sida. Aspx-sidan är identisk med den som konstruerades i MVC, men i code-behind har ett antal ändringar gjorts.

Kodexempel 3.5: MVP View

```
public partial class _Default : System.Web.UI.Page, IReaderView
{
    private ReaderPresenter presenter;
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
        presenter = new ReaderPresenter(this);
        this.ListBox1.SelectedIndexChanged +=
        delegate{presenter.DisplayReader();};
    }

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack) {
            presenter.Initialize();
```

```
        }
    }

    #region IReaderView Members
    public ListBox ReaderList
    {
        get{return ListBox1; }
    }

    public string Target{
        get{return lbl_target.Text; }
        set{lbl_target.Text = value;
    }

    public string Actual{
        get{return lbl_actual.Text; }
        set{lbl_actual.Text = value; }
    }

    public string Variance
    {
        get{return lbl_variance.Text; }
        set{
            int variance = Convert.ToInt32(value);
            if (variance < 0){
                lbl_variance.ForeColor = Color.Red;
            }else if (variance == 0) {
                lbl_variance.ForeColor = Color.Green;
            }
            else{
                lbl_variance.ForeColor = Color.Black;
            }
            lbl_variance.Text = value;
        }
    }
    #endregion
}
```

All logik som inte handlar om presentation har nu extraherats till en klass av typen ReaderPresenter. ReaderPresenter känner till View genom interfacet IReaderView. Model utgörs fortfarande av samma klasser, men kommunikationen mellan Model och View är nu tydligt strukturerad via Presenter.

Kodexempel 3.6: MVP Presenter

```
public class ReaderPresenter
{
   private IReaderView _view;
```

```
    public ReaderPresenter(IReaderView view)
    {
        _view = view;
    }

    public void Initialize()
    {
        _view.ReaderList.DataSource = Model.GetReaders();
        _view.ReaderList.DataBind();
    }

    public void DisplayReader()
    {
        _view.Target = "7";
        IReaderDTO rdto =
        Model.GetReader(_view.ReaderList.SelectedValue.ToString());
        _view.Actual = rdto.Reading.ToString();
        int variance =
        Convert.ToInt32(_view.Actual) -
        Convert.ToInt32(_view.Target);
        _view.Variance = variance.ToString();
    }
}
```

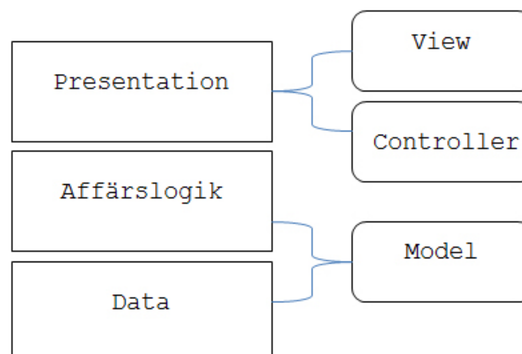
ASP.NET-strukturen lämpar sig naturligt för implementering av MVP [9]. Genom att tydligt separera ut Presenter från code-behind sidan överkommer man några av de tillkortakommanden som diskuterades i implementationen av MVC. Nu finns en tydlig separation av presentationslogik och domänlogik. Denna typ av implementation underlättar också för testdriven utveckling (TDD) [1]. Presenter-klassens metoder lämpar sig väl för enhetstester med valfritt testverktyg. Vissa av problemen som diskuterades i implementationen av MVC finns fortfarande kvar. Var validering sker på bästa sätt är inte tydligt. Ett sätt att hantera validering är att låta Model definiera valideringsreglerna, View visa reglerna för användaren och Presenter ser till att de efterlevs [8].

Nackdelen med denna typ av implementation ligger i den ökade komplexiteten, med fler antal klasser och interface.

3.9 N-skiktade lösningar och MVC/MVP

För att förstå hur MVC/MVP kan stödja skapandet av komponenter som skall verka i presentationslagret i en n-skiktad lösning krävs en förståelse för hur MVC/MVP står i relation till n-skiktade lösningar. MVC/MVP och n-skiktade lösningar står inte i motsatsförhållande till varandra. Båda mönstren kan existera inom samma lösning. [24]. En n-skiktad lösning beskriver separation mellan olika komponenter i olika lager, virtuella eller fysiska. MVC/MVP beskriver ett sätt att separera presentationslogik från domänlogik. Att följa MVC/MVP i en

n-skiktad lösning innebär att View och Controller/Presenter hamnar i presentationslagret, medan Model sprids ut över de övriga lagren. De olika delarnas placering i den n-skiktade lösningens lager visas i 3.7.



Figur 3.7: MVC och N-tier

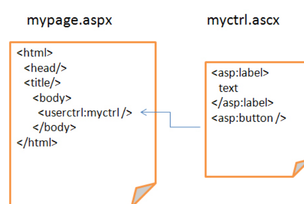
3.10 Tekniker för återanvändning i ASP.NET

De följande avsnitten beskriver de två mest användbara teknikerna som finns i ramverket ASP.NET för att skapa återanvändbara komponenter i presentationslagret.

3.10.1 User Controls

Dynamiska html-sidor i ramverket ASP.NET har ändelsen aspx, och kallas för aspx-sidor. En aspx-sida kan innehålla html-markup eller speciella ASP.NET-kontroller. User Controls är ett sätt att återanvända delar av en aspx-sida. En User Control är i princip ett stycke av en aspx-sida som inhyses på en annan aspx-sida. User-Controls har ändelsen ascx. En User Control länkas in på en aspx-sida genom en tagg som vilken annan ASP.NET-kontroll som helst. Till skillnad från aspx-sidor saknar User Controls HEAD och BODY. Dessa taggar ärvs från sidan som inhyser kontrollen, tillsamman med eventuella stilmallar. En User Control kan använda samma code-behind-struktur som en aspx-sida.

User Controls är ett utmärkt sätt att återanvända delar av ett gränssnitt inom ett projekt. De är lika lätta att utveckla som vanliga aspx-sidor och kräver ingen extra kunskap från utvecklaren. De lämpar sig inte lika väl för att distribuera över flera projekt. En utvecklare som vill använda samma User Control i många olika projekt får klippa och klistra in från ett projekt till ett annat.



Figur 3.8: User Control

3.10.2 Custom Server Controls

Ramverket ASP.NET innehåller mängder av kontroller som skall underlätta för utvecklare att skapa webbapplikationer såsom knappar, drop-down-listor och kalenderkontrollen. Dessa kontroller kallas i ASP.NET för Server Controls. ASP.NET erbjuder också utvecklaren möjligheten att utveckla egna sådana kontroller och dessa kallas då Custom Server Controls. En Custom Server Control kompileras till en DLL och länkas in i ett projekt via bin-katalogen. En Custom Server Control kan mata ut vilken HTML-markup som helst, och kan fungera precis som en User Control. Det går att inkludera såväl JavaScript som stilmallar i en Custom Server Control. Custom Server Controls fungerar också i designläge i Visual Studio och kan dra nytta av den så kallade Drag and Drop funktionen som låter utvecklaren dra och släppa kontroller på en aspx-sida och konfigurera dem utan att skriva en rad kod.

Custom Server Controls lämpar sig utmärkt för att återanvända i flera olika projekt. Distribuering och hantering av ändringar och fixar fungerar enkelt eftersom kontrollen kompileras till en DLL som kan sparas i Global Assembly Cache, en central lagringsplats som .NET-ramverket kan använda sig av. Nackdelen med Custom Server Controls är att de tar längre tid att utveckla än User Controls och att de kräver en del specialkunskaper och en förståelse för kontrollers livscykel.

Kapitel 4

Arbetsprocess

Detta kapitel beskriver hur designmönstren och de tekniker för återanvändning som diskuterats i den teoretiska fördjupningen har använts för att skapa ett antal återanvändbara gränssnittskomponenter till presentationslagret i Acinos Internetkontor.

Den första sektionen beskriver översiktligt hur Acinos Internetkontor ser ut. De följande sektionerna beskriver de olika faserna i arbetet och hur komponenterna utvecklats under arbetets gång.

4.1 Beskrivning av Internetkontoret

Acinos Internetkontor är ett ärendehanteringssystem som vilar på Acinos ramverk för intelligent interaktion över Internet och ramverket ASP.NET. Internetkontorets startsida visas i 4.1.

En typisk sida i Internetkontoret består av en Master-Page som definierar sidans generella utseende, och en eller flera User Controls som innehåller olika delar av gränssnittet.

De flesta aspx-sidor i Internetkontoret ärver från en gemensam klass som innehåller logik för säkerhet, validering och kommunikation med affärslogiklagret. Basklassen som aspx-sidorna ärver från kan ses som en Controller i MVC. User Controls ärver också från en gemensam klass som innehåller en referens till den aspx-sida, och därmed också den Controller, som kontrollen är inhyst på. Kommunikation mellan presentationslagret och affärslogiklagret sker med såkallade värdeobjekt (kallade VO eller DTO). Logik som inte passar i basklassen finns i sidans code-behind klass.

4.2 Arbetsprocess

Ett av de stora problemen med återanvändbarhet är att veta vad som kan återanvändas. I den första fasen av detta projekt identifierades ett antal återanvändbara komponenter via en informell analys av gränssnittet och applikationsdomänen.

Figur 4.1: Internetkontoret

Den informella analysen bestod av att bekanta sig med applikationen och applikationsdomänen, samt tala med utvecklarna av programmet för att försöka få uppslag till vilka delar av presentationslagret som lämpade sig för återanvändning.

Internetkontoret är ett ärendesystem och sökning bland ärenden kan antas förekomma i de flesta instanser av applikationen. Därför valdes Internetkontorets sökfunktion som en första komponent att försöka anpassa för återanvändning. Internetkontorets sökfunktion består av ett antal fält och dropdown-listor där användaren kan söka bland ärenden på olika kriterier. Sökträffar visas i en lista under sökfunktionen. Länkarna ovanför sökkriterierna låter användaren sortera sökträffarna på det angivna kriteriet. Sökfunktionen var implementerad som en User Control. För att öka återanvändbarheten mellan olika projekt implementerades kontrollen nu som en Custom Server Control. Internetkontorets sökfunktion visas i 4.2.

Komponenten fick arbetsnamnet SearchBox, eftersom det är en låda som

Figur 4.2: Internetkontorets sökfunktion

tillåter sökning. Den första versionen av komponenten motsvarade endast View i MVC. Controller/Presenter och Model lämnades åt code-behind och Internetkontorets databaslager. Komponenten skickade vidare events från sina olika knappar som sedan kunde konsumeras av den sida där den inhystes. Återanvändbarheten i denna implementation bestod enbart av HTML-kod, alltså komponentens utseende. Varje gång komponenten användes var det nödvändigt att skriva om beteendet för de olika knapparna och länkarna.

Kodexempel 4.1: SearchBox i aspx-sidan

```
<acino:SearchBox id="SearchBox1" runat="server"
OnSearchClick="Search_Click"/>
```

Kodexempel 4.2: SearchBox i aspx.cs

```
protected void Search_Click(Sender o, Eventargs e)
{
    // sök och bind data till SearchBox
    // controller är basklassen i Internetkontoret
    // en hel del logik utöver nedanstående rader sker här
    SearchBox.SearchResults.DataSource =
    Controller.getErrands(SearchBox.Text);
    SearchBox.SearchResults.DataBind();
}
```

I den andra versionen av SearchBox införlivades även delar av Controller-logiken i komponenten för att öka dess återanvändbarhet. Det mesta av logiken för sökning och sortering i Internetkontoret sker i basklassen och i de underliggande databasklasserna, men en hel del logik återfanns i code-behind filen.

Denna logik flyttades nu in i komponenten. Kommunikationen mellan komponenten och Internetkontorets Controller löstes via interfacet IWebAppController, som enkelt kunde implementeras av Internetkontorets basklass. SearchBox bestod i detta skede av endast en fil, SearchBox.cs, som innehöll såväl rendering av HTML och den flyttade Controller-logiken.

Kodexempel 4.3: SearchBox.cs

```
public class SearchBox: WebControl
{

    private IWebAppController controller;

    public SearchBox(IWebAppController controller)
    {
        this.controller = controller;
    }

    public IWebAppController Controller
    {
        set{controller=value;}
    }
    // metoder för rendering här
    protected override void CreateChildControls()

    // kod som tidigare implementerades i code-behind
    protected void Search_Click(Sender o, Eventargs e)
    {
    // sök och bind data till SearchBox
    // controller är basklassen i Internetkontoret
    // en hel del logik utöver nedanstående rader sker här
    this.SearchResults.DataSource =
    controller.getErrands(this.Text);
    this.SearchResults.DataBind();
      }
}
```

Kodexempel 4.4: SearchBox v2 i aspx-sidan

```
<acino:SearchBox id="SearchBox1" runat="server"/>
```

Kodexempel 4.5: Searchbox v2 i aspx.cs

```
protected override void OnInit()
{
    SearchBox1.Controller = this;
}
```

Denna implementationsstrategi för komponenten innebar ökad återanvändning, eftersom logiken för sortering och sökning inte behövdes skrivas om vid varje användning av komponenten. Nackdelen med denna strategi var att View och Controller nu blandats ihop inuti SearchBox.cs. Återanvändning av endast View eller endast Controller var inte möjlig med denna struktur. För att ändra på detta skapades en tredje och sista version av komponenten som lösgjorde kopplingen mellan View och Controller. Den tredje versionen följer implementationsstrategin för MVP som beskrivits i den teoretiska fördjupningen. Logiken för knapparnas beteende flyttades till en Presenter-klass, SearchBoxPresenter. SearchBoxPresenter ansvarar för kommunikationen med Internetkontorets Controller. Kopplingen mellan SearchBox och SearchBoxPresenter sker med interface av typen ISearchBox och ISearchBoxPresenter. I enlighet med MVP så finns ingen koppling mellan View och Model, all kommunikation sker via Presenter.

Denna implementationsstrategi möjliggör att en View kan återanvändas av olika implementationer av Presenter och Model. På samma sätt kan en Presenter återanvändas av olika implementationer av View. För att utöka SearchBox beteende infogades även ett antal JavaScript i komponenten. För att förbättra dess utseende infogades en stilmall. En djupare diskussion om JavaScript och stilmallar följer i det avslutande kapitlet.

Ytterligare en komponent implementerades enligt ovanstående strategi. Denna komponent döptes till SelectionBox och erbjuder ett gränssnitt för att välja ett antal ärenden ur en lista. Slutligen implementerades en enklare komponent, kallad ValidatingTextBox, som bara innehåller en View i MVP-mönstret. En noggrann beskrivning av dessa komponenter samt en mer ingående beskrivning av SearchBox återfinns i resultatkapitlet.

Kapitel 5

Resultat

Detta kapitel innehåller en beskrivning av de gränssnittskomponenter som implementerats.

5.1 ValidatingTextBox

ValidatingTextBox är den enklaste av de komponenter som implementerats. Syftet med denna komponent är att förenkla validering av textfält i Internetkontoret. Kontrollen har ingen explicit Presenter knuten till sig. Den består av ett textfält och ett antal alternativ som anger vilka valideringsregler som skall appliceras på textfältet. Kontrollen använder ASP.NET:s inbyggda valideringskontroller för alla valideringstyper förutom Pnr (Personnummer). Pnr valideras med en valideringskontroll som skapats specifikt för denna kontroll, PnrValidator.

5.2 SelectionBox

SelectionBox består av två listor. Kontrollen låter användaren välja flera alternativ från listan till vänster, som då visas i listan till höger. En knapp låter användaren välja samtliga alternativ i listan. SelectionBox förutsätter att det data som skall bindas är av typen IEligUserVO. Detta interface innehåller de fält som binds i listan som text och värde.

SelectionBox kopplas till en Presenter av typen ISelectionBoxPresenter. I Internetkontoret kan klassen JOBSelectionPresenter användas. JOBSelectionPresenter innehåller logiken som flyttar ett element från en lista till en annan. Den innehar även logik för att returnera en lista med samtliga element som valts. JOBSelectionPresenter innehar en referens till Internetkontorets Controller via interfacet IWebAppController.

5.3 SearchBox

SearchBox är en komponent som erbjuder ett gränssnitt för sökning bland ärenden. Den innehåller ett antal fält och drop-down-listor för att ange sökkriterier samt knappar för att sortera och söka på dessa kriterier. SearchBox innehåller en lista av typen GridView som presenterar det data som sökningen genererar. SearchBox förutsätter att de objekt som skall presenteras i träfflistan är av typen IErrandVO. Detta interface innehåller de fält som presenteras i listan.

SearchBox kopplas till en Presenter av typen ISearchBoxPresenter. I Internetkontoret kan klassen JOBSearchBoxPresenter användas. JOBSearchBoxPresenter använder sig av Internetkontorets affärslogiklager via ett interface för att erbjuda funktionalitet såsom sökning, sortering av träfflistan, samt handläggning av ärenden i träfflistan.

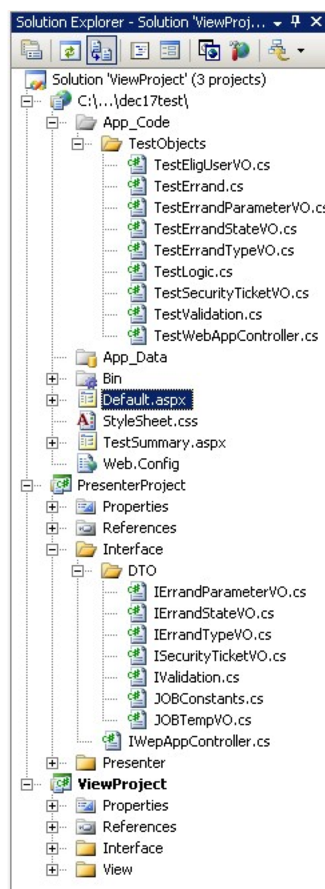
Utseendet på SearchBox bestäms av en stilmall som är inbäddad i kontrollen. Önskas ett annat utseende på kontrollen kan stilmallen omdefinieras genom att använda !important.

SearchBox innehåller även två JavaScript som underlättar användningen av kontrollen. JavaScripten tillåter en användare av kontrollen att markera samtliga ärenden i träfflista, se en detaljbild av ett visst ärende, samt kopiera texten från ett textfält till ett annat.

5.4 Användning i Acinos ramverk och Internetkontoret

Detta avsnitt beskriver hur komponenter som implementerats enligt ovanstående kan användas i Acinos ramverk och Internetkontoret för att öka återanvändbarheten. Presentationslagret organiseras i tre olika projekt. ViewProject innehåller WebControls samt interface som beskriver dessa kontroller, och dess Presenters. PresenterProject innehåller referenser till ViewProject. PresenterProject innehåller implementationer av interfacen i ViewProject. Om det skapas nya kontroller, Views, i WebProject kan de Presenters som tillhör dem med fördel läggas i detta projekt. 5.1 illustrerar hur organisationen av presentationslagret kan se ut.

Genom att organisera presentationslagret på detta sätt kan en View i ViewProject lätt återanvändas inom samma project, med olika eller samma Presenter. Eftersom View-klasserna är implementerade som Custom Server Controls kan de lätt återanvändas i olika projekt. När nya komponenter utvecklas kan de adderas till ViewProject och på lång sikt kan ett bibliotek av användbara gränssnittskomponenter byggas upp. Separeringen av View och Presenter gör att beteendet hos två komponenter med samma utseende kan skilja sig vid behov. På samma sätt kan två komponenter med olika utseende dela samma beteende. Uppdelningen av Presenterklasserna underlättar också vid enhetstestning. Om utvecklaren bedömer att extraarbetet med att skapa en View-klass i ViewProject inte är lönsamt kan View-klasser också skapas i WebProject som User Controls,



Figur 5.1: Presentationslagret

eller vanliga aspx-sidor.

Denna uppdelning innebär förbättrade möjligheter för återanvändning av presentationslagret. Kostnaden är ökad komplexitet och beroenden mellan de olika projekten. Att utveckla en Custom Server Control i ett separat projekt med tillhörande Presenter-interface kräver med arbete än att bara skapa en User Control direkt i Webprojektet. Det kräver en avvägning och en bedömning från utvecklaren när det kan vara värt att addera den ökade komplexiteten.

Kapitel 6

Sammanfattning

Detta kapitel sammanfattar arbetet och tar upp begränsningar hos de implementerade komponenterna. En kort diskussion gällande framtida arbete avslutar kapitlet.

Återanvändbarhet är ett mycket svårt ämne att ta sig an, och det är ofta svårt att på förhand veta vad som kommer att kunna återanvändas och inte. Det är därför svårt att säga om de implementerade komponenterna är återanvändbara eller inte. Organisationen som föreslagits innebär förbättrade möjligheter för återanvändning av presentationslagret. Kostnaden är ökad komplexitet och beroenden mellan de olika projekten. Att utveckla en Custom Server Control i ett separat projekt med tillhörande Presenter-interface kräver mer arbete än att bara skapa en User Control direkt i Webprojektet. Det kräver en avvägning och en bedömning från utvecklaren när det kan vara värt att addera den ökade komplexiteten. Även om många kontroller inte lämpar sig bra för att skapas som Custom Server Controls så finns det fortfarande ett värde i separationen mellan View, Presenter och Model.

6.1 Begränsningar

De implementerade komponenterna lämnar en del att önska när det gäller grafisk konfigurerbarhet. SearchBox tillåter att användaren ändrar namn på fält och knappar, men erbjuder inget stöd för att exempelvis ändra antalet sökkriterier. En sökruta med variabelt antal sökkriterer som kan konfigureras vid användning skulle kunna vara en utökning av komponenten.

JavaScript och stilmallar ligger inbäddade i komponenterna och förhöjer beteendet och utseendet. För att ändra utseende på komponenterna måste utvecklaren känna till klassnamnet på de html-element som bygger upp komponenten och omdefiniera deras stil genom att använda sig av nyckelordet !important i en stilmall. Det hade varit önskvärt om stilegenskaper kunde konfigureras i designläge, på samma sätt som namn på knappar och sökkriterier.

Med nuvarande implementation följer en stilmall och en JavaScriptfil med

för varje komponent som används på en sida. Det hade varit ett bra alternativ att istället förpacka alla stilmallar och JavaScript i en separat komponent.

Användning av komponenterna kräver en förståelse för hur de är uppbyggda, det vill säga vilka namn i Presenter-interfacet som svarar mot vilka fält i View-klassen. Dokumentation och exempel på hur komponenterna är avsedda att användas kan göras avsevärt bättre.

6.2 Framtida arbete

Framtida arbete skulle kunna implementera den föreslagna strukturen i Internetkontoret. Framtida arbete skulle även kunna undersöka andra komponenter som lämpar sig att implementera som Custom Server Controls och bli en del av Acinos gränssnittsbibliotek. En sådan komponent skulle exempelvis kunna vara en Login-kontroll. Denna typ av komponent förekommer i princip alla applikationer och skulle lämpa sig mycket väl för återanvändning.

Litteraturstudien visar en brist på teoretiska studier rörande mätbarhet av grafiska gränssnitts återanvändbarhet, samt tekniker och teorier kring identifikation av återanvändbara delar av gränssnittet. Fortsatta studier inom detta område skulle vara till stor hjälp vid utveckling av återanvändbara gränssnitt.

Sammanfattningsvis kan sägas att målet att öka återanvändbarheten av presentationslagret i Acinos ramverk och Internetkontoret inte uppnåtts eftersom endast ett fåtal komponenter implementerats. Den föreslagna strategin för organisation av presentationslagret kan förhoppningvis underlätta för återanvändning i framtiden.

Kapitel 7

Tack

Jag vill tacka min handledare Jurgen Burstler och Kent Sundberg för all hjälp jag fått. Jag vill tacka Hannes Kock för all teknisk vägledning. Jag vill tacka min fru för att hon stått ut.

Referenser

[1] Jean-Paul Boodhoo. Design patterns: Model view presenter. *MSDN Magazine*, 21(9), 2006. http://msdn.microsoft.com/msdnmag/issues/06/08/DesignPatterns/default.aspx#contents,accessed 2008-04-12.

[2] Andy Bower and Blair McGlashan. Twisting the triad: The evolution of the dolphin smalltalk mvp application framework. In *Tutorial Paper for ESUG 2000*, page {}. {}, 2000.

[3] Miguel Castro. Skin your web apps using mvc. http://www.devx.com/codemag/Article/27815/0/page/4, accessed 2008-03-13.

[4] James W. Cooper. *C-sharp Design Patterns: A Tutorial*. Addison Wesley Professional, Boston, MA, USA, 2003.

[5] Arnold Doray. *Beginning Apache Struts: From Novice to Professional*. APRESS, 2855 Telegraph Avenue, Suite 600 Berkeley, CA 94705, 2 2006.

[6] Martin Fowler. Gui architectures. http://www.martinfowler.com/eaaDev/uiArchs.html, accessed 2008-04-13.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[8] Jeff Handley. Extended mvp pattern - domain validation, 1 2008. http://blog.jeffhandley.com/archive/2008/01/15/extended-mvp-pattern---domain-validation.aspx,accessed 2008-03-12.

[9] Alex Homer. Design patterns for asp.net developers. *devx.com*, 2007. http://www.devx.com/dotnet/Article/33695/0/page/3, accessed 2008-04-13.

[10] Mehdi Jazayeri. Some trends in web application development. In *FOSE '07: 2007 Future of Software Engineering*, pages 199–213, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Ralph E. Johnson. Components, frameworks, patterns. In *SSR '97: Proceedings of the 1997 symposium on Software reusability*, pages 10–17, New York, NY, USA, 1997. ACM.

[12] Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.

[13] Nikola Malovic. Model view presenter (mvp) pattern, 2006. `http://blog.vuscode.com/malovicn/archive/2006/10/10/Model-View-Presenter-_2800_MVP_2900_-pattern.aspx`, accessed 2008-04-12.

[14] Scott Elbandit mfl. Decorator, 3 2008. `http://wiki.asp.net/page.aspx/317/decorator/`,accessed 2008-03-27.

[15] msdn. Implementing observer in .net, 3 2008. `http://msdn2.microsoft.com/en-us/library/ms998543.aspx`,accessed 2008-03-27.

[16] Nimesh Panchal. Implementing mvc design pattern in .net, 2003. `http://www.c-sharpcorner.com/UploadFile/napanchal/MVCDesign12052005035152AM/MVCDesign.aspx`, accessed 2008-04-13.

[17] Mike Potel. Mvp: Model view presenter. 1997. `http://www.wildcrest.com/Potel/Portfolio/mvp.pdf`, accessed 2008-04-13.

[18] Jeffrey S. Poulin. *Measuring software reuse: principles, practices, and economic models.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[19] Trygve Reenskaug. Models - views - controllers. technical note, xerox parc, December 1979.

[20] Dirk Riehle. Composite design patterns. *SIGPLAN Not.*, 32(10):218–228, 1997.

[21] Rick Strahl. What's ailing asp.net web forms, 2007. `http://www.west-wind.com/weblog/posts/198731.aspx`, accessed 2008-04-12.

[22] Various. *Enterprise Solution Patterns Using Microsoft .Net: Version 2.0 : Patterns & Practices.* Microsoft Press, Redmond, WA, USA, 2003.

[23] Uwe Zdun and Paris Avgeriou. Modeling architectural patterns using architectural primitives. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, pages 133–146, New York, NY, USA, 2005. ACM.

[24] Jianyun Zhou and Tor Stålhane. A component-based reference model for web-based systems. In M. H. Hamza, editor, *IASTED Conf. on Software Engineering and Applications*, pages 653–658. IASTED/ACTA Press, 2004.