CHALMERS

Automatiserad behörighets- och säkerhetsmodellering för Microsoft-plattformar

Examensarbete inom datavetenskap

Ola Ekstrand

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Göteborg, Sweden, April 2009

Automatiserad behörighets- och säkerhetsmodellering för Microsoft-plattformar

Ola Ekstrand

Examiner: Sven-Arne Andreasson

Department of Computer Science and Engineering

Chalmers University of Technology

SE-412 96 Göteborg

Sweden

Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering

Sammanfattning

I dagens företag och organisationer med flera tusen datoranvändare ställs stora krav på IT-avdelningarna som administrerar detta. Spintop har med hjälp av Microsoft Zero Touch Provisioning utvecklat en portal där vanliga processer har automatiserats och administrationen av konton sker på ett standardiserat sätt. Denna lösning är inte ekonomiskt lönsam för företag med en mindre organisation, syftet med rapporten är att visa ett alternativ till denna lösning som är ekonomiskt försvarbar för företag med färre än 1000 anställda.

Rapporten visar att det med hjälp av .NET-framework går att bygga en portal som erbjuder samma grad av säkerhet, funktion och användarvänlighet som Zero Touch Provisioning till en avsevärt mindre kostnad.

Innehåll

Introduktion

Stora organisationer och företag med många anställda ställer stora krav på administrationen av deras nätverk. Att underhålla tusentals användare och deras applikationer blir en stor belastning på IT-avdelningen. För att underlätta denna administration har Microsoft utvecklat en portaltjänst som tillhandahåller ett gränssnitt där administratörerna arbetar indirekt mot de bakomliggande systemen, på detta sätt kan man lätt standardisera hur information skall lagras samt att många processer kan automatiseras. Denna lösning är dock inte lönsam för företag med mindre organisation då kostnader för licenser är relativt stora och få använder det.

Spintop Netsolution AB

Spintop är ett företag beläget i Göteborg, Stockholm och Malmö med cirka 30 anställda. Deras affärsidé bygger på att förenkla och effektivisera hanteringen av PC-plattformar i större organisationer. Detta inkluderar administration av programvarudistribution, behörigheter, användarkonton, lösenord m.m. Deras lösningar bygger alla på Microsoftprogramvara som de i vissa fall har vidareutvecklat. Anställda på Spintop jobbar på konsultbasis och arbetar ofta ute hos kunder och utvecklar anpassningar av Spintops existerande lösningar för att kunna tillgodose kundens behov.

Bakgrund

Microsoft ZTP

ZTP, Zero Touch Provisioning, heter portalen Microsoft har utvecklat för att kunna automatisera och effektivisera PC- och användaradministration. ZTP bygger på Microsoft SharePoint, SharePoint är en portal med tjänster som fildelning, versionshantering av dokument och bygger på ASP.NET. En stor nackdel med ZTP är att de krav som ställs på systemets struktur ofta överrensstämmer dåligt med den uppbyggnaden kunderna har, ZTP har dessutom väldigt få tjänster i sin grundkonfiguration och kan i princip inte användas till något användbart innan den har vidareutvecklats. Dessa problem har Spintop tagit fasta på och erbjuder en utvecklad version av ZTP.

Figur 1 ZTP

Bilden ovan visar en version av deras portal för en specifik kund. I de flesta projekt de genomfört har de gjort en anpassad version av ZTP till varje kund, det har inneburit att man inte tagit vara på den utvecklingen man gjort i tidigare projekt utan fått bygga om produkten relativt mycket i varje nytt projekt. Däremot strävar de efter att göra en standardiserad produkt med ett antal krav på organisationen i det existerande systemet. Den anpassningen de har gjort handlar mycket om att tillgodose olika behov, ofta handlar det om olika tjänster som skall kunna genomföras via portalen. Från att ha gått från att sälja tjänster försöker de gå mot

att sälja en produkt, det största hindret för att detta skall gå att genomföra är att etablerade organisationer med färdiga hierarkier inte är så benägna att ändra på sin redan färdiga struktur.

Automatiserad behörighets- och säkerhetsmodell

Spintop har en produkt som lämpar sig väl till större organisationer med fler än 1000 användare av systemet, för mindre företag eller organisationer lönar sig inte produkten då den ställer krav på det existerande systemet i form av andra produkter från Microsoft. För att kunna göra en lönsam produkt även för mindre företag har de valt att undersöka hurvida det går att konstruera en likvärdig portal till ZTP som inte har dessa krav. Att undersöka om detta går, samt att göra en prototyp har varit mitt examensarbete. Då ZTP består av mer än gränssnittet mot användaren och en säkerhetesmodell har Joel Sanderi från IT-programmet på Chalmers även gjort sitt examensarbete inom företaget. Sanderi har visat att det går att skapa en flödeskontrollerad motor genom Windows Workflow Foundation, denna är sedan tänkt att arbeta i bakgrunden och hantera de flöden som initieras av portalen.

Examensarbetet har i stort bestått av två delar, gränssnitt och säkerhet. Att skapa ett gränssnitt motsvarande ZTP med hjälp av ASP.NET betyder att man frångår Microsoft SharePoint som bas, vilket i sin tur medför att en liknande bas måste byggas, säkerhetsmodellen för den nya lösningen måste följa samma riktlinjer som den för SharePoint.

Förstudie

Kravspecifikationen på den lösningen som skulle byggas var inte så omfattande, men dock fanns önskemål om att lösningen skulle likna Spintops ZTP i utseende och uppbyggnad. Efter en ganska kort tids analys av de alternativ som fanns på marknaden bestämde jag mig för att bygga min lösning på ASP.NET och använda Microsofts existerande lösning för autentisering och behörighet. Fördelarna jag såg med detta var många:

- Smidig integration med andra Microsoftprodukter
- Lättare för andra utvecklare på företaget att sätta sig in i koden
- Gränssnittet kan byggas upp på liknande sätt med s.k. WebParts
- Tjänster från ZTP kan modifieras för att användas direkt i den nya portalen
- Mycket arbete har gjorts på området

Alternativen till att bygga portalen i ASP.net var dessutom inte relevanta då de anställda på Spintop nästan uteslutande använde ASP.NET, att använda ett annat ramverk skulle innebära att jag i princip inte kunde få någon hjälp.

Design

Gränssnitt

För att kunna erbjuda användare av den nya portalen, i fortsättningen kallad OnePoint, samma möjligheter i gränssnittet som användare av SharePoint byggde jag upp sidan med så kallade WebParts.

WebParts

Tanken bakom WebParts är att användare ska kunna förändra webbplatsers innehåll, utseende och beteende genom webbläsaren. Detta realiseras genom att användaren kan byta en rad olika parametrar för varje WebPart, som till exempel teckensnitt och bakgrundsfärg, användaren kan dessutom flytta en WebPart mellan olika så kallade WebPart-zoner samt lägga till nya WebPart till en zon. Hela konceptet bygger på att användaren är inloggad mot en server som sparar dennes inställningar till en SQL-databas, då användaren loggar in mot portalen laddas de personliga inställningarna från databasen. Inställningar för sidan kan också göras för alla användare genom en så kallad ”shared view”.

User controls

User controls är kontroller där innehållet på portalen byggs upp, en user control består av en ASP.NET-sida och en kodfil, kodfilen innehåller logiken för sidan, till exempel vad som ska hända om användaren klickar på en knapp eller markerar ett val i en listbox. I ASP.NET-sidan registreras de kontroller, knappar textboxar o.s.v., som skall visas på sidan och deras attribut såsom position, utseende och vilka events som skall triggas. Tjänsterna i det som är portalen består oftast av en user control, mycket av den utvecklingen av nya tjänster som sker består av att utveckla en user control för ändamålet.

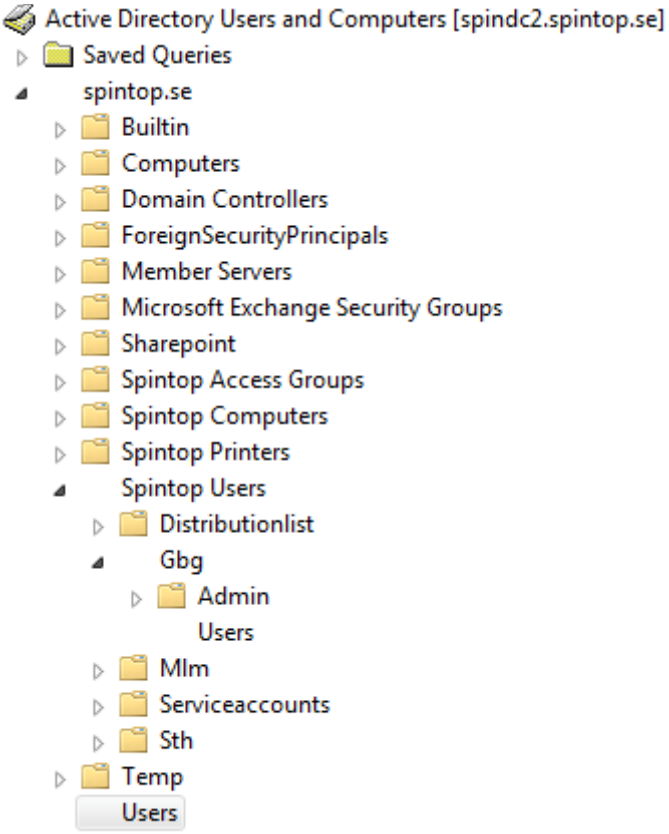
En user control kan även innehålla en annan user control, detta kan vara användbart om man har en tjänst som man vill använda på flera sidor, t.ex att söka upp en användares information från en databas.

Säkerhet

Konceptet med en portal där anställda kan gå in och utföra personliga tjänster bygger på att identiteten de har när de använder portalen är mappad till den identiteten de har i sitt företags nätverk, i de flesta fall används Active Directory. För att logga in på portalen krävs alltså ett användarobjekt i Active Directory.

Active Directory

Active Directory är Microsofts system för att hantera grupper och användare, för varje användare finns information om denne, till exempel e-post, telefonnummer och adress. Rättigheter kan dessutom sättas på användare eller grupper. Strukturen i Active Directory är en trädstruktur enligt nedan.



Figur 2 Active Directory

Noderna i trädet ovan är organisationsenheter, huvudsyftet med dessa är att organisera användare och grupper likt ett filsystem, för att ändra behörighet hos en användare tilldelar man denne gruppmedlemskap istället. Användarobjekt och grupper ligger under organisationsenheterna. En fördel med organisationsenheterna är att man kan bestämma vem som ska kunna redigera användarna under en viss enhet, i exemplet ovan kan man tänka sig att de som är ansvariga för skrivarna bara kan ändra på enheten ”Spintop Printers” medan administratörer för användare kan ändra innehållet under ”Spintop Users”.

Rollhantering

Med Active Directory kan man hantera grupper och användare, men för att enkelt kunna ge olika användare tillgång till olika resurser behövs ett abstraktionslager

mellan Active Directory och portalen. Microsofts lösning på det är rollhantering, vilket administreras genom verktyget Authorization Manager. Med Authorization Manager kan man bygga upp avancerade rollhanteringssystem där roller kan tilldelas rättigheter att utföra vissa uppgifter, i min lösning har jag använt det för att lättare kunna dela in portalens användare i grupper. Ett exempel på en roll kan vara administratör, fördelen med att använda roller istället för grupper i Active Directory är att ASP.net erbjuder bättre stöd för rollhantering än för att kontrollera grupptillhörighet.

Rollernas nödvändighet är tydlig då de tjänster som går att utföra i portalen är tänkta till olika användare, med roller kan innehållet lätt anpassas efter den typen av användare som är inloggad.

Säkra portalen

För att sätta upp en säker miljö är det första steget att bestämma vilken miljö som applikationen är tänkt att användas i. För företag som använder portaler är följande ofta sant:

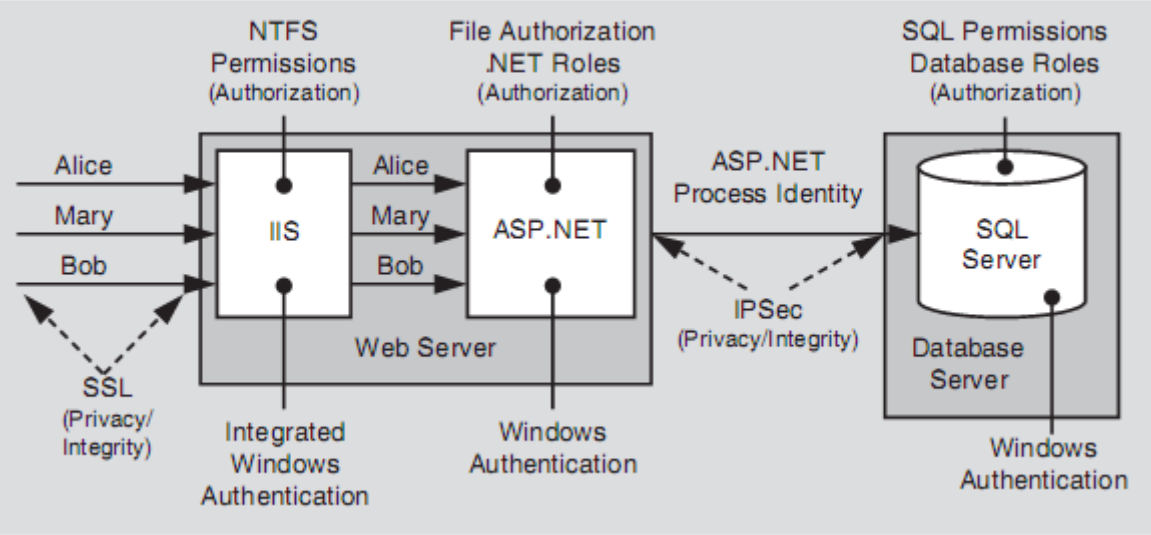
- Klienter använder Internet Explorer
- Användarkonton ligger i Active Directory
- Portalen hanterar känslig användardata
- Bara autentiserade användare skall ha tillgång till portalen
- Bakomliggande databaser litar på portalen att användare är autentiserade, portalen gör anrop åt användaren.

Med den här miljön kan man sen sätta upp en säkerhetslösning. Flera olika element ingår i begreppet säkerhet, vad jag tycker är viktigt är att särskilja på begreppen autentisering och rättigheter (engelskans authorization), med autentisering menas i det här fallet huruvida en användare kan logga in mot portalen och med rättigheter vilket innehåll denne har tillgång till.

Autentisering

För att erhålla en säker autentisering är Microsofts rekommendation att man använder stark autentisering i webbservern genom att använda sig av integrerad Windows-autentisering i Microsofts webbserver Internet Information Services (IIS). I ASP.net rekommenderas att man inte använder sig av personifiering utan även där Windows-autentisering, vilket också gäller för att säkra kommunikationen med databaserna. Databasen litar sen på ASP.net-processen att göra anrop. Som man ser

på bilden nedan så görs allt utifrån användarens kontext förutom då man vill kommunicera med sql-servern då man byter kontext till asp.net-kontots arbetsprocess.



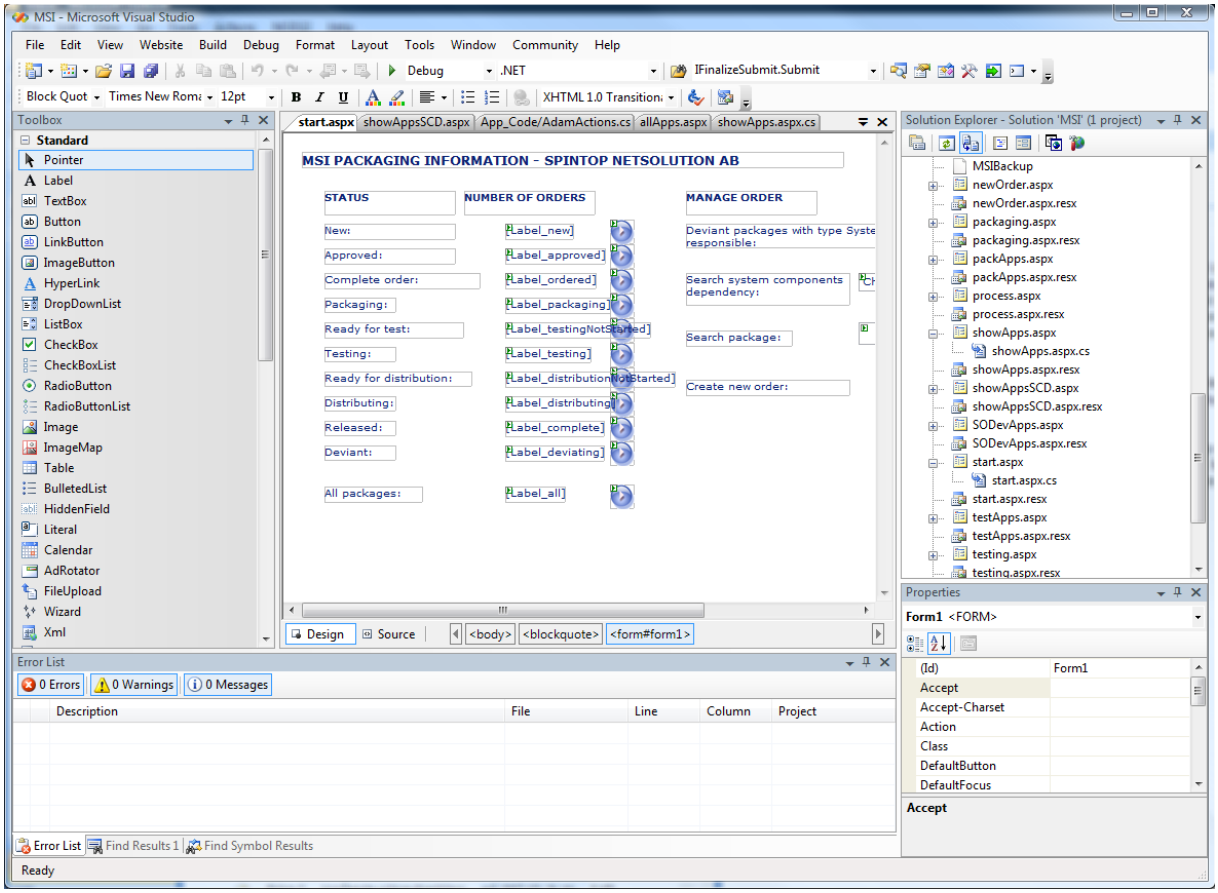
Figur 3 ASP.net autentisering

Arbetssätt

Utvecklingen av OnePoint har skett på Spintops kontor i Göteborg. På kontoret har jag haft tillgång till hjälp från anställda på Spintop i stort sett hela tiden som har varit till stor hjälp. Vad jag saknade var tillgång till källkoden till deras portal ZTP, detta hade kunnat spara en del tid då vissa problem jag stötte på redan var lösta i ZTP.

Utvecklingsmiljö

Projektet utvecklades i ASP.net med C# som programmeringsspråk, alternativet är Visual Baisc, men då mina erfarenheter är större i Java som har stora likheter med C# valde jag detta. Då jag arbetar nästan uteslutande i en Microsoftmiljö har jag valt att använda Microsoft Visual Studio som utvecklingsmiljö, fördelarna är många då man kan konfigurera miljön för till exempel webbutveckling.



Figur 4 Visual Studio

Miljön har flera vyer, till vänster finns en toolbox där standarduppsättningen av kontroller finns, man bygger enkelt sin sida genom att dra in kontroller. Till höger visas projektets klasser och sidor.

Ett problem med att utveckla avancerade webbapplikationer som kräver inloggning och autentisering är att sätta upp en miljö där du kan testa din lösning. Vad som underlättar stort med Visual Studio är att du kan testköra din webbplats utan att behöva sätta upp en demomiljö, Visual Studio förser dig med allt som behövs för att kunna testa din webbplats genom ASP.net development server. Förr eller senare måste du dock ändå flytta lösningen till en riktig miljö för att kunna testa under riktiga förutsättningar med krav på säkerhet.

Webbutveckling

Webbutveckling skiljer sig från traditionell programmering på ett antal olika sätt, det mest påtagliga är att variabler och data inte sparas mellan så kallade postbacks, alltså när sidan laddas om. Detta har en stor inverkan på hur man bygger upp webbapplikationer, vid en första inblick kan det se ut som man inte kan bygga upp någon avancerad logik, detta är dock inte sant då det finns hjälpmedel.

ViewState

ViewState är en array där information kan sparas undan mellan postbacks, denna är unik för varje kontroll eller sida, detta betyder att du inte kan spara information mellan sidor, utan bara för varje sida. I ViewState kan bara objekt som implementerar interfacet Serializable, ett exempel på objekt som inte är serialiserbart är HashTable.

Session

För att kunna spara information mellan olika sidor, till exempel användarinformation kan man använda Session som fungerar på samma sätt som ViewState med skillnaden att den är unik för varje session, alltså en per användare. I Session kan man dessutom spara objekt som inte är serialiserbara, dock bör man inte göra detta i för stor utsträckning då de kontroller som inte är serialiserbara kan sparas på utrymmeskrävande sätt.

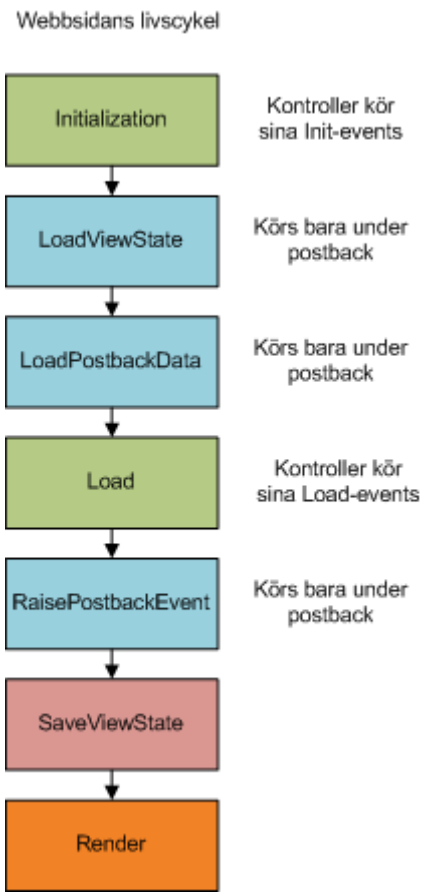
ViewState och SessionState går bra att använda för att spara information mellan postbacks, man behöver dock inte göra detta för kontroller i allmänhet, de flesta kontroller sparar sin information i viewstaten automatiskt, detta underlättar mycket då du annars skulle behöva spara alla kontrollers värden och ladda varje postback. Gör man en egen kontroll kan det vara en god idé att spara undan information i viewstaten. Man bör också tänka till innan man använder dessa metoder då de

belastar systemet på olika sätt, ViewState genom att den skickas med i varje anrop mellan klient och server. Session belastar servern då den sparas undan på servern för varje användare, i fall med tusentals användare kan detta lätt få ödesdigra konsekvenser.

Vad som också skiljer webutveckling från regulär utveckling är att man ofta inte instansierar klasser eller sidor själv, detta görs av ASP-motorn då en sida laddas, ett problem som ofta uppstår då är att man inte kan kommunicera mellan klasser eller sidor, mellan WebParts kan man lösa detta genom att skapa connections. Mellan olika tjänster kan man använda olika metoder för att kommunicera, jag har valt att använda interface i stor utsträckning, detta för att klasserna inte ligger i samma namespace. Ett exempel, klass A är en UserControl som behöver aktuellt datum från klass B, klass B implementerar då interfacet IDate som har en metod, GetCurrentDate, som returnerar dagens datum. För att klass A ska komma åt klass B använder jag då funktionen Page.FindControl som söker igenom underliggande kontroller efter ett ID, som jag sedan kastar (eng. cast) till IDate.

Webbsidans livscykel

Varje ASP.net-sida går igenom en livscykel, den är viktig att känna till och förstå för att kunna programmera säkra och stabila lösningar, till exempel kan man inte lägga till en zon efter initialization-steget, detta skulle resultera i ett felmeddelande när en användare kör sidan. Nedan följer en sammanfattad bild av sidan livscykel.



Figur 5 Sidcykel

Som namnet säger så initieras kontroller i Initialization-steget, som med alla steg kan man bestämma vad som ska hänga i detta steg genom att skriva över OnInit-metoden. Nästa steg körs bara då sidan laddas om, i detta steg får kontroller sina värden genom att de laddas från ViewStaten, om man vill populera kontrollerna på ett annat sätt än standard kan man skriva över metoden. I LoadPostbackData söks sidan igenom efter kontroller som implementerar interfacet IPostBackDataHandler och populerar dessa med postback-data. Load-stadiet förser kontrollerna med klientsideinställningar, till exempel bredd och höjd. Efter Load-stadiet söks kontrollerna på sidan igenom i RaisePostBackEvent-steget, om en kontroll har ändrats sen sist då RaisePostDataChanged kastas. SaveViewState gör vad den säger och sparar kontrollernas data i viewstaten, till sist körs Render som ritar upp kontrollerna på sidan.

Implementation

Att bygga en prototyp av en portal har bestått av tre delprojekt, implementera rollbaserat innehåll, bygga upp en motsvarighet till SharePoint i ASP.net och till sist att bygga upp en virtuell prototypdomän.

OnePoint

Hierarki

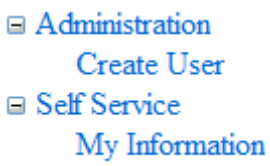
Överst i hierarkin finns en mastersida, i mastersidan specificeras gemensamma egenskaper för alla sidor, ofta vill man ha ett gemensamt utseende för hela webbplatsen, istället för att ange detta på varje sida gör man det i mastersidan som sedan gäller för alla undersidor.

Under mastersidan ligger huvudsidan i portalen, här anges vilka WebPart-zoner sidan ska bestå av och i kodfilen till den här sidan anger man också allt som ska inträffa då användaren loggat in. Om kommunikation ska kunna ske mellan olika web parts måste man ange det här också, hur det går till förklaras senare.

Huvudsidan består av tre zoner, grundtanken är att den vänstra zonen ska visa navigationsträdet, den mittersta innehållet i den för tillfället valda tjänsten och den högra egenskaper för en WebPart om användaren valt det.

Rollhantering

För att visa en prototyp av rollbaserat innehåll byggde jag ett enkelt gränssnitt med ett navigationsträd där användaren kunde se de noder som den rollen han hade tillät. Jag använde mig av en trädstruktur, se bild.

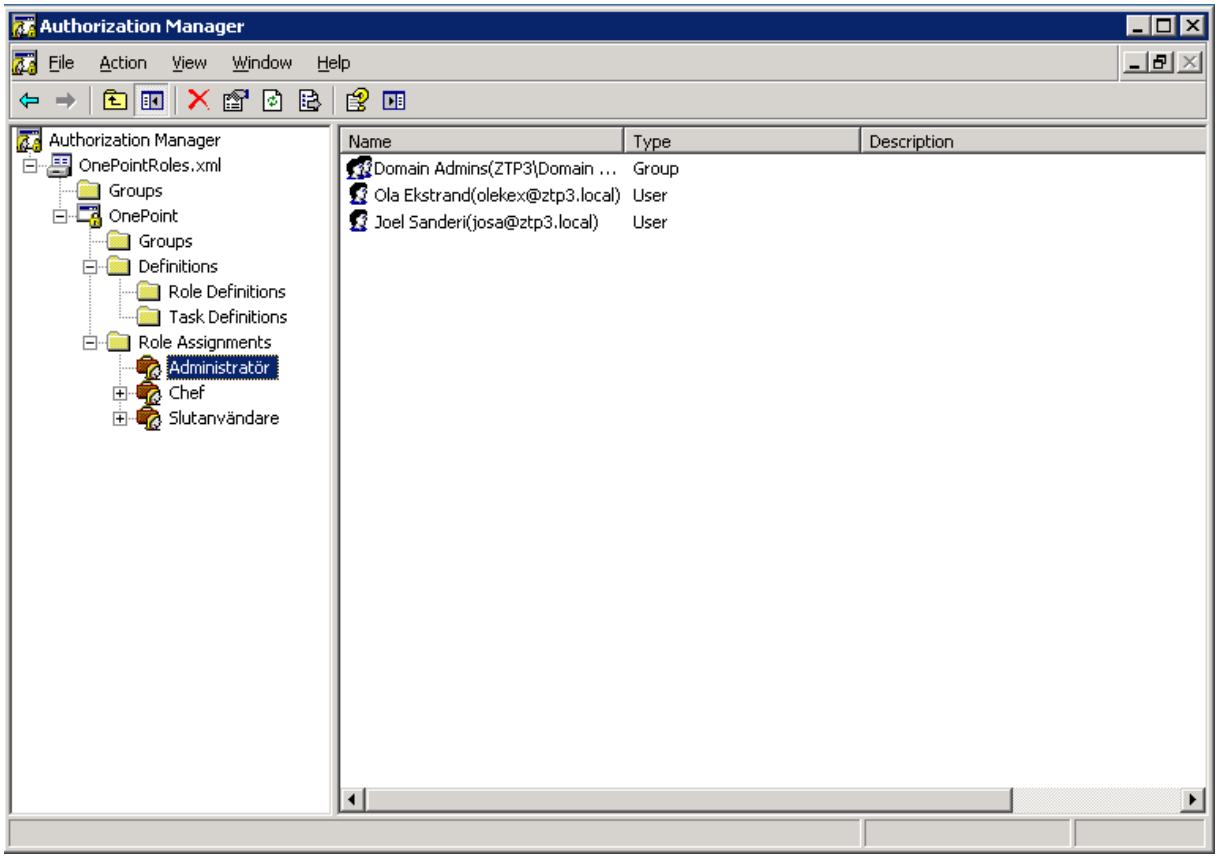


Figur 6 TreeView

Själva implementationen i ASP.net bygger på en XML-fil där jag sparar data om vilka roller som har rättigheter till vilka noder, navigationsträdet byggs sedan upp dynamiskt när sidan laddas.

Authorization Manager

Att bygga upp en så kallad Authorization Store består av att definiera vilka roller som finns och vilka grupper eller användare som tillhör dessa, eventuellt även definiera vad en roll har tillåtelse att utföra. Jag valde dock att nöja mig med ett abstraktionslager och definierade upp en Authorization Store enligt nedan.



Här har jag definierat rollerna administratör, chef och slutanvändare. För varje roll anger jag sedan vilka användare eller grupper från Active Directory som skall ingå i varje roll. Väljer man att använda grupper som abstraktionslager mellan användarobjekt och roll har man än fler möjligheter, nackdelen som jag ser det är att översikten blir sämre och större risk att användare hamnar i fel roller uppstår. För att kunna använda roller i utvecklingsmiljön finns en adapter från Microsoft som lätt ger tillgång till operationer som autentisering, listning av roller o.s.v.

De vägval jag gjorde i den här fasen var att säga att en tjänst i navigationsträdet motsvarade endast en roll, alternativet är att en nod kan ha flera roller associerad med sig, fördelen med att ha en roll per nod är att om man vill ändra rättigheter kan man göra det genom Authorization Manager istället för att ändra egenskaperna för navigationsträdet. Vill man att flera roller skall kunna komma åt en tjänst kan man lätt bygga upp en sådan struktur i Authorization Manager eller Active Directory. Vad

jag också beslutade var att implementera någon form av administratörsgränssnitt i den slutgiltiga prototypen, detta för att lätt kunna lägga till tjänster och ändra behörighet.

Implementation av OnePoint i ASP.net

Webbgränssnittet i portalen består i stort av fyra delar:

- Navigering
- Innehåll
- Administratörsgränssnitt
- Användarpersonifiering

Att implementera navigation i portalen handlar om att implementera web parten som sköter navigationen och få den att kommunicera med web parten som visar innehållet. Att visa innehåll innefattar även att göra en mall för hur man enkelt kan lägga till tjänster, vilket används i interaktionen med administratörsgränssnittet.

Navigering

Navigeringen består av web parten NavigationWebPart.cs och dess kommunikation med DetailsWebPart.cs som visar innehållet. I min tidigare lösning av rollbaserat innehåll byggde jag upp en TreeView baserad på en XML-struktur, detta kan ses som naturligt då TreeView har samma struktur som XML, problemet med den lösningen blir då man vill ändra eller läsa en nod i trädet. Jag beslutade därför att bygga upp en databas där strukturen är bevarad genom att varje tjänst har en referens till sin förälder. Att få ut en tjänst är sedan lätt om Idet till tjänsten sparas i navigationsträdet. Det svåra med den här lösningen är att skapa en SQL-query som tar ut tjänsterna så att de har samma hierarki som trädet. [Bild].

Kommunikation mellan web parts

När man arbetar med web parts kan man inte använda sig av referenser till andra web parts för kommunikation, istället får man använda sig av ett inbyggt koncept där man skapar connections mellan webparts. Det man gör då är att i huvudsidan anger att en kommunikation ska ha en sändare och en mottagare. I följande exempel konfigureras webpartmanagern för en kommunikation mellan webpartarna med ID ServiceConsumer och ServiceProvider.

```
            <asp:WebPartManager ID="WebPartManager1" runat="server">
                <StaticConnections>
                    <asp:WebPartConnection ID="connection"
ConsumerID="ServiceConsumer" ProviderID="ServiceProvider" />
```

```
        </StaticConnections>
    </asp:WebPartManager>
```

I sändar-webparten, ServiceProvider, anger man sedan vilken metod som skall kunna anropas. Här är ett exempel på en metod som kan anropas från en mottagar-webpart.

```
[ConnectionProvider("Service Provider"), "default")]
public IServiceInterface GetServiceData()
{
    return this;
}
```

Klassen som innehåller denna metod måste då implementera interfacet IServiceInterface, detta för att mottagar-webparten skall kunna få ut information utan att veta vilken klass som är sändare. Motsvarande metod i mottagar-webparten ser ut så här:

```
[ConnectionConsumer("Service Consumer")]
public void SetProviderData(IServiceInterface providerData)
{
    this.data = providerData;
}
```

När webparten körs anropas metoden som är markerad som ConnectionConsumer, i det här exemplet sparas datan undan för att kunna användas som vi vill senare. Interfacet kan vara utformat som vi vill, bara för att kunna få den informationen man behöver.

Ladda en tjänst

Att ladda en tjänst börjar med att användaren klickar på ett löv i navigationsträdet. Här visas vad som händer när användaren gjort detta.

Figur 7 Ladda tjänst

Dessa flöden körs parallellt, vid varje ny pageload så körs flödet till höger, för att inte sidan ska laddas om när användaren klickat på den tjänsten han är inne på kontrolleras detta. Det finns också ett fall då activeControl inte är satt, detta kan inträffa då användaren klickat på en nod som pekar på ett Id, eller att sidcykeln är inne i ett stadie då kommunikationen inte är initierad.

Administratörsgränssnitt

När en användare med rätt rättigheter ska ändra på portalen behövs ett gränssnitt för detta, valet låg mellan att skapa en egen webbplats för detta ändamål och att göra portalen på ett sätt så att innehållet kan ändras om du är inloggad som administratör. Fördelarna med det senare är att det blir mer intuitivt och lättanvänt,

fördelarna med en separat webbplats för ändamålet är att det blir lättare att implementera. Dock valde jag den mer dynamiska och lättanvända lösningen då det passar bättre ihop med konceptet med WebParts.

Webparten längst till höger visas då användaren klickar på ”edit” på en webpart, där visas de ändringar man kan göra på webparten, är man en vanlig användare visas endast egenskaper som påverkar layout och utseende, är man däremot administratör visas verktyg för att lägga till noder och tjänster i trädet.

Microsoft har en färdig EditorZone som kan innehålla olika EditorParts, dessa finns färdiga och kan manipulera layout och utseende på den aktuella webparten, vill man kunna ändra på andra egenskaper måste man göra en egen klass som ärver från EditorPart. I OnePoint har jag implementerat två egna EditorParts, en för att lägga till tjänster och en för att ändra på existerande tjänster. Vad man sedan gör för att koppla samman dem med en webpart är att man lägger till en instans av den i webpartens lista med EditorParts.

Skillnaden i kommunikationen mellan en WebPart och dess EditorParts är att man inte behöver skapa en statisk kommunikation mellan dem, ärver man från EditorPart får man ett fält WebPartToEdit som är en referens till den webparten som editeras för tillfället, detta gör det enkelt att uppdatera egenskaperna hos WebParten.

Klassdiagram

Att designa klassdiagram för webbutvecklingsprojekt går inte att göra på samma sätt som traditionell utveckling, detta på grund av det faktum att kommunikationen mellan klasser inte sker direkt på samma sätt som då en klass har en instans av en annan. I de diagram som följer är pilar mellan klasser tänkt att visa hur kommunikationen sker, vad gäller interface så har jag använt traditionell standard.

WebParts

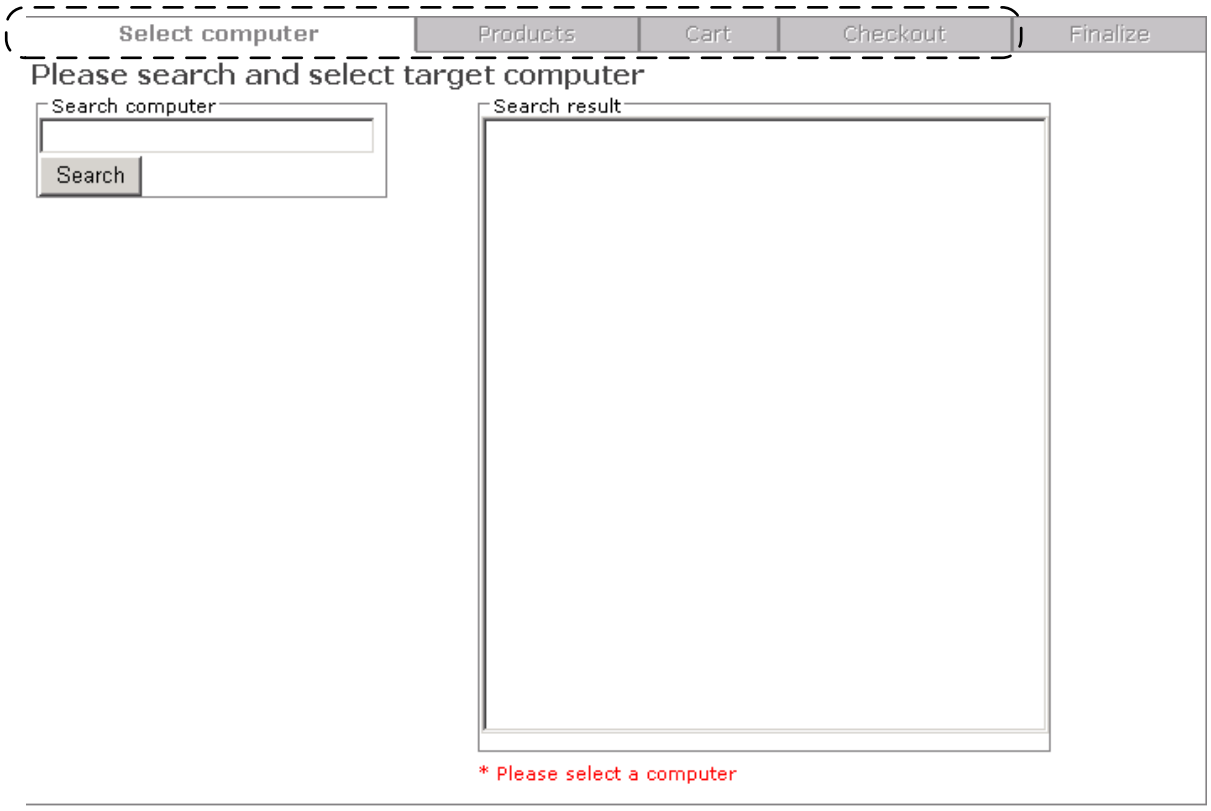
De WebParts jag använder mig av är:

- NavigationWebPart
- DetailsWebPart
- AdminWebPart

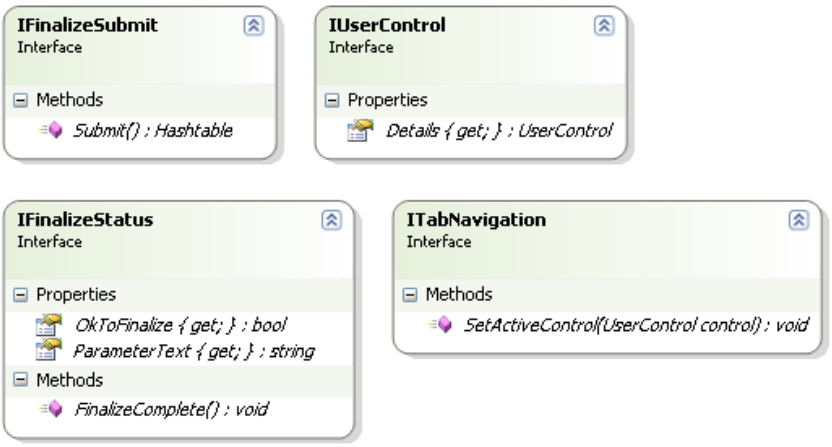
NavigationWebPart består av en TreeView som visar trädstrukturen på sidan och som är klickbar, beroende på vad användaren klickar så skickas tre olika events, TreeNodeCollapsed, TreeNodeExpanded och SelectedNodeChanged som hämtas av

respektive metoder. Det är i metoden SelectedNodeChanged värdet på den aktuella noden kontrolleras och sparas i ViewStaten, när sedan DetailsWebPart har laddats använder den sin uppsatta kommunikation för att hämta det värdet och laddar den nya UserControlen.

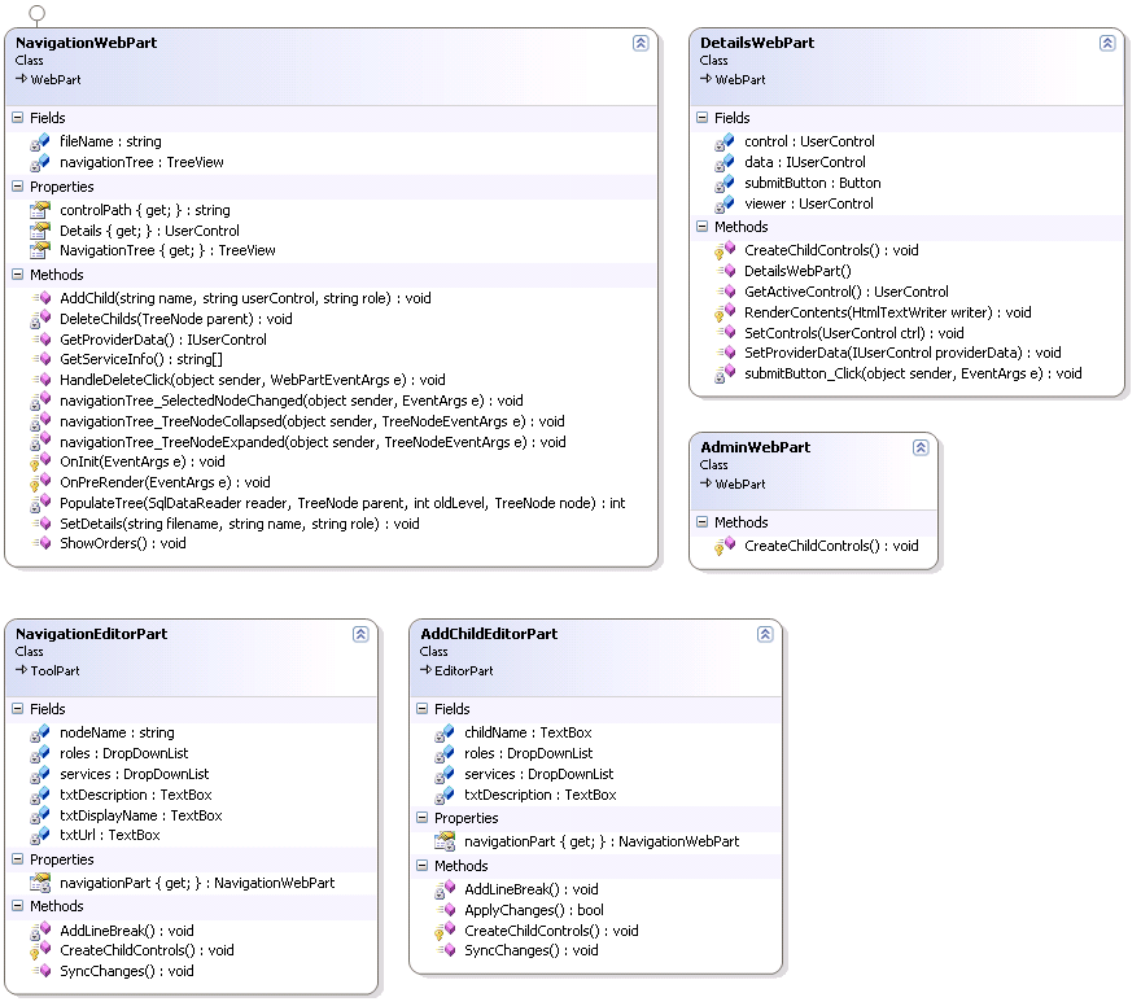
DetailsWebPart består av en s.k. MultiView som består av två vyer vilka användaren kan växla mellan, antingen genom flikar som i bilden nedan eller knappar. Den första fliken består av den aktuella tjänsten som i sig består av ännu en MultiView. I bilden nedan är tjänstens MultiView markerad. Jag valde att göra på detta sätt för att alla tjänster ofta består av ett antal olika steg efterföljt av ett sista steg då användaren godkänner de val han gjort och skickar en förfrågan.



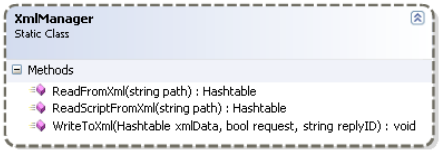
För att en tjänst ska kunna visas på detta sätt krävs att den implementerar ett antal interface.

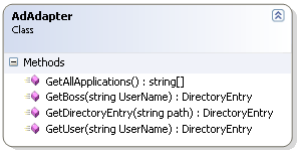


Om en tjänst implementerar interfacet IFinalizeSubmit måste tjänsten implementera metoden Submit vilken skickar parameterdata till XMLManagern som sedan skriver till en fil. IUserControl behövs för att NavigationWebPart och DetailsWebPart ska kunna kommunicera med varandra. I IFinalizeStatus hanteras information som skall visas för användaren när denne når sista steget i en beställning eller liknande. ITabNavigation möjliggör för kontrollen att visa flera flikar.



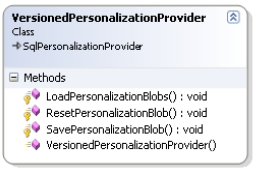
NavigationWebPart och DetailsWebPart hanterar den största delen av det användaren gör. De kommunicerar huvudsakligen genom metoden GetProviderData som returnerar den aktuella kontrollen som en IUserControl. NavigationWebPart kommunicerar även med NavigationEditorPart och AddChildEditorPart. Dessa EditorParts hanterar editeringsläget i portalen, i metoden SyncChanges synkroniseras de förändringar som gjorts i editeringsläget med inställningarna för den aktuella WebParten. Att notera hos alla klasserna är att metoden CreateChildControls finns med, den skapar alla kontroller som sidan/WebParten innehåller rekursivt. Om en WebPart innehåller en UserControl som i sin tur innehåller kontroller så skapas först UserControlen som sedan anropar CreateChildControls på sitt eget objekt för att skapa underkontroller.

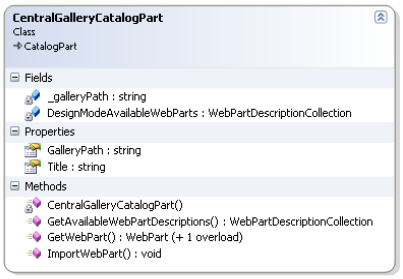


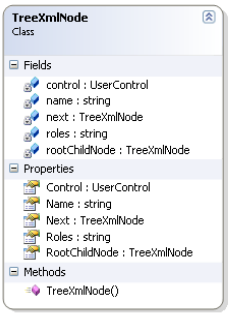


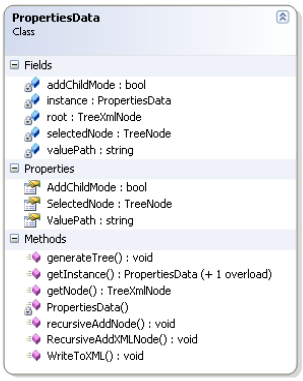
XmlManager är en statisk klass som fungerar som interface mot den skriptmotor som Joel Sanderi har gjort. Datan som skickas från tjänsten genom Submitsteget skrivs till en fil i XMLform. Den används även för att läsa från en XML-fil för att visa upp obesvarade ordrar för en administratör.

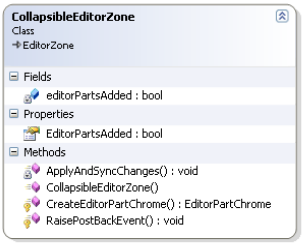
För att skriva och läsa mot ADt har jag skapat en klass med metoder för att hämta dels generella DirectoryEntry men också mer specifik information som inloggad användares chef eller vilka applikationer som finns tillgängliga för installation.

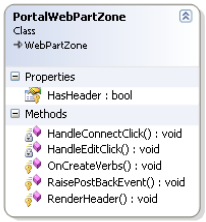


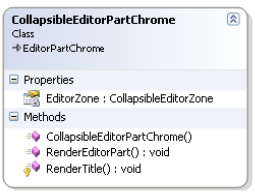


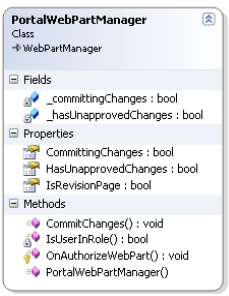












För att kunna erbjuda samma stöd för personalisering som Sharepoint har jag skapat en klass som arbetar mot en SQL-databas och sparar information om den inloggade användaren, VersionedPersonalizationProvider som ärver av SqlPersonalizationProvider.

Klasserna TreeXmlNode och PropertiesData erbjuder ett alternativ till att spara information om tjänster och navigeringsträd i databas, här sparas information som XML och laddas genom singelton-klassen PropertiesData.

Editeringspanelen består av flera editorparts som läggs till klassen CollapsibleEditorZone, utseendet

på dessa hanteras genom CollapsibleEditorPartChrome.

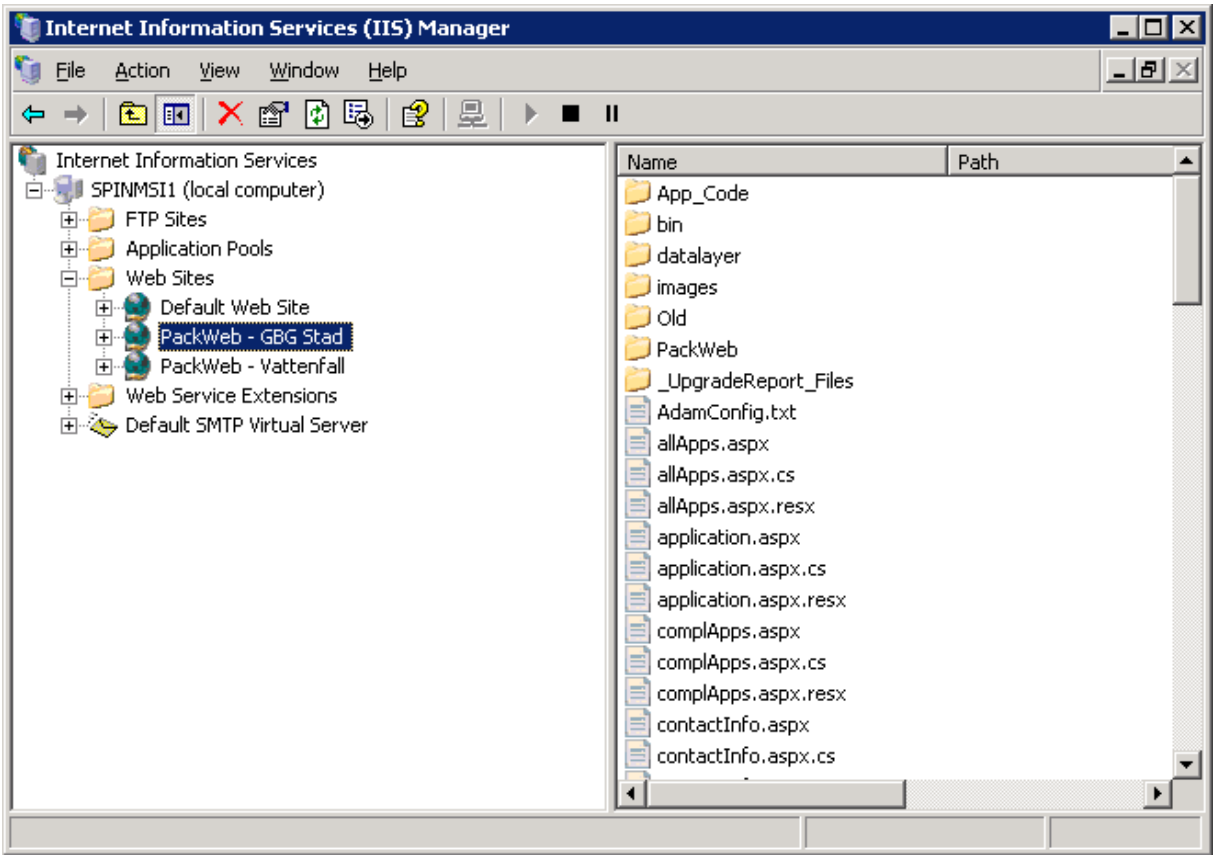
För att kommunikationen mellan WebParts skall fungera så måste man använda en WebPartManager, denna kontroll finns redan implementerad som en del av .NET-ramverket, jag har dock vidareutvecklat denna för att kunna hantera rolltillhörighet på ett bra sätt.

Virtuell domän

För att helt kunna testa prototypen i den miljön som den är tänkt att verka i satte jag upp en virtuell domän med två virtuella servrar, till detta har jag använt Microsoft Virtual Server. Den ena servern är en domänkontrollant och har ett Active Directory som portalen använder, den andre servern är en applikationsserver/webbserver, här ligger portalen, IISen, databasen och rollhanteringen. I en riktig miljö med stor organisation kan man välja att ha en multiserverlösning, med en webbserver, en domänkontrollant och en SQL-server. Har man dessutom en bakomliggande processmotor som kör kan man lägga denna separat för att få högre prestanda. Att sätta upp en server att kunna köra IIS med ASP.net kräver att man går igenom ett antal steg:

- Installera IIS och .net Framework
- Registrera ASP.net genom att köra aspnet_regiis
- Sätta upp IIS:en
- Ge rättigheter till ASP.net-kontot

När man har installerat IIS:en kan man köra Internet Information Services Manager.

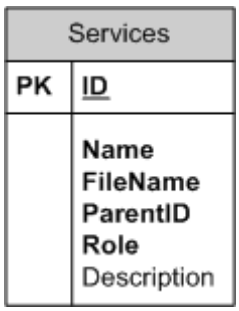


Figur 8 IIS Manager

Det är i IIS Manager man konfigurerar sina webbplatser, här kan man ställa in vilken säkerhet webbplatsen skall använda, vilket autentiseringssätt som skall användas och mycket mer.

SQL-databaser

För att hantera tjänster och kunna spara information om varje användares personliga inställningar har jag använt mig av två SQL-databaser. Services innehåller information om varje tjänst och ser ut så här:



Figur 9 Tjänstdatabas

ID är primärnyckel och används förutom för att hämta tjänster också för att kunna bygg upp trädstrukturen tillsammans med ParentID som pekar på dess förälder. FileName är en sökväg till UserControlen för att kunna ladda den dynamiskt och Role för att rollbaserat innehåll för varje tjänst ska gå att implementera. Den andre databasen kommer med ASP.net och innehåller användarinformation, de inställningar en användare kan göra med WebParts sparas där.

Integration med skriptmotor

För att kunna integrera min lösning med Joel Sanderis lösning för den bakomliggande flödesmotorn ställdes följande krav:

- En gemensam XML-standard för att kunna starta nya flöden
- En gemensam databas där godkännandesteg sparas

För att starta nya flöden skapar den aktuella tjänsten en XML-fil där de parametrar som krävs för att flödet skall fortgå listas, i exemplet Order Software innebär detta: mjukvarans namn, mjukvarans ID, beställare och godkännare samt e-postadresser till dessa. Då flödet har gått igenom och ett godkännande behövs måste det kunna göras från portalen, för att lösa detta använde vi en gemensam databas där användar-ID mappas till ett unikt ID som motsvarar en order, vid varje inloggning

från en administratör eller chef kontrollerar jag då databasen mot den inloggade användarens ID och tar ut eventuella ordrar. När administratören svarat på ordern skapas en ny XML-fil med svaret och en eventuell kommentar som skriptmotorn sedan hämtar upp och åtgärdar.

Slutsats

I examensjobbsbeskrivningen stod att jag skulle visa hur man kunde lösa uppgiften genom att utveckla en prototyp, detta har jag gjort, men man måste tänka på att det är en prototyp. Om en fullvärdig produkt skall implementeras kan man använda mycket av det jag har gjort, dock bör man göra en ordentlig målgruppsundersökning för att kunna erbjuda ett fullgott gränssnitt. Dessutom bör man testa hur lösningen hanterar ett stort antal användare som fallet ofta är på företagen som kan tänkas köpa en lösning liknande OnePoint.

Vägval

Det jag i efterhand önskar jag gjorde annorlunda var att jag inte började med WebPart-lösningen tidigare, att förstå hur WebParts kommunicerar och är uppbyggda tog relativt lång tid. I de flesta fall då jag har tvekat hur jag skulle göra har jag fått stor hjälp av min handledare och andra anställda på Spintop, detta har gjort att jag har kunnat undvika många felaktiga vägval.

Då jag inte hade tillgång till Spintops källkod till ZTP under större delen av mitt arbete var det till stor glädje när jag fick se denna och insåg att jag löst problem på snarlika sätt som de gjort.

Vidare utveckling

De mål jag hade med portalen nådde jag, men självklart kan nya områden utforskas. De delar som jag känner kan föra OnePoint eller liknande portaler framåt är:

- AJAX
- Gränssnitt mot bakomliggande skriptmotor
- Språkstöd

AJAX är en metod för att undvika att hela sidan laddas om varje gång en kontroll uppdateras, många nya möjligheter uppenbaras när med konceptet, sökningar som visar resultat interaktivt är ett exempel. Gränssnitt mot skriptmotor är ett sätt att underlätta administrationen då man lägger till nya tjänster, att kunna specificera XML:en i webbgränssnittet samt bestämma vilka skript som ska köras när en tjänst exekveras underlättar stort för underhållet av portalen. I den integrationen vi gjorde hårdkodas parametrar in i varje tjänst, detta måste då också göras i skriptmotorn och är en källa till onödigt arbete och misstag efter erfarenheter av arbete med Spintops lösning.

Språkstöd har implementerats i Spintops version av ZTP med gott resultat, här använder de sig av en resursfil som innehåller texten för alla kontroller som finns i portalen, ett alternativt sätt att lösa det på är att ha resursfilen i assemblyn då man även kan spara objekt, detta kan vara användbart om man t.ex. vill kunna ändra en bild beroende på vilket språk som är inställt. Fördelen med att ha en resursfil är att man kan ändra den dynamiskt, vilket jag ser som mycket användbart om man vill kunna ändra på texter i portalen dynamiskt.