TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Utveckling av webbapplikation för informationshantering i projekt

Robert Nygren

EXAMENSARBETE 2007

ÄMNE DATATEKNIK

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

Utveckling av webbapplikation för informationshantering i projekt

Development of a web application for managing information in projects

Robert Nygren

Detta examensarbete är utfört vid Tekniska högskolan i Jönköping inom ämnesområdet Datateknik. Arbetet är ett led i magisterutbildningen. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Lars-Olof Petersson
Omfattning: 10 poäng (C-nivå)
Arkiveringsnummer:
Datum: 2007-08-30

Abstract

Gain IT Architecture & Solutions (referred to as Gain IT) is a small IT-company based in Huskvarna. Gain IT:s business areas include products, consulting and support. Within the product area they are selling software and also their own customized web applications. The consulting area offers development of custom web applications. The Support area concentrate on support and helpdesk services for third party companies and for their own custom made web applications.

Gain IT had plans to begin developing a web application to manage information relating to different projects. The web application was supposed to manage different activities belonging to a specific project. In a given activity there can be different types of information making up that activity.
These pieces of information, called commenttypes, can for example be general system descriptions, notes, decisions and contacts. Developers and project managers can then keep track of progress made in different activities when a specific project is performing.

The purpose of the report is to answer two main questions:

- What does XML Web Services consist of?
- How to develop an XML Web Service with Visual Basic 2005 and .NET Framework 2.0?

The author's goal was to develop a web application to manage information in projects. It was supposed to manage different types of comments relating to projects and activities. The web application was also supposed to be built with service oriented architecture in mind. Information is then more easily accessed and shared between different systems.

Ideas and suggestions were made from different people on the development team, relating to aspects of the web application, while developing it.

The author's work has led to a well working web application to manage information relating to projects. Both the author and Gain IT are pleased with the result.

The author's personal opinion is that the web application can be improved even more in the future. New functionality can be added to the system more easily much thanks to XML, XSD, XSL and Web Services but also because good programming techniques have been used.

Sammanfattning

Gain IT Architecture & Solutions AB (refereras till som Gain IT) är ett litet Huskvarnabaserat IT-företag. De är verksamma inom de tre affärsområdena produkt, konsult och service. Inom produktområdet sker försäljning av programvaror och deras egenutvecklade webbapplikationer. På konsultsidan erbjuder Gain IT bl.a. utveckling av webbapplikationer. Serviceområdet inriktas bl.a. på support- och helpdesk-tjänster för företag men även support och utbildning för egna applikationer.

Gain IT har under en tid planerat att utveckla någon form av webbapplikation för att hantera information i projekt. Den ska kunna hantera olika aktiviteter som kopplas till ett specifikt projekt. Det finns olika typer av ”kommentarer” som kan kopplas till varje aktivitet i ett särskilt projekt. Exempel på kommentartyper är generella systembeskrivningar, anteckningar, beslut och kontaktuppgifter. Utvecklare, projektledare etc. kan på så sätt koppla viktig information till projekt under tiden som ett särskilt projekt bedrivs.

Författarens frågeställningar var följande:

- Vad består XML Web Services av?
- Hur utvecklar man en XML Web Service med Visual Basic 2005 och .NET Framework 2.0?

Författarens mål var att utveckla en webbapplikation för informationshantering i projekt. Den skulle kunna hantera olika typer av kommentarer som kopplas till projekt och aktiviteter. Webbapplikationen skulle också byggas med tanke på “service-oriented architecture”, vilket innebär att information ska kunna vara åtkomlig från fler system än bara webbapplikationen.

Allteftersom författaren utvecklade projekthanteringsystemet fick han bra synpunkter och ideèr till förslag till förbättringar, avseende webbapplikationen, från personer på utvecklingsavdelningen.

Examensarbetet har utmynnat i en välfungerande webbapplikation för hantering av information i projekt. Både författaren och uppdragsgivaren är nöjda med resultatet.

Författaren anser även att projekthanteringsystemet kan förbättras ännu mer i framtiden och att ny funktionalitet kan adderas enklare mycket tack vare teknologierna XML, XSD, XSL och Web Service men även pga. god programmeringsteknik har använts.

Nyckelord

XML
XSD
XSLT
SOAP
WSDL
UDDI
Web Services

Innehållsförteckning

Figurförteckning

1 Inledning

1.1 Förutsättningar

1.1.1 Företagets bakgrund

Gain IT Architecture & Solutions AB (refereras till som Gain IT) är ett litet Huskvarnabaserat IT-företag. De är verksamma inom de tre affärsområdena produkt, konsult och service. Inom produktområdet sker försäljning av programvaror och deras egenutvecklade webbapplikationer.
Som konsult erbjuder Gain IT bl.a. utveckling av robusta webbapplikationer. Serviceområdet inriktas bl.a. på support- och helpdesk-tjänster för företag men även support och utbildning för egna applikationer.

1.1.2 Författarens bakgrund

Författaren är i slutskedet av en magisterutbildning inom internetteknik. Författaren har en tidigare examen inom IT-området; högskoleingenjör i datateknik. Den nuvarande magisterutbildningen inriktar sig mot utveckling av webbapplikationer främst inom .NET-miljön.

1.2 Problemformulering

Gain IT har under en tid planerat att utveckla någon form av webbapplikation för informationshantering i deras projekt. Den ska kunna hantera olika aktiviteter som kopplas till ett specifikt projekt. Till varje aktivitet hör uppgifter (kommentarer). Det finns olika typer av kommentarer som Gain IT har behov att koppla till varje projekt och aktivitet. Exempel på kommentartyper är generella anteckningar, systembeskrivningar, beslut och kontaktuppgifter. Utvecklare, projektledare etc. kan på så sätt koppla viktig information till projekt under tiden som ett visst projekt bedrivs.
En dokumentatör kan under ett utvecklingsprojekt direkt hämta t.ex. systembeskrivningar så att en så fullständig och korrekt systemdokumentation kan göras.

Uppdragsgivaren, Gain IT, önskar att en webbapplikation utvecklas för informationshantering i projekt. Den ska kunna hantera olika typer av kommentarer som kopplas till projekt och aktiviteter. Webbapplikationen ska byggas med tanke på "service-oriented architecture", vilket innebär att information ska kunna vara åtkomlig från fler system än webbapplikationen t.ex. deras affärssystem.
Därmed har författaren utvecklat följande frågeställningar som ska besvaras av denna rapport:

- Vad består XML Web Services av?
- Hur utvecklar man en XML Web Service med Visual Basic 2005 och .NET Framework 2.0?

1.3 Syfte och mål

Gain IT arbetar i dagsläget med .NET-ramverket (på engelska .NET-Framework) och ASP.NET 2.0 för att utveckla webbapplikationer.
Det .NET-kompatibla språket Visual Basic 2005 (VB 2005) används i hög omfattning av företaget. Klassbibliotek har framtagits som används av utvecklarna för alla webbapplikationer. Därmed blir koden mer standardiserad och enklare eftersom utvecklarna i grund och botten jobbar med samma klasser för t.ex. dataåtkomst.
Vid diskussion med utvecklingsansvarig på Gain IT framkom önskemål om att använda dessa klassbibliotek. Mitt val av utvecklingsplattform blir därför .NET och ASP.NET 2.0 samt VB 2005 som programmeringsspråk. Följande teknologier/mjukvaror kommer att utnyttjas:

- Microsoft Sql Server 2000
- .NET-Framework 2.0
- Active Server Pages 2.0 (ASP.NET 2.0)
- Microsoft Visual Basic 2005 (VB 2005)
- Web Forms

Författarens syfte är att använda ovannämnda teknologier för att visa hur man kan skapa "tjänsteorienterade" applikationer.
Författarens mål är att utveckla en webbapplikation för informationshantering i projekt. Den skall byggas med tjänsteorienterad arkitektur vilket innebär att information i bakomliggande databas ska kunna vara enkelt åtkomlig från andra existerande samt framtida system.

1.4 Avgränsningar

Arbetet kommer inte att omfatta teori om databassystem då läsaren bör ha en viss insikt i detta ämnesområde. Teori om hur man programmerar i ett särskilt språk (syntax) kommer inte att avhandlas då läsaren förutsätts ha viss insikt om detta.

1.5 Disposition

Denna rapport är skriven enligt den mall som Tekniska Högskolan har för examensarbeten. Den teoretiska bakgrunden vilket besvarar författarens frågeställning presenteras först. Teorikaptilet vilket behandlar Web Services och deras ingående beståndsdelar presenteras sakligt och noggrant. Teknologier som presenteras av den teoretiska bakgrunden innefattar XML, XSD, XSL (T), SOAP, WSDL, UDDI samt kodning av Web Service. Exempel på programmeringskod presenteras och förklaras ingående. Därefter följer genomförandet där det översiktligt presenteras tillvägagångssättet att utveckla projekthanteringssystemet. Utrustning som har nyttjats beskrivs också. Efter genomförandet presenteras resultatet där det beskrivs vad som är möjligt att åstadkomma med webbapplikationen. Tillsist presenteras slutsatsen och diskussionen där det beskrivs bl.a. vad som arbetet har utmynnat i, vilka teoretiska kunskaper som har inhämtats och framtidsutsikter.

2 Teoretisk bakgrund

2.1 Introduktion till XML Web Services

Idag finns det it-plattformar (olika datorsystem t.ex. PC med Windows) som skiljer sig drastiskt åt. Organisationer utnyttjar mjukvaror och it-plattformar från flera tillverkare. Då mångfalden och variationen av dessa it-plattformar ökar blir det svårare att hitta gemensamma protokoll för att transmittera, representera och utbyta information. För att kunna få ut så högt värde som möjligt av sin it-miljö (summan av alla datorresurser dvs. servrar etc.) är det av stor betydelse att de olika it-plattformarna kan kopplas samman.

Ett språk som kan fungera som ett gemensamt protokoll för representation och utbyte av information är eXtensible Markup Language (XML). Det är ett markup-språk (eller med andra ord meta-språk) som kan utnyttjas av datorer oberoende av operativsystem, hårdvara etc. Det finns inga fördefineriade taggar i markup-språket utan dessa är det upp till utvecklaren att komma på.

XML vilket började bli populärt under senare delen av 90-talet har också lett till utveckling av andra markup-språk (samma regler som XML men semantiskt annorlunda) baserade på XML. Exempel på det är XML Schema Definition (XSD) och Extensible Stylesheet Language (XSL) [4, pp. 72,73] . XML används för att representera strukturerad data medan XSD nyttjas för att säkerställa att informationen följer uppsatta regler som definieras av XSD [4, pp. 72,73]. XSL transformationer (XSLT) görs för att konvertera XML-strukturerad data till t.ex. HTML.

Vad är då XML Web Services (skrivs kortfattat Web Services) och vad kan den teknologin ha för fördelar med avseende på en it-miljö med varierande it-plattformar? Web Services är en typ av applikation som bygger på standardiserade protokoll (XML, XSD, SOAP, WSDL, UDDI etc.) för att utbyta information mellan it-system (servrar), oberoende av it-plattform (operativsystem), över Intranät och Internet [5].
XML används för att representera data medan Simple Object Access Protocol (SOAP) används för att definiera meddelandens struktur (informationen) som skickas mellan berörda parter (server -> server etc.) [3, pp.54]. Web Services Decription Language (WSDL) används för att beskriva hur en utomstående part (t.ex. klientapplikation) ska kommunicera med Web Servicen och vad Web Servicen är kapabel att utföra [12].

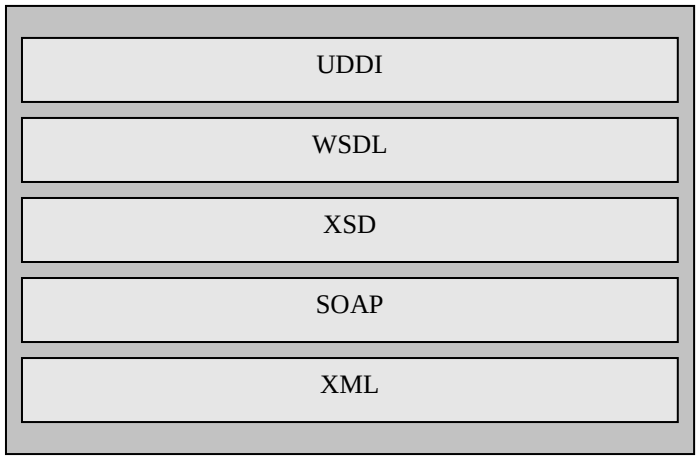
Eftersom Web Services kan ”installeras” på servrar både inom nätverk för intern åtkomst och för publik åtkomst, dvs. Internet, uppkommer frågan hur externa organisationer ska kunna lokalisera en viss Web Service och veta hur de ska använda den? [3, pp.52] Svaret på den frågan är Universal Description Discovery and Integration (UDDI).

UDDI definierar en register-tjänst för Web Services. Ett UDDI-register är en Web Service som hanterar information om publikt tillgängliga Web Services (som olika organisationer och företag har utvecklat) [6].
Med godkännandet av UDDI 3.0.2 som en OASIS-standard (2005) ska, enligt källor, UDDI Business Registry (ett publikt register för Web Services) läggas ned då Microsoft, IBM och SAP utvärderat registret och fattat beslutet att registrets mål har uppnåtts [6][7][8]. Microsoft, IBM och SAP som har implementerat stöd för UDDI i sina respektive produkter kommer fortfarande att stödja UDDI-protokollet [8].

Behovet av standarder som definierar avancerad funktionalitet för Web Services speciellt för säkerhet, pålitlighet och transaktioner är överhängande. Microsoft och andra företag svarade på detta behov och skapade en uppsättning specifikationer som refereras till som Ws-* arkitekturen [12].

2.2 Web Services komponenter

Figur 2-1 beskriver de fundamentala byggstenar som ligger till grund för Web Services-plattformen [2][13, pp.40].



Figur 2-1 Web Services grundläggande byggstenar

Det är av största betydelse att förstå nämnda teknologier (XML, SOAP, XSD, WSDL, UDDI) mer ingående därför att de utgör basen för Web Services. Därför kommer resten av den teoretiska bakgrunden i huvudsak fokusera på detta.

2.3 Extensible Markup Language

2.3.1 Introduktion till XML

XML vilket härstammar från Standard Generalized Markup Language (SGML), är ett markup-språk som utvecklades av World Wide Web Consortium (W3C) 1998 [3, pp.1,2]. W3C är ett internationellt konsortium där medlemsorganisationer utvecklar standarder och riktlinjer för webben [10]. Sedan lanseringen av XML har det vuxit i popularitet. Vad är det som gör XML till ett betydelsefullt markup-språk? Utvecklare kan skapa egna taggar (element och attribut) enligt egna önskemål [3, pp.1]. Det finns alltså inga fördefinierade taggar i XML som det gör i t.ex. Hypertext Markup Language (HTML) [3, pp.1]. XML är med andra ord ett metamarkup-språk eftersom man kan skapa egna markup-språk [3, pp.1].
XML kan således definiera och beskriva strukturerad data. För transport av XML kan webben och dess underliggande kommunikationsmetoder nyttjas.

Vad kan då XML utnyttjas för i verkligheten? Det kan brukas för alltifrån att göra enkla datafiler till att utbyta strukturerad data mellan applikationer i helt skilda nätverk [3, pp.1]. XML kan naturligtvis också nyttjas för att t.ex. utväxla dokument med ett gemensamt format mellan företag och organisationer.

2.3.2 Välformulerad och giltig XML

XML-kod skrivs enligt hårt satta regler [3, pp.3]. För att ett XML-dokument (som XML lagras i) ska vara ”välformulerad” måste den följa speciella regler som definieras av XML-specifikationen från W3C. Exempel på dessa regler är följande [3, pp.3]:

- Dokumentet måste innehålla exakt ett rotelement
- Dokumentet kan innehålla ett eller flera element
- Namnet på ett elements sluttagg matchar elementets starttagg

XML-dokument (mer om detta nedan) som ska vara giltiga måste inte bara vara ”välformulerad” utan även följa reglerna i ett associerat schema.
XML-dokumentet måste alltså valideras mot ett schema som kan uttryckas i form av t.ex. Document Type Definition (DTD) eller XML Schema Definition (XSD) [3, pp.3].

2.3.3 XML-Dokument

XML som beskriver strukturerad data lagras i dokument. XML-dokument kan lagras i vanliga datafiler (med filändelsen .xml) eller skulle kunna finnas i systemminnet på en datamaskin. XML-dokument består bara av ren text vilket betyder att inga speciella verktyg förutom en texteditor behövs för att skapa XML-dokument [3, pp.3].

```
<Kontakter>
   <Kontakt id="1">
      <Foretag>Företaget xyz AB</Foretag>
      <Stad>Jönköping</Stad>
      <GatuAdress>Företagsgatan 5</GatuAdress>
      <PostNr>555 10 </PostNr>
      <Epost>person@xyz.se</Epost>
   </Kontakt>
</Kontakter>
```

Figur 2-2 XML-fragment

Figur 2-2 beskriver ett utdrag av ett XML-dokument. Figuren beskriver inte ett komplett XML-dokument. Texten i figuren är indenterad och färgformaterad för läsbarhets skull. XML-fragmentets rotelement heter `<Kontakter>`. Också värt att notera är att taggar innesluter data. Starttaggar (ex. `<Kontakt>`) avslutas alltid (`</Kontakt>`) oavsett om dessa innehåller data eller inte. Den korrekta benämningen av taggen `<Kontakt>` är i XML-terminologi element. Elementet `<Kontakt>` har ett attribut som heter `id` och har värdet `1`. Elementet `<Kontakt>` innehåller också andra element.
Attribut läggs till i element för att ”markera” ytterligare information [3, pp.6]. Attributet `id` i Figur 2-2 är egentligen bara betydelsefullt för en applikation som behandlar XML-fragmentet. En applikation kan då separera (identifiera) en kontakt från en annan.

Element kan bestå av bokstäver (gemener och versaler i valfri kombination), siffror, bindestreck och ”underscore” [3, pp.5]. Däremot tillåts inte ”vita tecken” (mellanslag) i elementnamn.

För att XML-koden i Figur 2-2 ska kunna betraktas som ett komplett XML-dokument saknas dock en XML-deklaration [3, pp.4]. Deklarationen identifierar dokumentet som ett XML-dokument och specificerar vilken version av XML som används. Det är även möjligt att specificera vilken teckenkodning (t.ex. UTF-8) som XML-dokumentet är skrivet med.

XML-koden i Figur 2-2 skulle med en XML-deklaration se ut som XML-dokumentet i Figur 2-3.

```
<?xml version="1.0" encoding="utf-8" ?>
<Kontakter>
   <Kontakt id="1">
      <Foretag>Företaget abc AB</Foretag>
      <Stad>Jönköping</Stad>
      <GatuAdress>Företagsgatan 5</GatuAdress>
      <PostNr>555 10 </PostNr>
      <Epost>person@xyz.se</Epost>
   </Kontakt>
</Kontakter>
```

Figur 2-3 Ett komplett XML-dokument

Tecken inom `<?` Och `?>` tolkas av ”XML-processorer” som särskilda intruktioner [3, pp.4]. XML-deklarationen, `<?xml version="1.0" encoding="utf-8" ?>` , i Figur 2-3 talar om att detta dokument använder XML version 1.0 och teckenkodning UTF-8. Teckenkodningen är dock valfri att ange.

Text-data som finns i element kan uttryckas på två sätt. Det ena är som Parsed Character Data (PCDATA) och det andra är Character Data (CDATA) [3, pp.7]. PCDATA är vanlig text som tolkas av XML-processorn som markup-data (t.ex. text i elementet `<Foretag>`) [3, pp.7]. CDATA är användbart när man önskar skriva tecken som ska ignoreras av XML-processorn (t.ex. `<, >, &`). Särskilda tecken som t.ex. `<, >, &` har speciell betydelse i XML och kan inte skrivas direkt inuti ett element eftersom de är reserverade tecken. För att skriva reserverade tecken som PCDATA får man då referera till ett visst tecken med &#teckenkod och ; (tex. < för tecknet <).
Ofta används CDATA-sektioner för att inbädda kod som t.ex. VBScript eller JavaScript [3, pp.7]. Data som ska skrivas i en CDATA-sektion skrivs mellan `<![CDATA[` och `]]>`. Eftersom element som skrivs i XML-dokument kan ha samma namn men tillhöra olika föräldraelement är det viktigt att unikt kunna identifiera element och attribut på ett logiskt sätt. För att dela in en viss ”grupp” av element och attribut kan man utnyttja namespaces (namnutrymmen) [3, pp.8]. Namespaces används också för att associera ett visst XML-dokument med ett särskilt DTD eller XML-schema (t.ex. XSD) [3, pp.8].

```
<?xml version="1.0" encoding="utf-8" ?>
<Kontakter xmlns="http://www.xyz.se/Kontakter">
   <Kontakt id="1">
      <Foretag>Företaget abc AB</Foretag>
      <Stad>Jönköping</Stad>
      <GatuAdress>Företagsgatan 5</GatuAdress>
      <PostNr>555 10</PostNr>
      <Epost><![CDATA[ <person@xyz.se> ]]></Epost>
   </Kontakt>
</Kontakter>
```

Figur 2-4 Xml-dokument med namespace och CDATA-sektion

Figur 2-4 visar hur ett komplett XML-dokument med namespace och CDATA-sektion kan se ut. Attributet `xmlns` deklarerar ett namespace för elementet `<Kontakter>` [3, pp.10]. Tillåtna värden för attributet `xmlns` är Universal Resource Indicators (URI). En URI är ett unikt namn som identifierar en resurs [3, pp.9]. Uniform Resource Locators (URL) och Uniform Resource Numbers (URN) är tillåtna URI:er. Värdet som `xmlns` är satt till i Figur 2-4 behöver alltså inte peka på en webbresurs, det är ju en symbolisk identifierare [3, pp.10]. Alla element och attribut som har föräldern `<Kontakter>` tillhör därmed namespace ”http://www.xyz.se/Kontakter”. Men det är även möjligt att deklarera att element och attribut tillhör ett särskilt namespace på ett annat sätt. Nämligen med namespace-prefix [3, pp.10]. XML-dokumentet i Figur 2-4 skulle då kunna skrivas om till XML-dokumentet i Figur 2-5.

```
<?xml version="1.0" encoding="utf-8" ?>
<k:Kontakter xmlns:k="http://www.xyz.se/Kontakter">
   <k:Kontakt k:id="1">
      <k:Foretag>Företaget abc AB</k:Foretag>
      <k:Stad>Jönköping</k:Stad>
      <k:GatuAdress>Företagsgatan 5</k:GatuAdress>
      <k:PostNr>555 10</k:PostNr>
      <k:Epost><![CDATA[ <person@xyz.se> ]]></k:Epost>
   </k:Kontakt>
</k:Kontakter>
```

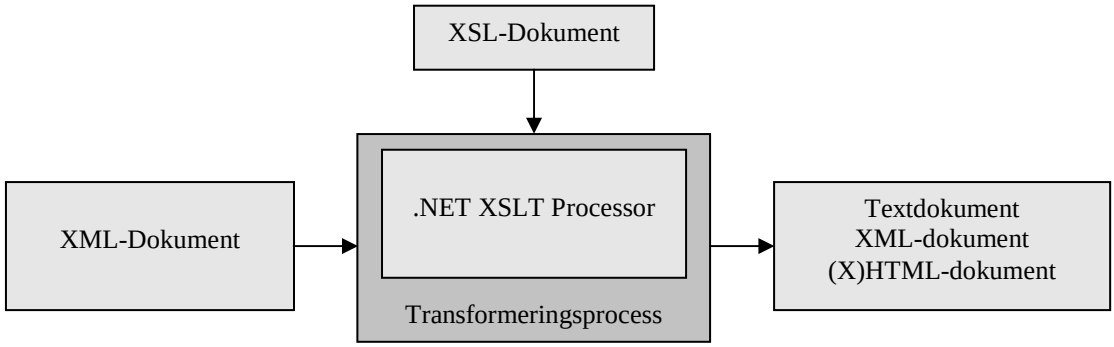
Figur 2-5 namespace-prefix

Elementet `<Kontakter>` (i Figur 2-5) deklarerar ett namespace-prefix, k. Dock behöver inte prefix skrivas med ett tecken det skulle kunna vara flera tecken. De element och attribut som använder prefixet, k, antas tillhöra det för k-deklarerade namespace-värdet (”http://www.xyz.se/Kontakter”).

2.4 Extensible Stylesheet Language Transformation

XSLT vilket är en förkortning av ovanstående titel har utvecklats av W3C som ett språk för att ”konvertera” (transformera) XML-dokument till andra typer av dokument [3, pp. 174]. Fastän XSLT inte ingår i basen av markup-språk och teknologier för Web Services-plattformen är det viktigt att känna till det. Att beskriva XSLT ingående är inte målet med denna rapport. Om läsaren är intresserad rekommenderas denna att läsa mer om XSLT på webbreferenserna http://www.w3schools.com/xsl/ och http://www.w3.org/TR/xslt.

XSLT-instruktioner skrivs för att kunna konvertera XML-dokument till (X)HTML-dokument, textdokument och till andra XML-dokument. XSL-filer (XSL-Dokument) vilket i grund och botten är XML-dokument lagrar instruktionerna som ska utföras av en XSLT-processor [3, pp.174].



Figur 2-6 Transformeringsprocess

Figur 2-6 beskriver processen att transformera ett XML-dokument med ett XSL-dokument och en XSLT-processor. En XSLT-processor (i exemplet .NET-plattformens) tar XML-dokumentet som ”input” och utför instruktioner som beskrivs i XSL-dokumentet [3, pp.174]. Det finns XSLT-processorer från olika tillverkare och för olika plattformar (tex. .NET-plattformen).

För att hämta data för de element och attribut som man ska infoga i det nya dokumentet (det som ska produceras) kan man använda XPath. XPath är ett slags frågespråk som möjliggör navigering/lokalisering av data i XML-dokument [3. pp.18].

För att läsaren ska få en liten inblick i hur man kan skriva XSL-dokument tar vi ett exempel. Hur skriver man ett enkelt XSL-dokument som extraherar och presenterar data från XML-dokumentet i Figur 2-3?

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
   xmlns:objKontakt="http://www.xyz.se/Kontakter"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" indent="yes" encoding="utf-8"/>
   <xsl:template match="objKontakt:Kontakt">
      <div>
         <table border="0" align="center" width="550px">
            <tr>
               <td style="width:15%"><span>Id:</span></td>
               <td><xsl:value-of select="@id" /></td>
            </tr>
            <tr>
               <td><span style="width:15%">Företag:</span></td>
               <td><xsl:value-of select="objKontakt:Foretag" /></td>
            </tr>
            <tr>
               <td><span style="width:15%">Stad:</span></td>
               <td><xsl:value-of select="objKontakt:Stad" /></td>
            </tr>
         </table>
      </div>
   </xsl:template>
</xsl:stylesheet>
```

Figur 2-7 XSL-dokument

Figur 2-7 beskriver ett XSL-dokument vars syfte är att konvertera XML-dokumentet i Figur 2-3 till ett HTML-dokument. Texten i Figur 2-7 är färgkodad med tanke på läsbarhet. Här har vi valt att endast presentera id, företag och stad för en kontakt men det kunde lika gärna inkludera alla tillgängliga element och attribut för XML-dokumentet i Figur 2-3. Längst upp i XSL-dokumentet återfinns XML-deklarationen vilket är bekant sedan tidigare. XML-deklarationen följs av ett element vid namn `<xsl:stylesheet/>` [3,pp.178].
Det elementet specificerar rotelementet för XSL-dokumentet. Attribut för elementet `<xsl:stylesheet/>`, i Figur 2-7, anger viken version (version=”1.0”) av XSL och namespace-prefixet `objKontakt` (med värdet `”http://www.xyz.se/Kontakter”`) specificeras. Även namespace-prefixet `xsl` deklareras vilket associeras med XSLT-specifikationen (`”http://www.w3.org/1999/XSL/Transform`”). Elementet `<xsl:output>` deklarerar diverse attribut för det resulterande dokumentet.
Attributen för elementet `<xsl:output>` i Figur 2-7 definierar det resulterande dokumentet som ett HTML-dokument kodat med teckenkodningen UTF-8. Attributet `indent=”yes”` anger att det resulterande dokumentet ska indenteras enligt sin hierarkiska struktur. Elementet `<xsl:template>` (Figur 2-7) definierar början av en mall. Attributet `match=”objKontakt:Kontakt”` betyder att mallen associeras med elementet `<Kontakt>`. Uttrycket ”`objKontakt:Kontakt”` är ett XPath-uttryck som väljer ut elementet `<Kontakt>`. För att hämta värden för element skriver man `<xsl:value-of select=”prefix:Elementnamn”`, se Figur 2-7.

2.5 XML Schema Definition

XML-dokument måste skrivas så att de är välformulerade och giltiga.
När det gäller ”giltighetskravet” är det nödvändigt att på något sätt kunna definiera regler för vilka element, attribut etc. som får finnas med.
För att avgöra om ett XML-dokument är giltigt måste man validera det mot ett associerat XML-schema. XML-schema vilket utvecklades av W3C är ett dokument som skrivs med XML-syntax där man definierar strukturen för XML-dokument [13, pp.25]. För att implementera W3Cs-rekommendationer för XML-schema valde Microsoft att utvecklad ett språk, XSD med XML-syntax [13, pp.41]. För att definiera element, attribut och datatyper kan man använda fördefinierade taggar i XSD [13, pp.41].

Det finns 44 inbyggda (enkla) datatyper att välja mellan [3, pp.16].
Dessutom kan man skapa egna datatyper baserade på de inbyggda [3, pp.16].
Datatyperna kan kategoriseras till två typer, enkla och komplexa [3, pp.45,46].
Man deklarerar enkla datatyper för element som inte innehåller andra element och attribut. Det motsatta gäller för komplexa datatyper.
Element som innehåller andra element och attribut kan man deklarera komplexa datatyper för.

Hur skulle ett XSD-dokument för XML-dokumentet i Figur 2-3 kunna se ut?

```
1   <?xml version="1.0" encoding="utf-8" ?>
2   <xsd:schema
3     targetNamespace="http://www.xyz.se/Kontakter"
4     elementFormDefault="qualified"
5     xmlns:ct="http://www.xyz.se/Kontakter"
6     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
7
8        <xsd:simpleType name="string5">
9           <xsd:restriction base="xsd:string">
10             <xsd:maxLength value="5" />
11          </xsd:restriction>
12       </xsd:simpleType>
13       <xsd:simpleType name="string250">
14          <xsd:restriction base="xsd:string">
15             <xsd:maxLength value="250" />
16          </xsd:restriction>
17       </xsd:simpleType>
18       <xsd:simpleType name="epost">
19          <xsd:restriction base="xsd:string">
20             <xsd:maxLength value="255" />
21             <xsd:pattern value="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"/>
22          </xsd:restriction>
23       </xsd:simpleType>
24
25       <xsd:element name="Kontakter">
26          <xsd:complexType>
27             <xsd:sequence>
28                <xsd:element name="Kontakt" maxOccurs="unbounded">
29                   <xsd:complexType>
30                      <xsd:sequence>
31                         <xsd:element name="Foretag" type="ct:string250" minOccurs="1"/>
32                         <xsd:element name="Stad" type="ct:string250" minOccurs="1"/>
33                         <xsd:element name="GatuAdress" type="ct:string250" minOccurs="1"/>
34                         <xsd:element name="PostNr" type="ct:string5" minOccurs="1"/>
35                         <xsd:element name="Epost" type="ct:epost" minOccurs="1"/>
36                      </xsd:sequence>
37                      <xsd:attribute name="id" type="xsd:int" use="required"/>
38                   </xsd:complexType>
39                </xsd:element>
40             </xsd:sequence>
41          </xsd:complexType>
42       </xsd:element>
43
44    </xsd:schema>
```

Figur 2-8 XSD-dokument

Figur 2-8 beskriver ett XSD-dokument som skulle kunna användas för att validera XML-dokumentet i Figur 2-3. Vi antar att XML-dokumentet (Figur 2-3) har samma namespace som XSD-dokumentet. Vidare är markup-koden i Figur 2-8 färgformaterad och identerad för att maximera läsbarheten.
På första raden (Figur 2-8) återfinns den välkända XML-deklarationen.
Den följs av (rad 2) `<xsd:schema>` elementet vilket måste finnas i alla XSD-dokument [3, pp.16]. Inom elementet `<xsd:schema>` definieras vilket namespace (”`http://www.xyz.se/Kontakter`”) som schemat tillhör (`targetNamespace`) [3, pp.16]. Attributet `elementFormDefault=”qualified”` innebär att element måste föregås av namespace-prefix [3, pp.16]. Namespace-prefixet `xsd` definieras inom `<xsd:schema>` elementet som refererar till namespace för XML-schema-specifikationen [3, pp.16].

Vidare definieras också ett eget prefix, `ct`, vilket associeras med XSD-dokumentets namespace.

Längre ned i XSD-dokumentet (rad 8 till 23) definieras egna datatyper som baseras på de inbyggda. De enkla datatyperna `string5`, `string250` och `epost` baseras på datatypen `string` (`<xsd:restriction base="xsd:string">`).
Antal tecken som de får innehålla begränsas med `<xsd:maxLength value=”tal”>` där ”tal” representerar maximalt antal tecken.
Extra begränsningar har satts för datatypen `epost` (rad 21) eftersom man önskar att e-post-adressen även följer ett reguljärt mönster (`<xsd:pattern value”…”`) så att den är så korrekt som möjligt. Reguljära mönster är uttryck som begränsar hur data (information) kan skrivas. Det är dock inte målet med denna rapport att lära ut hur man skriver reguljära uttryck så vid intresse rekommenderas läsaren att själv söka information om detta.

Vilka element, attribut etc. som XML-dokumentet i Figur 2-3 tillåts innehålla definieras av raderna 25 till 42 i Figur 2-8. Ett överordnat rotelement, `<Kontakter>`, deklareras (rad 25). Det är ett element som deklareras som en komplex datatyp (`<xsd:complexType>`) därför att den innehåller barnelement. Barnelement till rotelementet måste även komma i en viss ordning vilket deklareras med elementet `<xsd:sequence>`. Vidare deklareras ett barnelement, `Kontakt` (rad 28), till rotelementet. Attributet `maxOccurs=”unbounded”` (rad 28) innebär att elementet `<Kontakt>` får förekomma hur många gånger som helst i XML-dokumentet (Figur 2-3) [13, pp.46].

Elementen `Foretag`, `Stad`, `GatuAdress`, `PostNr` och `Epost` definieras som barnelement till elementet `Kontakt`. Datatyp för respektive element anges med `type=”ct:namn”` där `ct` är namespace-prefix och namn anger datatyp. Attributet `MinOccurs` för nämnda barnelement anger minimalt antal gånger ett element måste förekomma. Attributet `id` för elementet `Kontakt` deklareras på rad 37. Datatypen är `int` vilket innebär att attributet `id` kan anta heltalsvärden. Attributet måste även förekomma exakt en gång/`Kontakt` (`use=”required”`).

2.6 XML Web services på djupet

Web Services är en teknologi där det är möjligt att konstruera plattformsoberoende applikationer. Det innebär t.ex. att en applikation som utvecklas i VB 2005 och som körs på ett Microsoft Windows-baserat system kan ges åtkomst till från ett Linux eller Unix-baserat system.
Web Services ökar möjligheterna att integrera applikationer skapade med skilda utvecklingsmiljöer och som körs på olika it-plattformar (operativsystem) [13, pp. 32, 33].

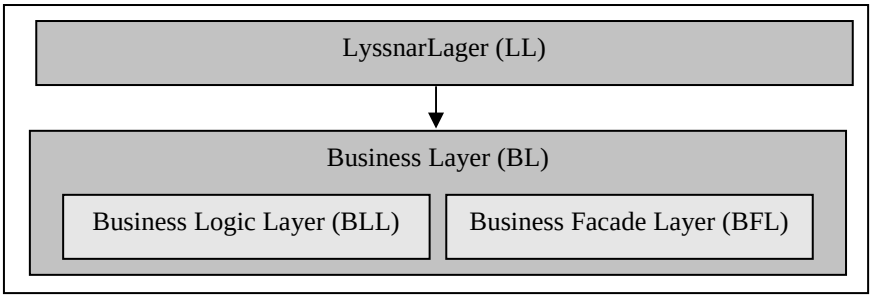
För att definiera vad är en Web Service är kan vi uttrycka oss på följande sätt: En Web Service är en applikation som exponerar funktionalitet (applikationslogik) till andra applikationer över det standardiserade protokollet HTTP [13, pp.33].

Web Service-plattformen är inte bunden till någon specifik tillverkare vilket betyder att det är fullt möjligt att skriva applikationslogik i t.ex. Java, VB 2005 och C# 2.0 [13, pp.32].

Att utveckla Web Services med .NET Framework 2.0 påminner i hög grad om att skriva webbapplikationer med ASP.NET. En viktig skillnad är dock att man inte bygger något grafiskt gränssnitt till applikationslogiken som man gör med ASP.NET (Web forms pages) [3, pp. 412].

En applikation som avser att anropa applikationslogik hos en Web Service benämns som Web Service clientapplication (klientapplikation) och applikationen som är värd för Web Servicen för Web Service provider application (provider-applikation). För att en klientapplikation ska kunna anropa metoder (logik) hos provider-applikationen behöver den känna till URL:en för Web Servicen [13, pp.33].

Figur 2-1 beskriver de byggstenar (teknologier) som ligger till grund (som Web Services är beroende av) för Web Services-plattformen. Utan de byggstenarna hade inte Web Services kunnat existera. Den interna arkitekturen för en Web Service, vilket dikterar hur Web Services fungerar består av två lager, ett lyssnarlager (LL) och ett Business Layer (BL). Figur 2-9 beskriver detta.



Figur 2-9 Web Services interna arkitektur

När en klientapplikation ska anropa logik hos en Web Service så skickar den ett ”request” till lyssnarlagret i Web Servicen (se Figur 2-9). LL tolkar ”request” som skickats från klientapplikationen och vidarebefordrar det till Business Layer. Den applikationslogik som ska exekveras för ett särskilt anrop (metod) implementeras av Business Logic Layer (BLL). Business Facade layer (FCL) implementerar gränssnittet mot BLL. Även viktigt att nämna är att LL konverterar resultat från exekvering av logik till ett format (SOAP) som klientapplikationen kan förstå [13, pp.34,35].

För att skriva egna Web Services kan man t.ex. utnyttja utvecklingsmiljön Microsoft Visual Studio 2005. Den Visual Studio mall som ska väljas heter ”ASP.NET Web Service”. Vanligtvis skapas det tre filer när man väljer en sådan mall. Den ena filen, ”Service.asmx”, definierar vilket språk (t.ex. `Language ="vb"`), codebehind-klass (tillhörande klass, `Class`) och namn på källkodsfil (`CodeBehind`) för codebehind-klassen (se Figur 2-10) [3, pp.413]. Och den andra filen är en källkodsfil, ”Service.vb”, och den sista filen, ”web.config”, definierar inställningar för Web Servicen (precis som för vanliga ASP.NET-webbapplikationer).

```
<%@ WebService Language="vb" CodeBehind="~/App_Code/Service.vb"
   Class="Service" %>
```

Figur 2-10 Service.asmx

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebService(Name:="Service", Description:="Min första Web Service" _
        , Namespace:="http://minwebbservice.se/")> _
    <WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
    Partial Public Class Service
    Inherits System.Web.Services.WebService

    <WebMethod(Description:="Adderar tal1 med tal2")> _
    Public Function AdderaTal(ByVal tal1 As Integer, _
        ByVal tal2 As Integer) As Integer
        Return tal1 + tal2
    End Function

End Class
```

Figur 2-11 Källkod för filen Service.vb

Figur 2-11 beskriver hur en codehind-klass för ”Service.asmx” kan se ut. För detta exempel valdes att skriva enkel kod för att sätta in läsaren i ämnet. Även för de som inte programmerar i VB borde koden vara lätt att förstå. Klasser deklareras med nyckelordet `class` (`partial` betyder att klassen inte behöver residera i en fil), vilka klasser som ska användas kan ”importeras” (kortare skrivning det går även utan `imports`) med nyckelordet `imports`. Funktioner, vilket är metoder med returvärde definieras med nyckelordet `Function`.

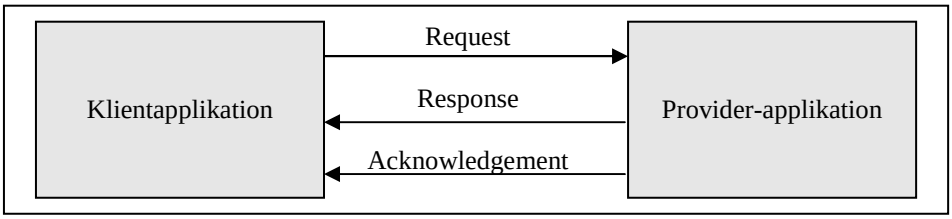
Enligt Figur 2-11 har vi en metod, `MultipliceraTal`, som tar två argument nämligen två heltal (32-bitars signerade heltal). Resultatet från anropet av metoden är summan av de två argumenten (`tal1+tal2`).
Naturligtvis är man inte begränsad av att returnera just heltal till klientapplikationen utan det är möjligt att returnera i princip vilken typ som helst (förutsatt att klientapplikationen är skriven i .NET).

Vad som speciellt bör uppmärksammas är att klassen `Service` ärver från klassen `System.Web.Services.WebService`. Nyttan med det är att man då får tillgång till funktionsrika ASP.NET-objekt som tex. `Application`, `Cache`, `Request`, `Session` och `Server`. Det är dock inget krav att en Web Service klass ska ärva från ovannämnda klass (`System.Web.Services.WebService`) [3, pp. 413].
Ytterligare en annan intressant del av koden är värt att nämna. För att metoder ska vara synliga från en klientapplikation måste de ”märkas” med attribut som specificerar deras ”beteende” [3, pp.415]. Som ett minimikrav ska metoder alltså märkas med attributet `<WebMethod()>` för att vara synliga och ”anropningsbara” från en klientapplikation (se Figur 2-11).
Egenskapen `Description` för nämnt attribut anger en beskrivning av metoden (som bäddas in i WSDL-beskrivningen) [3, pp.416] .

Det finns också ytterligare två attribut i Figur 2-11 som är värda en förklaring. Det är attributen `WebService` och `WebServiceBidning`. `Name`-egenskapen för attributet `WebService` anger namnet på Web Servicen i WSDL-beskrivningen. Egenskapen `Description` för samma attribut anger som tidigare nämnts en beskrivning av metoden men bäddas inte in i WSDL-beskrivningen utan visas istället i webbläsaren när man surfar till ”Service.asmx”. Och egenskapen `Namespace` anger det `xmlnamespace` som Web Servicen tillhör (som identifierar den och särskiljer den från andra Web Services). Attributet WebService är valfri att ange [3, pp.414]. Det andra attributet, `WebServiceBindning` anger vilken ”specifikation” som Web Servicen ska följa (`WsiProfiles.BasicProfile1_1`) [3, pp.414, 415].

2.7 Simple Object Access Protocol

SOAP vilket är en förkortning av ovanstående titel är ett protokoll som används för att kommunicera mellan applikationer i en distribuerad miljö.
SOAP bygger på request-response modellen (se Figur 2-12) där en applikation skickar ett s.k. request (tex. HTTP-POST) från en dator till en annan över ett nätverk och sedan förväntar sig svar från mottagarapplikationen. Meddelanden som utväxlas mellan applikationerna använder XML som data-representationsformat. SOAP är med andra ord XML med fördefinierade regler för hur meddelanden ska struktureras (fördefinierade element, attribut etc.). Eftersom XML är oberoende av it-plattform, hårdvara etc. har SOAP därmed samma karakteristik. SOAP och XML är ju bara vanlig text. [13, pp.54]



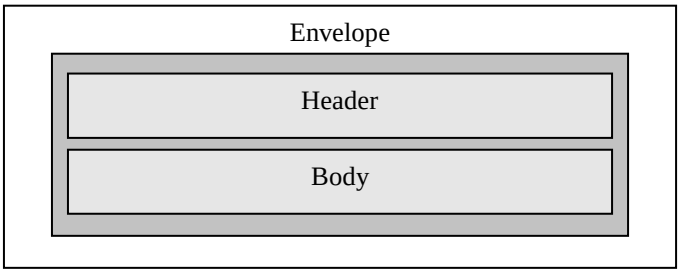
Figur 2-12 Transmission av SOAP-meddelanden över ett nätverk

Web Services använder SOAP för utväxla strukturerad data. Fastän SOAP har många fördelar som t.ex. att HTTP kan utnyttjas, att SOAP-meddelanden kan krypteras med HTTP-metoder etc., finns det några nackdelar som är värda att nämna. Att transmittera SOAP-meddelanden över nätverk kräver god bandbredd. XML som SOAP bygger på kräver större kvantiteter av minne och CPU. Från ett utvecklarperspektiv tillåter dessutom inte SOAP att man kontrollerar fel vid design-time (när applikationen designas) det måste göras vid run-time (då applikationen körs).
SOAP är trots nämnda nackdelar ändå ett fördelaktigt protokoll som ofta utnyttjas för att transmittera data mellan applikationer [13, pp.55,56].

SOAP:s interna arkitektur kan delas in i följande komponenter;
SOAP-meddelanden, SOAP remote procedure call (RPC) och SOAP-kodningsregler. SOAP-meddelanden kan skrivas med två olika stilar, antingen som dokument/textbaserat eller som RPC/kodad-form. Web Services använder dokument/textbaserad-stil som standard [13, pp.63].

2.7.1 SOAP-meddelanden

SOAP-meddelanden skickas enligt request-response modellen vilket nämnts tidigare (se Figur 2-12). Data som ska överföras bäddas in i de komponenter som ett SOAP-meddelande består av. Ett SOAP-meddelande byggs upp av komponenterna envelope, header och body (se Figur 2-13). Som läsaren kanske redan misstänker motsvarar då envelope meddelandets rotelement och de två andra komponenterna är barnelement till envelope [13, pp.60].



Figur 2-13 Komponenter i ett SOAP-meddelande

Header-elementet innehåller information om hur klientapplikationen ska bearbeta SOAP-meddelandet. Elementet är valbart och behöver inte definieras, dock kan flera header-element definieras i samma SOAP-meddelande [13, pp.60].

Body-elementet innehåller information som ska ingå i SOAP-meddelandet och är obligatorisk. Elementet består av barnelement som definierar t.ex. vilken metod som ska exekveras eller resultatet av densamme [13, pp.62].

Nu ska vi se hur ett SOAP-meddelande för exekvering (”request” från klientapplikation) av metoden i Figur 2-11 kan se ut (med SOAP 1.1).

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AdderaTal xmlns="http://minwebbservice.se/">
      <tal1>10</tal1>
      <tal2>15</tal2>
    </AdderaTal>
  </soap:Body>
</soap:Envelope>
```

Figur 2-14 SOAP-meddelande för exekvering av metod (request)

Elementet `Envelope` definierar tre xml-namespace-prefix (värde för respektive prefix kan ses i Figur 2-14). Elementet `Body` definierar själva huvudtexten i meddelandet. Barnelementet `AdderaTal`, till `Body`, innebär att metoden `AdderaTal` ska anropas. Argumenten för metoden, `AdderaTal`, anges som barnelement (värde 10 och 15 för respektive argument).

Resultatet för exekvering av metoden i form av ett SOAP-meddelande kan ses i Figur 2-15 (med SOAP 1.1).

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AdderaTalResponse xmlns="http://minwebbservice.se/">
      <AdderaTalResult>25</AdderaTalResult>
    </AdderaTalResponse>
  </soap:Body>
</soap:Envelope>
```

Figur 2-15 SOAP-meddelande för resultat av exekvering av metod (response)

Skillnaden mellan det resulterande SOAP-meddelandet och meddelandet för exekvering av metoden är ganska klar (se Figur 2-15) . Elementet `AdderaTal` heter istället `AdderaTalResponse` och har ett barnelement som heter `AdderaTalResult`. Värdet som ses inom detta barnelement är resultatet av exekvering av metoden `AdderaTal` hos Web Servicen.

2.7.2 SOAP Remote Procedure Call

Den andra komponenten som SOAP består av är SOAP remote procedure call (RPC). RPC är ett sätt att anropa en procedur (remote procedure, RP) på en dator som kan t.ex. residera i ett helt annat nätverk än den egna datorn (i viss grad liknande Web Services). En av de stora skillnaderna med RPC jämfört med Web Services är att RPC inte är begränsat till HTTP-protokollet utan kan använda helt andra protokoll [5]. SOAP RPC innebär alltså att RPC använder SOAP som datarepresentationsformat.

När man ska anropa en procedur (RP) på en annan dator följer man SOAP:s request-response modell vilket förklarats tidigare. Vid anrop av RP ska en s.k. `call`-instruktion medfölja SOAP-meddelandet. En `call`-instruktion är i själva verket ett stycke SOAP-kod (lite liknande Figur 2-15) som specificerar vilken procedur som ska anropas och vilka argument och värden som den tar.
När fjärrdatorn, dator som exekverar procedur, är klar returneras en s.k. result-”instruktion” vilket är resultatet av exekvering av RP i form av SOAP-kod [13, pp.64,65].

2.7.3 SOAP-kodningsregler

SOAP-kodningsregler är en uppsättning regler, liknande XSD, som bestämmer hur ett SOAP-meddelande kan konstrueras. Kodningsreglerna specificerar vilka datatyper som kan inkluderas i ett SOAP-meddelande. Datatyper som kan användas är antingen av typen enkel eller sammansatt. Enkla datatyper är de typer som definieras av specifikationen för XML Schema. Exempel på enkla datatyper är `string`, `integer`, `float` och `date` [13, pp.66, 67].
En sammansatt datatyp är en typ som består av en eller flera enkla datatyper. Det finns två typer av sammansatta datatyper, struktur och vektor (indexerad variabel). En struktur (jämför med `Structure` i VB 2005) är en samling variabler med olika namn och med samma eller olika datatyper.
Vektor är (tänk på en lista) alltså en uppsättning variabler som har samma namn och datatyp men värdet för en enskild variabel nås med ett index (heltalsvärde) [13, pp.66, 67].

2.8 Web Service Description Language

W3C vilket utvecklade WSDL är en förkortning av ovanstående titel och är ett beskrivningsspråk som praktiskt manifesteras i form av ett WSDL-dokument som definierar vad en Web Service är kapabel att utföra. För att kunna anropa metoder som en Web Service erbjuder måste man veta bl.a. vilka argument de tar och returvärden etc. Detta och mer nödvändig information beskrivs i form av XML i ett sådant dokument [13, pp.84].

När man skapar Web Services i Visual Studio 2005 så skapas automatiskt WSDL-dokument. WSDL-dokumentet återfinns i en fil som heter likadant som Web Servicen men slutar med filändelsen wsdl [13, pp.85].

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://minwebbservice.se/" xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://minwebbservice.se/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Min första
        WebService</wsdl:documentation>
 <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://minwebbservice.se/">
       <s:element name="AdderaTal">
          <s:complexType>
             <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="tal1" type="s:int" />
                <s:element minOccurs="1" maxOccurs="1" name="tal2" type="s:int" />
             </s:sequence>
          </s:complexType>
       </s:element>
       <s:element name="AdderaTalResponse">
          <s:complexType>
             <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="AdderaTalResult" type="s:int" />
             </s:sequence>
          </s:complexType>
       </s:element>
    </s:schema>
 </wsdl:types>
 <wsdl:message name="AdderaTalSoapIn">
    <wsdl:part name="parameters" element="tns:AdderaTal" />
 </wsdl:message>
 <wsdl:message name="AdderaTalSoapOut">
    <wsdl:part name="parameters" element="tns:AdderaTalResponse" />
 </wsdl:message>
 <wsdl:portType name="ServiceSoap">
    <wsdl:operation name="AdderaTal">
       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Adderar tal1 med tal2
       </wsdl:documentation>
       <wsdl:input message="tns:AdderaTalSoapIn" />
       <wsdl:output message="tns:AdderaTalSoapOut" />
    </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="AdderaTal">
       <soap:operation soapAction="http://minwebbservice.se/AdderaTal" style="document" />
       <wsdl:input>
          <soap:body use="literal" />
       </wsdl:input>
       <wsdl:output>
          <soap:body use="literal" />
       </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
 <wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="AdderaTal">
       <soap12:operation soapAction="http://minwebbservice.se/AdderaTal" style="document" />
       <wsdl:input>
          <soap12:body use="literal" />
       </wsdl:input>
       <wsdl:output>
          <soap12:body use="literal" />
       </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="Service">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Min första
       WebService</wsdl:documentation>
    <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
       <soap:address location="http://localhost:1255/src/Service.asmx" />
    </wsdl:port>
    <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
       <soap12:address location="http://localhost:1255/src/Service.asmx" />
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Figur 2-16 WSDL-dokument för Web Servicen i Figur 2-11

WSDL-dokument består av följande komponenter [13, pp.84]:

- Definitions
- Types
- Message
- Operation
- PortType
- Binding
- Port
- Service

Figur 2-16 visar hur WSDL-dokumentet för Web Servicen i Figur 2-11 kan se ut. Dokumentet inkorporerar nämnda komponenter vilket kommer att förklaras översiktligt.

2.8.1 Definitions

Den välkända XML-deklarationen återfinns längst upp i WSDL-dokumentet i Figur 2-16 (rad 1). Det är ju i grund och botten ett XML-dokument. Elementet `definitions` skrivs så att definitioner för namnutrymmen finns med i dokumentet (se Figur 2-16 rad 2-7). Elementet är också dokumentets rotelement [13, pp.88].

2.8.2 Types

WSDL-dokument innehåller definitioner av datatyper som krävs för att kunna utbyta SOAP-meddelanden mellan klientapplikation och Web Service. Dessa datatyper definieras inom elementet `types` (se Figur 2-16 rad 10-28). WSDL-dokument använder XSD för definition av datatyper [13, pp.89,90].

2.8.3 Message

Elementet `message` definierar struktur på data som ska överföras mellan klientapplikation och Web Service [13, pp.92,93]. Barnelement till `message` är `part` vilket definierar logiska delar i ett meddelande som behöver kommuniceras mellan båda parter (klientapplikation och Web Service). Elementet `AdderaTal` som definieras inom types-elementet skickas som en parameter till `part` ”Parameters” (se Figur 2-16 rad 29-31). Elementet `part`, på rad 31 i Figur 2-16, associeras med den komplexa datatypen som definierar metoden `AdderaTal`. Varje inmeddelande som definieras för varje prototokoll (t.ex. SOAP) har också ett utmeddelande (se Figur 2-16 rad 32-34).

2.8.4 Operation

Web Service erbjuder metoder (operationer) som utför en specifik funktionalitet. För att definiera operationer så används elementet `operation`. Meddelanden som ska skickas mellan båda parter måste associeras med en specifik operation. Barnelement till operation är `input` och `output`. Elementet `input` tar inmeddelandet `AdderaTalSoapIn` som värde och elementet `output` tar utmeddelandet `AdderaTalSoapOut` (se Figur 2-16 rad 39,40) [13, pp.93,94]. WSDL-dokument stödjer fyra typer av operationer vilka är envägsoperation, notifieringsoperation, request-response-operation och ”solicit-response”-operation [13, pp.94,95].

2.8.5 PortType

Flera operationer kan ”grupperas” till en uppsättning operationer.
En sådan uppsättning definieras med elementet portType. En portType kan användas för att associera operationer med specifika ”protokoll” som t.ex. SOAP, HTTP GET och HTTP POST (se Figur 2-16 rad 35) [13, pp.95].

2.8.6 Binding

En `portType` som grupperar ett antal operationer kan associeras med nämnda protokoll med elementet `binding` (se Figur 2-16 rad 43 [13, pp.96].

2.8.7 Port

Elementet `port` specificerar adressen för en `binding` (t.ex. i form av en URL) så att en klient kan kommunicera med Web Servicen [13, pp.98,99]. Se Figur 2-16 rad 70-72. WSDL-dokument får innehålla flera `ports`.

2.8.8 Service

Flera `ports` kan ”grupperas” och då används ett `Service`-element för detta ändamål [13, pp.99]. Se Figur 2-16 rad 67,76.

2.9 Universal Description Discovery and Integration

UDDI vilket är en förkortning av ovanstående titel är en industristandard ursprungligen utvecklad av Ariba, IBM och Microsoft. UDDI definierar en registertjänst/katalog som används för att registrera, publicera och söka efter Web Services [13, pp.72]. På senare tid har flera organisationer och företag börjat samarbeta och tillsammans vidareutveckla UDDI. Särskilda källor nämner siffror som mer än 300 st organisationer/företag [13, pp.72].

Som tidigare nämnts och förklarats lades UDDI Business Registry ned. Detta innebär dock inte att UDDI inte fortfarande stöds. Flera företag inkluderande Microsoft, IBM och SAP stödjer fortfarande UDDI i sina respektive produkter. Det kan t.ex. nämnas att Microsofts serveroperativsystem Windows Server 2003 har stöd för UDDI-protokollet [9]. Företag kan skapa både privata och publika UDDI-kataloger [1]. Att utföra sökningar i en sådan katalog liknar sättet att söka i sökmotorer på webben [13, pp.72].

Vad kan man då göra med UDDI?. UDDI gör att det blir enklare, och tar mindre tid att hitta Web Services. Men inte bara det. Funktionsmässigt kan man t.ex. publicera Web Services, registrera organisationer, registrera mjukvara som utnyttjar Web Services, skapa förfrågningar för existerande eller ickeexisterande Web Services och lokalisera Web Services hos tredje part [13, pp.81]. Publicering av Web Services i en UDDI-katalog innebär att man först registrerar den dvs. anger information som beskriver Web Servicen, vilka metoder som finns och hur man ska kommunicera med Web Servicen [13, pp.72]. Innan registrering av Web Service kan ske i UDDI-kataloger behövs dock detaljer om organisationen/företaget som t.ex. namn, kontakuppgifter etc. [13, pp.81]. Att man registrerar mjukvara som använder en särskild Web Service innebär att man bl.a. anger teknisk information om Web Servicen [13, pp.81].
En av UDDI:s stora fördelar är att man kan lokalisera Web Services hos tredje part, dvs. externa organisationer/företag, med förutsättningen att de har ett publikt UDDI-register.

3 Genomförande

3.1 Inlärningsfas

Att utveckla en webbapplikation för informationshantering i projekt för Gain IT Huskvarna, har varit mitt mål. Men att bygga en sådan applikation har krävt mycket mer än bara programmering och design av Web Service och tillhörande webbsideprogrammering. Eftersom man också önskade att det ska vara enkelt och smidigt att uppdatera dvs. bygga vidare på applikationen har jag även använt mig av teknologier som XML, XSD och XSLT. Därför var jag tvungen att speciellt inhämta kunskap från böcker som behandlar dessa ämnen. Böcker som jag har använt mig av inkluderar:

- Professional Asp.Net 2.0
- Professional Asp.Net 2.0 XML
- Elektronisk bok: XML Web Services Professional Projects

Författaren kunde inte enbart använda dessa källor för specifika problem. Han var tvungen att också granskat mycket av dokumentationen, ”MSDN library for Visual Studio 2005” som medföljer Visual Studio 2005. Sökning på webben med hjälp av sökmotorn www.google.se har också skett vid flertalet tillfällen.

Allteftersom författaren utvecklade projekthanteringsystemet fick han bra synpunkter och ideèr till förslag till förbättringar från personer på utvecklingsavdelningen. Författaren anser att det har behövts ett stort mått av kreativitet och problemlösningsförmåga för att finna lösningar på de problem som han har stött på.

3.2 Utrustning

En dator med operativsystemet Microsoft Windows XP Professional har använts. Förutom det har ett antal teknologier utnyttjats vilket inkluderar:

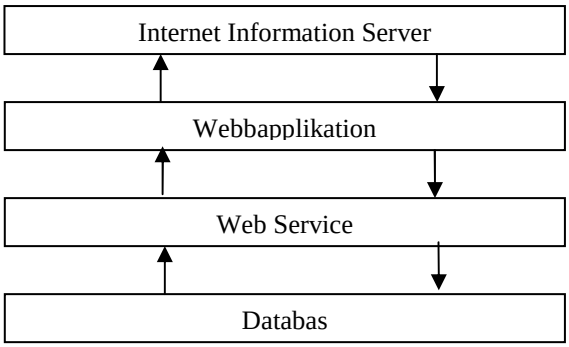
- Microsoft Visual Studio 2005
- Microsoft Sql Server 2000
- Microsoft Internet Information Server (IIS)

Förutom nämnda verktyg/teknologier har även verktyg som ingår i samband med installation av Microsoft Sql Server 2000 använts.

3.3 Projekthanteringssystemet

Webbapplikationen (projekthanteringssystemet) som har designats och utvecklats för Gain IT, Huskvarna, har byggts med tanke på tjänsteorienterad arkitektur. Det innebär att en Web Service anropas då kommunikation med underliggande databas ska ske (se

Figur 3-1). Andra teknologier/protokoll som utnyttjats är XML, XSD och XSLT i mycket hög grad. Dessa protokoll har använts för att det ska vara enkelt att bygga vidare på applikationen. När nya kommentartyper ska adderas till originalsystemet behöver applikationen då inte kompileras om. Detta har således orsakat mycket fungeringar och tankemöda. För att få svar på några av mina funderingar diskuterade jag med personer från Gain IT:s utvecklingsteam. Eftersom det fanns en tidigare version av ett projekthanteringsystem som dock inte var komplett har jag utnyttjat specifika delar av den programmeringslogik och webbsidor som ligger bakom den.
Den databas som jag utnyttjat har tidigare utvecklats av personal på Gain IT. Modifieringar och tillägg av den har dock behövts då bl.a. lagrade procedurer (sql stored procedures) inte fullt ut har funnits. Att jag har kunnat använda detta som bas har lett till en högre utvecklingstakt då det tar en ansenlig tid att lära sig XML, XSD, XSLT och Web Service fullt ut.



Figur 3-1 Grafisk representation av det uppbyggda systemet

En av mina frågor var hur man genererar ett grafiskt gränssnitt utifrån data i XML-dokument. Information som användaren anger sparas nämligen som xml-dokument i bakomliggande databas. XSLT (XSL lagras i databas) kunde ju naturligtvis användas för detta då det kan konvertera ett dokument av en typ till en annan typ (HTML). Men det är inte enbart HTML som genereras av webblösningen utan det är även en hel del ASP.NET-specifik formateringskod (ASP.NET-taggar).

Detta innebär att ASP.NET:s egna motor måste kompilera resultatet av en XSLT-konvertering. Programmeringsmässigt har detta varit det svåraste problemet eftersom data ”laddas in i” webbsidor på ett annorlunda sätt. Hanteringen av den resulterande HTML och ASP.NET-koden blev mer avancerad än brukligt eftersom bakomliggande applikationslogik är starkt obunden till den data och grafiskt gränssnitt som presenteras. Problemet löstes dock efter mycket tankemöda.

Ett annat problem som jag i ett tidigt skede konfronterades med var hur man genererar XML utifrån grafiska komponeter (ASP.NET-taggar). Hur kan man veta vad användaren har angett som inmatning? Det görs nämligen inga antaganden om vilka grafiska komponenter som är resultatet av en XSL-transformering. Data som anges måste hamna inom rätt XML-element/attribut. Detta problem löstes genom att utforma speciell logik som vid run-time (exekvering) tar reda på vilken data som användaren angett och med XSL-attribut, från XSL-transformeringen, parar ihop värde som angetts med korrekt XML-element/attribut. XSD-schema bäddas även in osynligt i webbsidan (aspx-sidan) så att ett XML-dokument med korrekt struktur kan skapas utan att behöva hämta associerat XSD-dokument från databas.

Ett annat problem som jag har haft var hur man kopplar applikationslogik till händelser (händelsehanterare) som sker för grafiska komponenter på en webbsida. Vi vet ju som bekant inte vilka grafiska komponenter som finns. Lösningen på det problemet var inte så svår. Speciell logik undersöker vilka komponenter som finns på webbsidan och avgör vilka händelsehanterare som ska kopplas ihop med grafiska komponenter.
Hur Web Service fungerar och används på bästa sätt har även varit föremål för funderingar. Problemen har lösts genom att studera den elektroniska boken, XML Web Services Professional Projects, vilket har varit till stor hjälp.

3.3.1 Webbsidor

De webbsidor som har producerats är `Default.aspx` och `installningar.aspx`. På sidan `Default.aspx` finns kod som hanterar visning och redigering av befintlig kommentar, oberoende av typ (även borttagning), men även kod för att med det grafiska gränssnittet kunna skapa nya kommentarer.
Sidan `installningar.aspx` används för att kunna lista alla kommentarer under en särskild aktivitet, checka in, checka ut och redigera samt skapa nya kommentarer.

3.3.2 XSL-kod

Tre kommentartyper har skapats med XSL. De är kontakt, anteckning och telefonkontakt. Dessa kommentartyper lagras som ren text i databasen.
Typen kontakt används för att lagra kontaktuppgifter till exempelvis ett företag, myndighet eller organisation. Den andra typen (anteckning) kan användas för att hantera information som berör uppgifter t.ex. som en minnesanteckning. Den sistnämnda typen utnyttjas för att lagra information som berör samtal mellan individer på t.ex. olika företag, en slags bokföring av samtalets innebörd.

3.3.3 XSD-kod

För att kunna skapa korrekta XML-dokument har ett XSD-dokument skapats för varje kommentartyp. XSD-dokumenten beskriver vilka element som får finnas med, datatyp, längd (antal tecken), vilken ordning de får förekomma etc.

3.3.4 Klasser

För att kunna utföra diverse operationer på nämnda webbsidor och funktionalitet i samband med det har ett antal klasser skapats, dessa är:

- `XsltProcessor`
- `XmlValidator`
- `FaltValidator`
- `GrafikLage`
- `Globaler`
- `Loggning`

Syftet med klassen `XsltProcessor` är att kunna konvertera XML-dokument till andra format med XSL och en XSLTprocessor. Detta är själva kärnkomponenten när det gäller konvertering av XML till andra typer av markup-språk. För att kunna validera XML-dokument mot XSD-dokument används klassen `XmlValidator`. Det är alltid av stor betydelse att kontrollera användarens inmatning så att inte t.ex. data utelämnats. Det är syftet med klassen `FaltValidator` – att validera indata i webbformulär.
För att byta vy t.ex. visning, redigering eller skapa nytt används klassen `GrafikLage`. Globala variabler som används i applikationen har samlats i klassen `Globaler` på ett strukturerat och bra sätt. När fel inträffar i programkoden så loggas de. Detta sker med hjälp av metoder i klassen `Loggning`.

3.3.5 Web Service

Web Servicen har ett antal metoder för att hantera databasen. De funktioner som de utför är bl.a. uppdatering av kommentar, borttagning av särskild kommentar (tas inte bort i databas men syns inte i vänstermenyn), tillägg av nya kommentarer, skapa ny aktivitet, ta bort befintlig aktivitet och hämta enstaka kommentar eller flera kommentarer.

3.3.6 Lagrade sql-procedurer

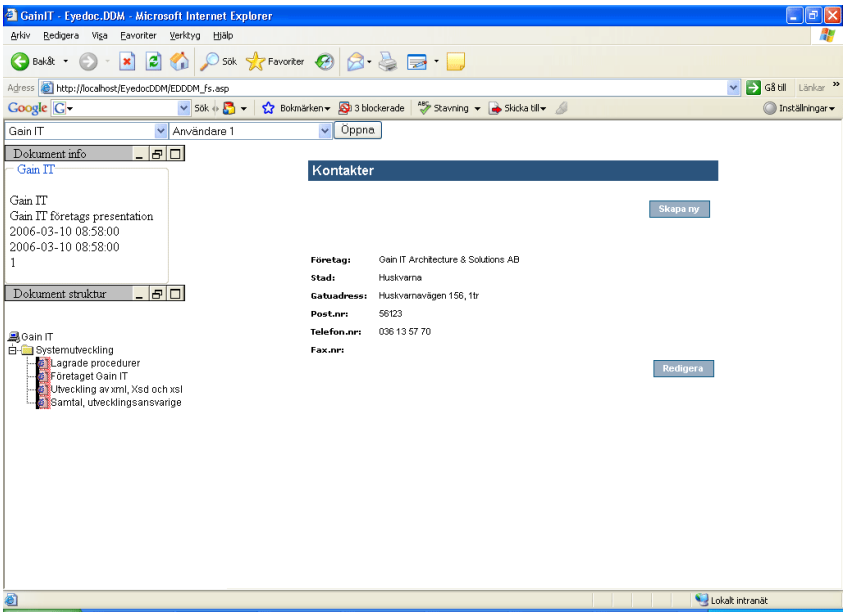
De lagrade sql-procedurerna (stored procedures) vilket anropas av Web Servicen är modifierade och några är tillagda. Nya lagrade procedurer inkluderar `SP_EDDDM_Objekt_Dolj` och `SP_EDDDM_ObjektTyp_Hamta`. Den första tar bort (döljer) en kommentar och den andra hämtar namn på kommentartyperna. Eftersom databasen från början är byggd av Gain IT på ett särskilt sätt så behövs inte så många lagrade procedurer. Däremot så blir varje lagrad procedur större kodmässigt.

4 Resultat

Examensarbetet har utmynnat i en välfungerande webbapplikation för hantering av information i projekt. Bristen på tid och omfattningen på arbetet har dock begränsat vilken funktionalitet som finns med.
Ytterligare funktionalitet som krävs för att det ska gå att skapa projekt från början saknas. Kärnfunktionaliteten i webbapplikationen (skapa aktivitet, skapa/ta bort/redigera kommentar, checka in, checka ut etc.) fungerar dock mycket väl. En mycket bra s.k. finess med webbapplikationen är att den är enkel att uppdatera med nya kommentartyper mycket tack vare användning av XML, XSD och XSL.
Författaren anser att webbapplikationen och projektet har utförts tillfredställande. Uppdragsgivaren är också nöjd med åstadkommet resultat.

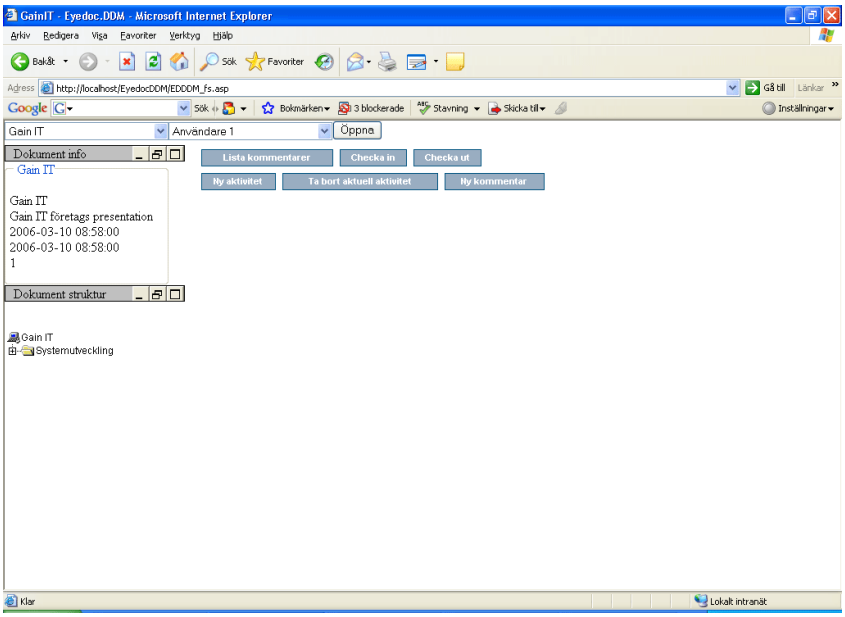
När en användare utnyttjar webbapplikationen kan denne enkelt skapa nya aktiviteter för ett projekt. Aktiviteter kan skapas på olika nivåer dvs. underaktiviteter kan skapas för en föräldraaktivitet.
Kommentarer av tre olika typer, kontakt, telefonkontakt och anteckning, kan skapas under en särskild aktivitet.
För att kunna gå tillbaka i tiden (versionshantering) och spara varje ändring av kommentarer checkas kommentarer under en särskild aktivitet in (ändrad kommentar kan ses på rödaktig ikon i vänstermenyn). Detta innebär att nya poster skapas i databasen för varje incheckning som utförs. För att sedan ha möjlighet att redigera/skapa nya kommentarer checkar användare ut en aktivitet. Versionshanteringen finns dock inte skapad så att en användare kan välja detta på webbsidorna. Utvecklare kan dock se dessa ändringar i databasen. Versionshanteringsfunktionen skulle dock kunna ha skapats (för användare på webbsidorna) om det hade funnits mer tillgänglig tid.

För att lista alla kommentarer under en särskild aktivitet kan en användare navigera till sidan `installningar.aspx`. På den sidan kan användare välja att bl.a. se alla kommentarer som tillhör en särskild aktivitet. Det är även möjligt att skapa nya samt redigera befintliga kommentarer.



Figur 4-1 Webbapplikationen, översiktsbild, sidan Default.aspx

Figur 4-1 visar webbapplikationen översiktligt (`Default`.aspx). I vänstermenyn ses kommentarerna under aktiviteten ”Systemutveckling”. De rödaktiga ikonerna framför kommentarerna i vänstermenyn signalerar att kommentarerna under aktiviteten inte är incheckad. Vänstermenyn är inte personligen utvecklad av författaren. Den finns sedan tidigare utvecklad av Gain IT. Även de två rullningslistorna och knappen ”Öppna” är framtaget av Gain IT sedan tidigare.

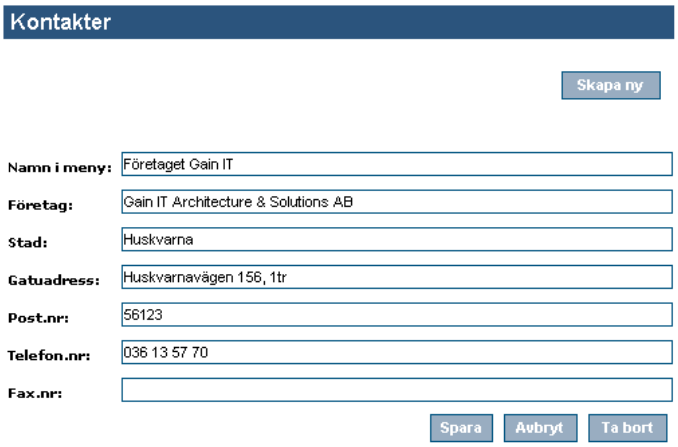


Figur 4-2 Webbapplikationen, sidan installningar.aspx

Figur 4-2 visar sidan `installningar.aspx`. Här kan man checka in, checka ut, lista kommentarer under en aktivitet, skapa ny aktivitet och skapa nya kommentarer.

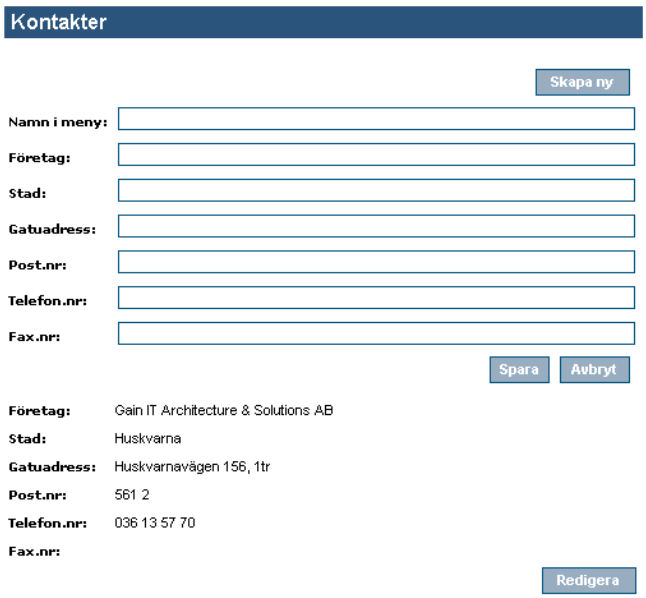
Figur 4-3 Kommentartypen kontakt i visningsläge

Figur 4-3 visar kommentartypen kontakt i visningsläge. Som kan ses i figuren kan användare enkelt redigera kontakten genom att klicka på knappen ”Redigera” (exempeldata visas i Figur 4-3 och i övriga figurer). För att skapa en ny kontakt kan tryck på knappen ”Skapa Ny” göras..



Figur 4-4 Kommentartypen kontakt i redigeringsläge

Figur 4-4 visar kommentartypen kontakt i redigeringsläge. Här kan man enkelt ändra på informationen och sedan trycka på knappen ”Spara” när man är tillfredsställd med ändringen. Det finns även validatorer som validerar inmatning i textrutorna. De visas om data inte angetts. Önskar man inte utföra någon ändring kan man trycka på knappen ”Avbryt”. Om kontakten ska tas bort kan detta göras med tryck på knappen ”Ta bort”. Textrutan intill texten ”namn i meny:” avser namnet i vänstermenyn.



Figur 4-5 Kommentartypen kontakt i ”skapa ny”-läge

Figur 4-5 visar typen kontakt i ”tilläggs”-läge (”ny”-läge). Textrutor där data kan skrivas in visas. För att spara kontakten tryckes på ”Spara” annars på ”Avbryt” om någon ny kontakt inte ska skapas. Det finns en kontakt som visas under ”spara”-knappen vilket är en tidigare skapad kontakt.

Figur 4-6 Kommentartypen telefonkontakt i visningsläge

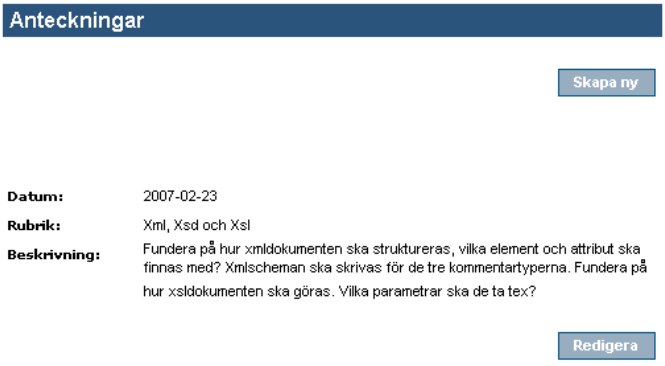
Figur 4-6 Visar typen telefonkontakt i visningsläge. Redigering och tillägg av telefonkontakt fungerar på samma sätt som kontakt.

Figur 4-7 Kommentartypen telefonkontakt i redigeringsläge

Figur 4-7 visar typen telefonkontakt i redigeringsläge. Att spara gjorda ändringar fungerar på samma sätt som för kommentartypen kontakt. Även på motsvarande sätt för att avbryta redigeringen och gå tillbaka till visningsläget.

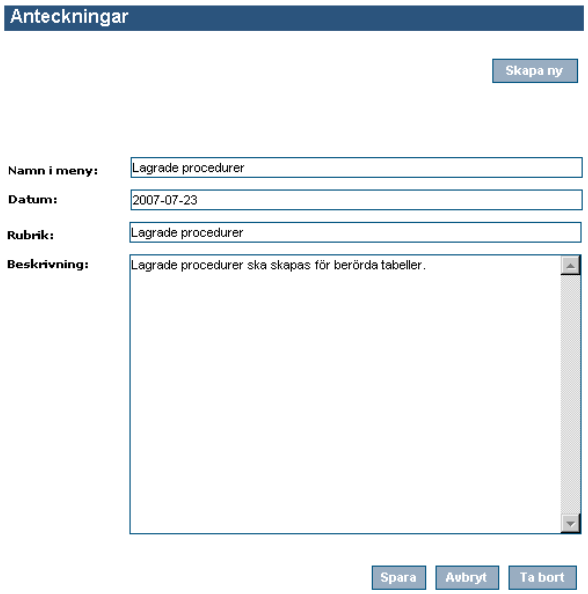
Figur 4-8 Kommentartypen telefonkontakt i ”skapa ny”-läge

Figur 4-8 visar telefonkontaktstypen i ”tilläggs”-läge (”ny”-läge). Information anges på samma sätt som för övriga kommentartyper och valideras även vid klick på knappen ”Spara”.



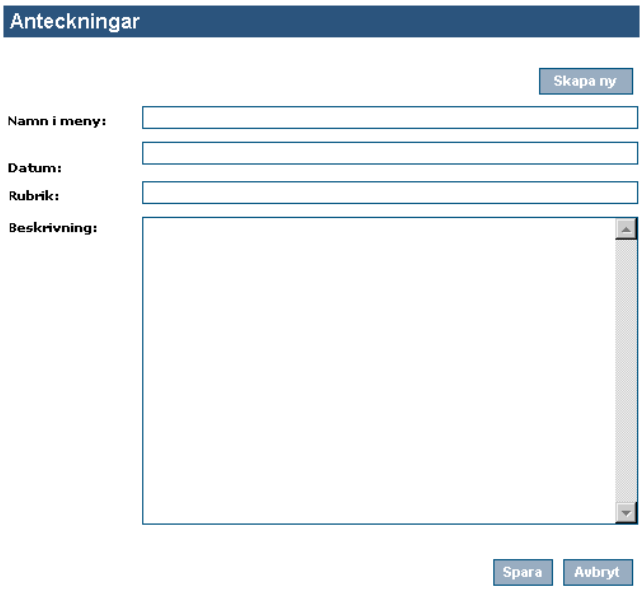
Figur 4-9 Kommentartypen anteckning i visningsläge

Figur 4-9 visar kommentartypen anteckning i visningsläge. Som förväntat fungerar redigering och tillägg av ny anteckning exakt likadant som för övriga typer.



Figur 4-10 Kommentartypen anteckning i redigeringsläge

Figur 4-10 visar en anteckning i redigeringsläge. Redigering av detaljinformation för en anteckning fungerar som övriga typer.



Figur 4-11 Kommentartypen anteckning i ”skapa ny”-läge

Figur 4-11 visar typen anteckning i ”tilläggs”-läge. Naturligtvis fylles detalj-informationen i på exakt samma sätt som övriga typer och funktionaliteten skiljer sig inte mot övriga typer.

5 Slutsats och diskussion

Projekthanteringssystemet som har utvecklats för Gain IT, Huskvarna, är byggt inte bara med slutanvändaren i åtanke utan även med tanke på de personer som ska vidareutveckla applikationen. Viktiga huvudkomponenter för webb-applikationen är XML, XSD, XSL och Web Service. XML används för att representera kommentarer strukturerat, XSD för att validera XML-dokument, XSL för att definiera det grafiska gränssnittet och Web Service för att skapa en god möjlighet till integration mot andra system. Eftersom XML, XSD och XSL lagras i bakomliggande databas av märket Sql Server 2000 kan dessutom webbapplikationen uppdateras med nya kommentartyper utan att applikationen behöver kompileras om.

När författaren introducerades med förslaget att bygga en webbapplikation med nämnda teknologier hade han inte mycket kunskap om nämnda ingredienser. Författaren har under arbetets gång studerat mycket information om XML, XSD, XSL och Web Service med största hjälp från böcker som tidigare nämnts. Ytterligare information för att förstå och hitta lösningar på specifika detaljproblem har krävts. Källor för den informationen inkluderar elektroniska artiklar (se Referenser), Microsofts egen produktdokumentation, ”MSDN library for Visual Studio 2005”, och naturligtvis sökning på http://www.google.se.

Författaren anser att webbapplikationen blev tilltalande och enkel att använda. Det finns dock ytterligare funktionalitet som hade behövts finputsats ännu mer. Även ny funktionalitet som t.ex. versionshantering åtkomlig från grafiskt gränssnitt, skapa nya projekt från början, skulle ha förbättrat systemet ytterligare. Varför de funktionerna inte adderades till systemet har att göra med tidsbrist och omfattning på ovannämnda teknologier.

Att jobba med nämnda teknologier har givit författaren ännu mer förståelse för att bygga webbapplikationer och system av kommersiell karaktär. Det finns som bekant en uppsjö av teknologier, verktyg och metoder som kan användas för systemutveckling med avseende på webben. Författarens personliga åsikt är av karaktären att teknologier som han har jobbat med kan få större betydelse i framtiden.

Författaren anser även att projekthanteringsystemet kan förbättras ännu mer i framtiden och att ny funktionalitet kan adderas enklare mycket tack vare teknologierna XML, XSD, XSL och Web Service men även pga. god programmeringsteknik har använts.

6 Referenser

[1] *FAQS*
http://www.uddi.org/faqs.html#whatis
(acc. 2007-04-06)

[2] *MSDN Magazine, The XML files, The Birth of the Web Services*
http://msdn.microsoft.com/sswebservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp
(acc. 2007-01-11)

[3] Thiru Thangarathinam (2006) *Professional ASP.NET 2.0 XML*, Wiley Publishing, Indianapolis, ISBN 0-7645-9677-2

[4] Thomas Erl (2005) *Service-oriented Architecture Concepts, Technology and Design*, Pearson Education, Upper Saddle River NJ, ISBN 0-13-185858-0

[5] *Simple Object Access Protocol (SOAP) 1.1*
http://www.w3.org/TR/2000/NOTE-SOAP-20000508
(acc. 2007-02-28)

[6] *UBR Shutdown FAQ*
http://uddi.microsoft.com/about/FAQshutdown.htm
(acc. 2006-12-19)

[7] *UDDI Business Registry*
http://www.uddi.org/find.html
(acc. 2006-12-19)

[8] *UDDI Business Registry - Discontinuation Notice*
http://www-306.ibm.com/software/solutions/webservices/uddi/
(acc. 2006-12-19)

[9] *UDDI Services*
http://www.microsoft.com/windowsserver2003/technologies/idm/uddi/default.mspx
(acc. 2007-04-06)

[10] *W3C, About the World Wide Web Consortium (W3C)*
http://www.w3.org/Consortium/
(acc. 2007-01-16)

[11] *Web Services and the Microsoft Platform*
http://msdn.microsoft.com/webservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/wsmsplatform.asp
(acc. 2006-12-19)

[12] *XML Web Services Basics*
http://msdn.microsoft.com/sswebservices/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp
(acc. 2006-12-19)

[13] Geetanjali Arora; Sai Kishore; NIIT (2002*) XML Web Services Professional Projects*, Muska & Lipman Publishing, ISBN 1-931841-36-5

7 Sökord

8 Ordlista

Attribut	Attribut läggs till i element för att ”markera” ytterligare information
Element	Strukturerar upp information i logiska komponeneter.
Web Service	Web Services är en typ av applikation som bygger på standardiserade protokoll (XML, XSD, SOAP, WSDL, UDDI etc.) för att utbyta information mellan it-system (servrar), oberoende av it-plattform (operativsystem), över Intranät och Internet
Xml-schema	XML-schema vilket utvecklades av W3C är ett dokument som skrivs med XML-syntax där man definierar strukturen för XML-dokument
XML-dokument	Strukturerad XML-data lagras i XML-dokument.
XSLT-dokument	XSL transformationer (XSLT) görs för att konvertera XML-strukturerad data till t.ex. HTML.