KTH Mikroelektronik
och informationsteknik

Migrering av befintliga Internetlösningar till den nya .NET-plattformen

Björn Blom

Den 15:e Juni 2004

Pocket Mobile Communications AB, Stockholm
Examensarbete 20 p

Handledare (PocketMobile): Fredrik Schmidt
Handledare och Examinator (IMIT, KTH): Thomas Sjöland

Abstract

The aim of this project is to improve/modernise the stationary part of a communication system (PreCom), developed by PocketMobile Communication AB, and investigate if it's possible to migrate the existing system to the new Microsoft .NET platform.

A lot of systems today, like PreCom, are written in ASP but future development will probably be focused on ASP.NET. As shown in this report there are simply too many advantages with ASP.NET to stay with ASP.

To understand the migration considerations and the implementations carried out later, this report includes a brief description of ASP and a more detailed description of Microsoft .NET framework, which includes ASP.NET, followed by a comparison between the two techniques.

Since ASP and it's successor ASP.NET are not 100% compatible with each other, there is a great chance for things to go wrong when the system is to be migrated. To minimize that risk it is required to make an investigation of the present system and decisions have to be done to choose the best migration strategy. Horizontal and vertical migration were two of the strategies that were considered to simplify the migration process and avoid migrating everything at once. As this report shows, another strategy, "page by page", combined with interoperability between ASP and ASP.NET often is the best choice. Performance is also an important parameter when considering migration. Therefore a performance test and a comparison between different migration approaches is performed. The result from this is that the 2-3 times performance boost that Microsoft claims will come from ASP.NET not always shows in realistic scenarios.

To solve the difficulties that arise from migration and to fulfil PocketMobiles needs, two implementations were carried out. The first is an ASP.NET application that is migrated from the old ASP application to test some different migration approaches and evaluate these. The second is a prototype of a new version of PreCom totally rewritten with ASP.NET and the use of the new techniques that it provides.

The result of this project shows that a total rewrite of an ASP-system are not realistic in most cases, but if the system isn't very complex and major architectural changes are needed then it could be a good alternative to migration, as in the case with PreCom. Otherwise migration with an appropriate strategy adapted to the actual application along with the use of a code migration program is the only alternative.

Sammanfattning

Syftet med detta projekt är att förbättra/modernisera den stationära delen av ett kommunikationssystem (PreCom), utvecklat av PocketMobile Communications AB, och utreda om det är möjligt att migrera det nuvarande systemet till Microsofts nya .NET-plattform.

Många av dagens system är i likhet med PreCom skrivna i ASP, men den fortsatta utvecklingen kommer troligtvis att ske i ASP.NET. I denna rapport visas att det vanligtvis finns alltför många fördelar med ASP.NET för att det ska vara rimligt att fortsätta utveckla i ASP.

För att förstå de faktorer som spelar in vid migrering samt senare utförda implementationer, innehåller denna rapport en kortfattad beskrivning av ASP samt en mer detaljerad beskrivning av Microsoft .NET framework, som inkluderar ASP.NET, följt av en jämförelse mellan de båda teknikerna.

Eftersom ASP och dess efterträdare ASP.NET inte är 100% kompatibla med varandra finns det en stor risk att ett migreringsprojekt fallerar. För att minimera den risken är det nödvändigt att göra en undersökning av det nuvarande systemet och beslut måste fattas för att välja den bästa migreringsstrategin. Horisontell- och vertikalmigrering var två strategier som övervägdes för att förenkla migreringsprocessen, samt undvika att migrera allt på en gång. Denna rapport visar att en annan strategi, ”sida för sida”, kombinerat med interoperabilitet mellan ASP och ASP.NET ofta är det bästa valet. Prestanda är också en viktig parameter att tänka på vid migrering. Därför genomfördes ett prestandatest samt jämförelse mellan olika migreringsstrategier. Resultatet från detta visar att den 2-3 gångers prestandaökning som Microsoft hävdar att ASP.NET ska ge upphov till, inte alltid visar sig i realistiska situationer.

För att lösa svårigheterna som uppstår vid migrering samt att fullgöra PocketMobiles önskningar, konstruerades två implementationer. Den första är en ASP.NET-applikation som är migrerad från den gamla ASP-applikationen för att testa några olika angreppssätt vid migrering samt bedöma dessa. Den andra är en prototyp av en ny version av PreCom som är helt omskriven i ASP.NET och använder sig av dess nya tekniker.

Resultatet av detta projekt visar att en fullständig omskrivning av ett ASP-system i de flesta fall inte är realistiskt, men om systemet inte är alltför komplext och är i behov av stora arkitekturella ändringar så kan detta vara ett bra alternativ till migrering som i fallet med PreCom. I annat fall är migrering med en lämplig strategi, anpassad till den aktuella applikationen, tillsammans med användandet av ett migreringsprogram det enda tänkbara alternativet.

Innehållsförteckning

1 Introduktion

Pocket Mobile Communications AB designar och utvecklar mjukvara för mobila enheter som tillåter dessa att kommunicera via handdatorer. Detta examensarbete syftar till att förbättra/modernisera PocketMobiles egenutvecklade kommunikationssystem PreCom (se 1.1 nedan) genom att bland annat undersöka migration till Microsofts .NET teknik. I dagsläget är den delen av PreCom som detta arbete kommer att behandla baserat på ASP. Det finns stora skillnader mellan ASP och ASP.NET. ASP är ett skriptspråk medan ASP.NET kan använda flera olika objektorienterade programmeringsspråk som t.ex. C# (C-sharp), J# (J-sharp), C++ och Visual Basic. Teoridelen i denna rapport kommer att beskriva teknikerna ASP och ASP.NET med tyngdpunkten på en jämförande studie syftande till att ge förståelse för hur man kan gå till väga vid migrering från ASP till ASP.NET samt beskriva problem, överväganden och alternativ vid migrering.

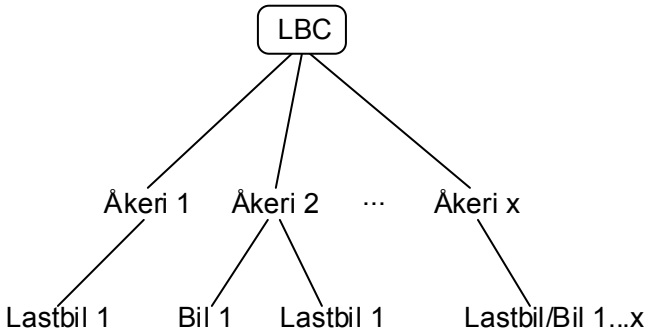
Eftersom det finns enormt många olika system i världen baserade på ASP så kommer denna rapport av tidsmässiga skäl begränsas till allmänna nyckelproblem samt den trelagersteknik (presentationslager, mellanlager och datalager) som PocketMobiles system PreCom använder sig av. Denna är troligtvis en av de vanligaste systemuppbyggnadsteknikerna som används. Dessutom kommer olika prestandatest att utföras eftersom detta kan vara en av huvudanledningarna till en migrering.

Det finns flera likheter mellan .NET framework (programmeringsmodellen i .NET) och Java. Därför kommer det att på flera ställen i rapporten finnas jämförelser med Java för att underlätta förståelsen för läsare som är bekanta med detta språk.

1.1 Kommunikationssystemet PreCom

Kommunikationssystemet PreCom kan delas upp i två huvuddelar. Den stationära delen, som för närvarande bara finns för budbilar, ska kunna köras via ett webbgränssnitt på t.ex. lastbilscentralen (Figur 1) eller budbilscentralen (inga åkerier, budbilarna ligger direkt under budbilscentralen). Den mobila delen körs på de mobila enheterna som vanligtvis är handdatorer monterade i bilar och lastbilar. Dessa handdatorer kör operativsystemet PocketPC som är ett subset av Windows CE (Compact Edition). I denna rapport kommer endast den stationära delen av PreCom att behandlas. Dock kommer viss kommunikation med den mobila delen att beaktas. En viktig del av funktionaliteten i PreCom är att lastbils- eller budbilscentralen ska kunna lägga order som skickas ut till en eller flera mobila enheter.

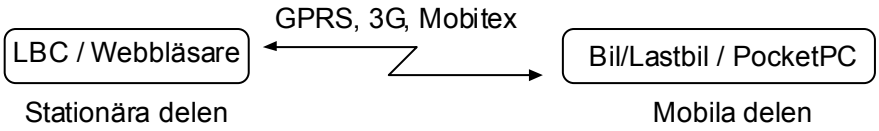
PreCom har idag ingen heltäckande stationär del som klarar flera olika typer av order utan förlitar sig i de flesta fall på kontorssystem från andra leverantörer som t.ex. Hogia och IBS.



Figur 1. *Exempel på hierarkin av åkerier, bilar och lastbilar under lastbilscentralen (LBC) LBC kan dessutom hyra in körningar från andra åkerier som ej tillhör LBC.*

1.1.1 Vision

Visionen är att PreCom alltid ska kunna användas på den stationära sidan. I dagsläget finns den stationära delen ”PreCom Bud” men denna kan endast användas för budservice. Därmed finns det ett behov att ”utöka” PreCom Bud så att även denna kan användas av lastbilscentralen (se Figur 2).



Figur 2. *Förenklad bild över kommunikationen mellan den stationära och mobila delen*

1.2 Bakgrund

Många företag står inför valet hur de ska gå till väga vid migrering till ASP.NET. Det finns en del information från Microsoft men den tenderar att inte särskilt väl belysa de problem som uppkommer samt att det i många fall kan vara helt orimligt att migrera i så stora steg som föreslås. Det dokument från Microsoft som jag anser vara bäst är ”Microsoft .NET/COM Migration and Interoperability” [3] av Steve Busby och Edward Jezierksi.

Som en motpol till Microsofts dokument finns ”Moving to ASP.NET – White Pages” [5], från företaget Consonica som tillverkar en produkt vid namn StateStich som tillåter delning av sessionsvariabler[1] mellan ASP och ASP.NET. Det är lite förvånande att ASP.NET inte har något som helst stöd för detta från början. I Consonicas dokument läggs stor vikt på de problem som kan uppstå vid migrering. Vad som inte är så konstigt är att de framhäver sitt eget verktyg som den bästa migreringslösningen och tonar ned andra lösningar. Oavsett om de har rätt i det eller inte så känns dokumentet i övrigt realistiskt och objektivt. Särskilt efter de egna erfarenheter jag haft med ”Implementation 1 – Migrering av orderhantering till ASP.NET” i kapitel 7.

Denna rapport är ett försök att göra en så objektiv betraktelse som möjligt samtidigt som den i implementeringsdelen speciellt tar hänsyn till PocketMobiles programvarulösningar.

[1] Sessionsvariabler används för att dela information mellan olika webbsidor som tillhör samma applikation (se 5.6.2)

Alternativen till migrering är att antingen inte migrera alls eller att göra en total omskrivning av all koden. Det senare innebär både stora nackdelar och stora fördelar. En omskrivning samt utökning av PreCom Bud genomförs i ”Implementation 2 – Omskrivning av orderhantering till ASP.NET” i kapitel 8.

1.3 Mål

Målen med detta arbete är:

- Att ge underlag för vilka delar av det gamla systemet skrivet i ASP, kallat PreCom Bud, som är fördelaktigt att migrera till ASP.NET, vilka migreringsalternativ som finns samt om något av dessa är rimligt att genomföra. Några av de aspekter som kommer att vägas in är prestanda, expanderbarhet och utvecklingsmöjligheter. För att testa olika tekniker och strategier framtogs Implementation 1 (se kap. 7).

- Att utveckla en prototyp för orderläggning i PreCom med .NET-teknik samt göra ett underlag för hur olika moduler kan anslutas. T.ex. en faktureringsmodul och en positioneringsmodul. Hur modulerna ska kunna kommunicera t.ex. med hjälp av Web Services. Prototypen är tänkt att kunna ersätta nuvarande orderläggningsdel av PreCom. Funktionaliteten av PreCom Bud kommer att ingå som en av delarna i denna prototyp. Utvecklingen av prototypen finns beskriven i Implementation 2 (se kap. 8).

1.4 Metod

För att uppfylla projektets mål var det nödvändigt att först sätta sig in i den gamla ASP-tekniken samt den nya .NET tekniken som används i ASP.NET. Samtidigt så undersöktes vilka skillnader som fanns mellan dessa. Dessutom så undersöktes hur ASP och ASP.NET kan samexistera i ett och samma system genom kommunikation och utbyte av variabler. Detta är nödvändigt om migrering ska kunna ske steg för steg.

För att kunna genomföra arbetet på en högre nivå och förstå vad som försiggår bakom kulisserna var en omfattande litteraturstudie nödvändig. Även vid felsökning och optimering är det nödvändigt med lite djupare kunskaper om hur .NET och ASP fungerar.

Mycket tid har gått åt till att sålla ut intressant och relevant information ur Microsoft jättelika informationsbibliotek Microsoft Developer Network (MSDN) som innehåller information om Microsofts produkter. Den mest använda delen av MSDN har varit ”Library” som innehåller dokumentation, tekniska artiklar samt referensguider. Den del av MSDN som dokumenterar .NET klasser är motsvarigheten till Suns klassdokumentation av java. Även andra .NET-relaterade webbsidor har varit av stort intresse.

1.4.1 Implementationer

För att uppfylla målet var det sedan nödvändigt att ta fram två implementationer. Under arbetets gång togs även bland annat lämpliga metoder fram för att mäta och jämföra prestanda, se kapitel 9.

- Den första implementationen visar i princip hur den gamla budservicedelen i PreCom kan migreras till .NET med så liten modifikation som möjligt. Samtidigt så testas olika migreringsstrategier. Den visar även prov på interoperatibilitet mellan ASP och ASP.NET samt COM[2] och .NET.

[2] COM (Component Object Model) är Microsofts teknologi för mjukvarukomponenter. COM används för kommunikation mellan mjukvaror.

- Den andra implementationen är byggd helt med .NET-teknik. Den är mer generell och omfattar förutom budservice även grus, container, lokal- och fjärrtransporter. I den visas tydligt hur produktiviteten ökar med ASP.NET.

1.4.2 Val av programspråk

Då den första implementationen handlade om migrering från ASP-sidor skrivna med VBScript, så var programspråket Visual Basic i .NET det naturliga valet för att minimera arbetet med att översätta koden. På så sätt var endast ett mindre antal ändringar nödvändiga.

I den andra implementationen valde jag att använda C# (uttalas C-sharp). Detta beroende på att vissa delar i programmet redan var skrivna i C# samt att syntaxen i C# är snarlik den i Java vilken jag var van vid. Min handledare Fredrik Schmidt på PocketMobile nämnde kompatibilitetsproblem mellan Suns Java och Microsofts relativt nysläppta Java variant J#. Detta tillsammans med att C# utvecklades speciellt med .NET framework i åtanke gjorde att jag valde C#.

1.5 Rapportens upplägg

Nedan följer en kort beskrivning av rapportens olika kapitel.

Kapitel 1 - Introduktion
Kapitel 1 ger en inledande introduktion av uppgifterna som ska lösas/utredas med mål och omfattning.

Kapitel 2 - Webbprogrammering
Kapitel 2 syftar till att ge en grundläggande förståelse av hur webbaserade system fungerar samt varför det är motiverat att använda dessa.

Kapitel 3 Active Server Pages (ASP)
Kapitel 3 ger en kort beskrivning av ASP, som används i PocketMobiles gamla orderhanteringsprogram PreCom Bud.

Kapitel 4 - Microsoft .NET
Kapitel 4 beskriver Microsoft .NET-platform och .NET-framework med jämförelser till ASP där detta är möjligt och intressant i migreringssyfte.

Kapitel 5 - Jämförelse mellan ASP och ASP.NET
I kapitel 5 görs en jämförelse med tanke på migrering från ASP till ASP.NET. Semantiska skillnader som är nödvändiga att känna till vid migrering, speciellt från VBScript i ASP. Andra skillnader som jämförs är t.ex. objekttyper, data access, autentisering och utbyte av variabler.

Kapitel 6 - Migreringsstrategi
I kapitel 6 ges en introduktion till olika migreringsstrategier som testats i Implementation 1 samt utvärderats i kapitel 9.

Kapitel 7 - Implementation 1 – Migrering av orderhantering till ASP.NET
I kapitel 7 görs ett implementeringstest av olika migreringsstrategier för ASP-applikationen PreCom Bud. Teknikerna som används beskrivs i kapitel 2 - 6.

Kapitel 8 - Implementation 2 – Omskrivning av orderhantering till ASP.NET
I kapitel 8 beskrivs utvecklingen av ett orderhanteringssystem från grunden med delar av funktionaliteten som är ekvivalent med budservice-delen i PreCom (PreCom Bud). Teknikerna som används beskrivs i kapitel 2 och 4.

Kapitel 9 - Analys, utvärdering och jämförelse
I kapitel 9 analyseras och diskuteras de olika migreringsalternativen. Dessutom genomförs ett jämförande prestandatest då prestanda alltid bör vägas in vid migrering.

Kapitel 10 - Slutsats
Kapitel 10 Redogör för rapportens slutsatser.

Kapitel 11 - Referenser
I kapitel 11 redovisas de referenser som har använts i rapporten.

Kapitel 12 - Appendix A
I kapitel 12 listas förkortningar som använts i rapporten.

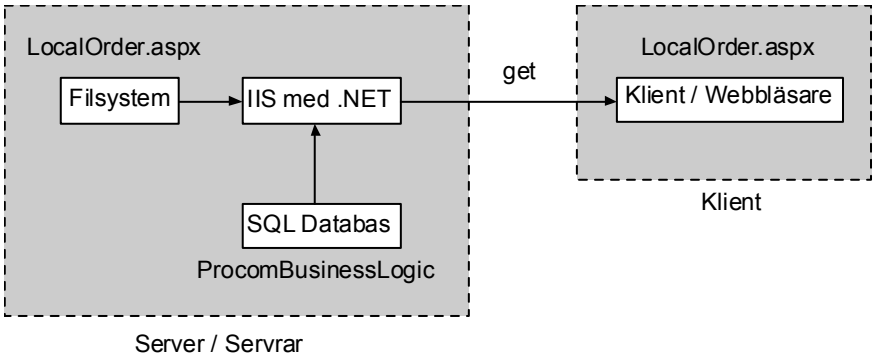
Kapitel 13 – Appendix B
Kapitel 13 innehåller en ”tracelogg” från körning av en sida konstruerad i Implementation 2.

2 Webbprogrammering

I detta kapitel beskrivs grundläggande kunskaper relaterade till webbprogrammering som är nödvändiga för förståelse av efterföljande kapitel. Med webbprogrammering avses i den här rapporten programmering för webben med ASP.NET. Liknande förfaringssätt måste användas vid programmering för webben med andra tekniker än ASP.NET. All programmering som utförts under arbetets gång i de båda implementationerna har varit webbrelaterade så detta ämne kommer att vara genomgående för hela rapporten.

2.1 Internet Information Services (IIS)

Internet Information Services 5.0 (IIS) är en webbtjänst i Windows 2000 som bland annat innehåller en webbserver och en ftp-server. Som standard kan den interpretera ASP-sidor (sidor med filändelsen .asp). Efter installation av .NET kan den även interpretera och exekvera ASP.NET-sidor (sidor med filändelsen .aspx). Figur 3 visar hur en begäran av en .aspx-sida kan gå till.



Figur 3. *Klienten begär att få filen LocalOrder.aspx från webbplatsen. LocalOrder.aspx processas av IIS:en som med hjälp av .NET kör tillhörande kod och skapar html-sidan LocalOrder.aspx. Observera att LocalOrder.aspx inte är densamma på klienten som på servern. LocalOrder.aspx på servern innehåller information (bland annat specificering av så kallade ”webcontrollers”) för att IIS:en/.NET dynamiskt ska kunna skapa html-sidan LocalOrder.aspx som skickas till klienten.*

2.2 HTTP - Tillståndslöst kommunikationsprotokoll

All kommunikation till och från klienten sker med HTTP som är ett tillståndslöst kommunikationsprotokoll, vilket betyder att varje förfrågan sker oberoende av andra och föregående förfrågningar. HTTP är ett så kallat förfrågan/svar protokoll som finns beskrivet i RFC-2616 [13]. Klienten skickar en förfrågan till servern innehållande URI, protokollversion och ett MIME-liknande meddelande. Servern svarar med statusinformation (lyckad/ej lyckad förfrågan), ett MIME-liknande meddelande med serverinformation, metainformation samt möjligt ”body” innehåll.

2.2.1 GET och POST

Det finns olika metoder i HTTP som medger olika möjligheter att skicka eller motta information. De vanligaste är GET för att hämta en sida samt POST (används ofta vid forms, se 2.3) för att skicka information tillbaka till servern. Vid POST anges vilken sida (.aspx-fil i ASP.NET) som ska ta emot och processa informationen.

2.3 Formulär (”Forms”)

Forms eller formulär som det kallas på svenska används för att användaren ska kunna fylla i ett antal uppgifter och/eller göra ett antal val som sedan skickas tillbaka till servern med POST metoden. Det kan t.ex. vara ett antal textrutor på en webbsida där man fyller i namn, adress, telefonnummer och ett större fält för synpunkter. Sedan klickar användaren på en knapp (submit) för att sända iväg uppgifterna till servern som sedan kan e-posta uppgifterna till en förbestämd e-post adress. I ASP.NET finns Web Forms (se 4.8.1) som gör det lätt att skapa mer avancerade formulär med bland annat klasserna DropDownList och Datagrid vilka båda kan bindas till innehållet i t.ex. en databas.

2.4 Tekniker för lagring av tillstånd

Med kunskapen om den tillståndslösa naturen hos webbsidor är det upp till utvecklaren av ett webbprogram att få det att kännas som ett icke tillståndslöst system för användaren. Till sin hjälp finns ett antal olika tekniker för att servern bland annat ska kunna autentisera och identifiera användaren. Ett exempel är *cookies*, vilka ofta lagras som små filer på klienten innehållande t.ex. session ID som är en unik identifikation av en klientanvändare. Session ID skickas över till servern vid varje förfrågan och servern kan sedan agera utefter bestämda regler för just denna användare.

Utveckling av program för webben som körs i en webbläsare[3] får i och med klient/server uppbyggnaden samt HTTP protokollet en annorlunda struktur än vanliga program som körs lokalt. Med Microsofts .NET framework och Visual Studio .NET (VS.NET) så har skillnaden mot traditionell programmering krympt jämfört med de flesta andra webbprogrammeringstekniker såsom ASP och PHP. Dock måste man fortfarande själv bestämma var kortlivad respektive mer långlivad information ska sparas. Eftersom endast en sida exekveras i taget så kommer variabler ej att vara tillgängliga mellan olika sidor utan enbart på den sida som exekveras och under den tid som exekveringen pågår. Variabler som man vill använda för att kommunicera mellan sidor eller användare måste lagras i något av de tre tillstånden (engelska: states), eller på annat sätt lagras i databas eller liknande, som finns i ASP.NET (se 5.6).

2.5 Klient- och serverskript

Som namnen antyder så är ett klientskript en programsnutt som exekveras på klientsidan medan ett serverskript exekveras på serversidan. Vid migrering så berörs endast skript på serversidan eftersom det endast är där som .NET verkar. För klientsidan krävs som tidigare nämnts endast en vanlig webbläsare (systemen i detta arbete är optimerade för Internet Explorer).

Följande exempel är från en .aspx-sida (webbsida för ASP.NET).

Exempel på ett klientskript: (Samma kommmentarstecken som för html ”<!--” och ”-->”)

```
<script language=javascript>
<!--
   Koden placeras här...
-->
</script>
```

[3] Det mesta av programmeringskoden körs normalt på servern medan enklare javascript körs på klienten för att förbättra funktionaliteten kring användargränssnittet.

Exempel på ett serverskript: (Samma kommenteringstecken som för det programspråk som används. I detta fall C# som använder ”//”)

```
<script runat=server>
   // Koden placeras här...
</script>
```

Istället för <script runat=server> kan ”script-taggarna” <% och %> användas, vilka härstammar från traditionell ASP. I ASP.NET behövs i många fall inga klientskript skapas manuellt då dessa automatiskt genereras vid rendrering av olika komponenter vilket senare tas upp i kapitel 4.8.2.

2.6 Motivering för webb-baserade program/system

Anledningen till att köra webb-baserade program är enkelhet för både kund och säljare. För ett företag som säljer varor över Internet är fördelarna uppenbara. Att motivera att utveckla ett kontorssystem, med vilket här menas ett system som körs på den stationära sidan, för ett kommunikationssystem är inte lika självklart. Även i detta fall innebär ett webb-baserat system många fördelar. T.ex. så är det enda ett litet företag behöver för att komma igång en dator med Internetuppkoppling samt login och lösenord till systemet. Olika användare får givetvis tillgång till sina egna databaser efter inloggningen. Databas och backup kan tillhandahållas av säljaren. Detta medför att installation och underhållskostnaderna för kunden blir minimala. Som exempel kan nämnas att licensen för Microsoft SQL-server kostar ca 50 000 kronor per processor. Större kunder som har egen it-personal och databasservrar kan välja att vara värd för systemet själva men samtidigt använda samma programvara. Uppdateringen av programvaran är även den mycket enkel särskilt om säljaren tillhandahåller systemet.

3 Active Server Pages (ASP)

Eftersom ett av huvudmålen med denna rapport är att utreda migrering av en ASP-applikation till ASP.NET (se 6 och 7) så kommer detta kapitel ge en kort beskrivning av ASP som används för många äldre webbapplikationer.

En ASP fil är en HTML-fil med inkluderade skript som exekveras på servern och översätts till HTML i filen som skickas till klienten. Skriptkoden placeras mellan <% och %> i ASP filen. ASP är föregångaren till ASP.NET som är en del av Microsoft .NET framework. En ASP-sida är en vanlig textfil med filändelsen .asp. Skriptkod och html-kod blandas på samma sida. Två skriptspråk medföljer ASP, VBScript [10] och Jscript. VBScript är baserat på Visual Basic och Jscript är baserat på Java. Andra skriptspråk som t.ex. Perl och Python kan också användas om man installerar en COM-kompatibel skriptmotor till dessa. Webservern bearbetar .asp sidan och all inbäddad skriptkod exekveras på servern. HTML-sidan som fås efter exekveringen skickas till klientens webbläsare som inte behöver exekvera någon kod förutom eventuella klientskript.

Exempel på en asp-sida som genererar en html-sida med texten ”Hello World” (HelloWorld.asp):

```
<%@ language="VBScript"%>
<html>
<body>
<%
Response.Write("Hello World!")
%>
</body>
</html>
```

Beskrivning av koden i exemplet ovan: Första raden anger att programmeringsspråket som skall användas är VBScript (Visual Basic Script). Andra och tredje raden innehåller ren HTML-kod eftersom dessa står utanför ”script-taggarna” (<% %>). Femte raden kör metoden Write i Response-objektet[4]. Denna skriver ut text på den genererade HTML-sidan. De två sista raderna innehåller precis som tredje raden statisk HTML-kod.

3.1 COM

COM är Microsofts teknologi för mjukvarukomponenter. Det används för kommunikation mellan olika mjukvaror, d.v.s. kod som är skrivna i olika språk.

Eftersom VBScript och andra scriptspråk som kan användas tillsammans med ASP ej är objektorienterade samt relativt begränsade när det gäller funktionalitet så kombinerar man ofta ASP med olika COM objekt, vilka kan vara egenkonstruerade eller köpta från en tredjepartsleverantör. COM objektet innehåller ofta affärslogik[5] och är vanligtvis utvecklat i Visual Basic. Att skriva ASP-sidan i ett skriptspråk för att därifrån kommunicera med ett icke skriptspråk via COM kan kännas lite som en omväg. Två av fördelarna med Microsofts nya .NET teknologi som beskrivs i nästa kapitel är den betydligt större funktionaliteten som finns

[4] Observera att man kan köra metoder i objekt samt att man kan skapa objekt i ASP. Dessa är dock inte skrivna i VBScript utan vanligtvis i Visual Basic. I ASP finns sex inbyggda COM-objekt som ej behöver instantieras. Dessa är Server, Request, Response, ObjectContext, Application och Session.

[5] Affärslogik (engelska: BussinessLogic) ingår i mellanlagret i en flerlagersarkitektur (engelska: n-tier), se Figur 8, och innehåller bland annat metoder för att hämta och uppdatera data i databasen.

i klassbiblioteket samt att allt skrivs i ett objektorienterat språk (spelar inte så stor roll vilket då samtliga språk kompileras till Microsoft Immediate Language (MSIL), som är gemensamt för samtliga språk i .NET och kan jämföras med Javas bytekod, se 4.4). Det innebär att man i många fall inte behöver använda COM objekt (med undantag för migrering som beskrivs i kapitel 6) utan kan utveckla allt i .NET.

3.1.1 ActiveX Data Objects (ADO)

ADO är ett bibliotek med olika ActiveX-objekt[6] för åtkomst till databaser. I biblioteket finns bland annat RecordSet-objektet som innehåller resultatet av en databassökning samt Connection-objektet som skapar en anslutning till en databas. ADO är föregångaren till ADO.NET som beskrivs i kapitel 4.9. Eftersom ADO och ADO.NET skiljer sig en hel del måste detta beaktas vid en migrering. För att inte behöva lägga ner alltför mycket tid på att skriva om ADO-anrop till ADO.NET-anrop så fortsätter man vanligtvis att använda ADO i ASP.NET, vid en migrering, då .NET kan kommunicera med COM-objekt via en så kallad RCW som beskrivs i kapitel 5.1.1. En jämförelse mellan ADO och ADO.NET ges i kapitel 5.3.

[6] Ett ActiveX-objekt är en typ av COM-objekt. Ett ActiveX-objekt implementerar vanligtvis förutom ”interfacet” ”IUnkown” ett antal ”standard-interface” för bland annat inbäddning och användargränssnitt.

4 Microsoft .NET

Microsofts nya, stora mjukvarusatsning .NET-plattform[7] innefattar allt ifrån klienter och ”services” till servrar. I plattformen ingår bland annat programmeringsmodellen .NET framework, SQL server, Windows 2000/XP/CE, Visual Studio.NET och XML Web Services.

En av huvudtankarna med Microsofts .NET teknik är att det ska vara enkelt att koppla samman olika system med varandra genom t.ex. XML Web Services (se 4.8.3)

Något som kännetecknar flera av Microsofts tidigare system är oviljan att skapa öppna standarder och dela med sig av källkoden. I fallet med .NET har Microsoft valt en annan, troligtvis smartare, strategi där snarast det motsatta gäller. På så sätt har man öppnat systemet för forskarvärlden samtidigt som man underlättar för utvecklare av bland annat verktyg för .NET.

I januari år 2002 släpptes den första officiella versionen av programmeringsmodellen Microsoft .NET framework. Denna bygger på ett antal publika standarder som finns publicerade och fritt tillgängliga på standardiseringsorganet Ecmas webplats [1, 2]. Microsoft har även släppt källkoden till en fungerande implementering av ECMA CLI [2] och ECMA C# [1] programspråk specifikation. Denna samling källkod kallas Shared Source Common Language Infrastructure (SSCLI) [4] och får laddas ner och användas relativt fritt för icke kommersiella ändamål. SSCLI kompilerar under Windows 2000, Windows XP och FreeBSD. Många av källkodsfilerna i SSCLI är daterade år 1998 vilket visar att utvecklingen av .NET har pågått en längre tid.

Både ”runtime-filerna”[8] (filer som behöver installeras på den dator .NET applikationer avses köra) och utvecklingsverktyg (bland annat kompilator, assembler, dissassembler samt debugger) är gratis. Man kan således skriva .NET och ASP.NET program enbart med hjälp av en texteditor och de gratis kommandobaserade utecklingsverktygen tillhandahållna i .NET framework SDK. För att arbeta snabbare och mer effektivt är det dock en stor fördel att använda sig av någon av de IDE (Integrated Development Environment) som finns på marknaden. Microsofts kommersiella IDE som använts vid implementationerna i detta arbete heter Visual Studio .NET 2003. En annan kommersiell IDE på marknaden är ”Borland C# Builder for the Microsoft .NET framework”. WebMatrix heter en gratis IDE byggt helt i C# med .NET framework och Windows Forms för användargränssnittet. Ett annat intressant projekt under utveckling kallas Mono [20] och är en implementering av .NET med öppen källkod som ska kunna köras under Linux/Unix. Mono stöds bland annat av företagen Ximian och Novell.

4.1 Uppbyggnad av .NET framework

Programmeringsmodellen Microsoft .NET framework är i stort uppbyggt av följande delar

- Common Language Runtime (CLR)
- Just In Time Compiler (JIT)
- Microsoft Immediate Language (MSIL)
- Framework Class Library (FCL) (Gemensamt för VB, C#, J# och ”managed” C++)

[7] ”.NET” används ofta lite slarvigt för att ibland referera till .NET platform och ibland till programmeringsmodellen .NET framework (4.1)
[8] Runtime processen som körs på datorn är Aspnet_wp.exe

4.2 Common Language Runtime (CLR)

CLR:en är stommen i .NET och sköter minneshantering (bland annat ”garbage collector”), trådhantering, integration mellan olika språk, interoperabilitet med befintlig kod och system samt säkerhet. Den Exekverar MSIL med hjälp av JIT kompilering.

4.3 Just In Time Compiler (JIT)

Varje gång då CLR:en kör en .NET applikation så använder den sig av JIT för att kompilera MSIL koden till binärkod för den maskin som JIT kör på. JIT optimerar även koden. JIT kompilerar inte hela koden på en gång utan de delar, klasser och metoder, som anropas. Har den väl kompilerat ett stycke av programmet behöver denna ej kompileras om då stycket körs igen. JIT kommer ihåg vad den kompilerat och kompilerar aldrig samma kod två gånger så länge som programmet körs.

4.4 Microsoft Immediate Language (MSIL)

Samtliga programspråk i .NET framework kompileras till MSIL då kompilering till ”managed code” är valt. MSIL brukar även refereras som IL. IL är specificerat i ECMA CLI [2]. MSIL kan jämföras med bytekoden som Java Virtual Machine (JVM) läser. För att erbjuda interoperabilitet mellan olika programspråk är typerna i .NET framework CLS kompatibla och kan därmed användas av vilket språk som helst med en kompilerare som följer ”Common Language Specification”. Även CLS finns specificerat i ECMA CLI [2].

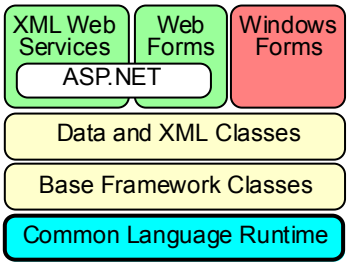
4.4.1 Verktyg för MSIL

Verktyget för att dissamblera ett .NET program är *ildasm.exe*. Ett dissamblerat program kan modifieras för att sedan assemblera det igen med verktyget *ilasm.exe*. Båda dessa verktyg ingår i .NET SDK. Möjligheten att dissamblera ett, till maskinkod kompilerat, program är normalt inget större problem då programfilen innehåller maskinkod. Vid dissamblering av MSIL fås dock en kod som är snarlik källkodsfilen, med klasser och metoder uppstrukturerade. Det finns även program som går ett steg längre och dissemblerar direkt till C# och då fås en nästan identiskt källkod till originalet. Detta kan vara ett stort problem för företag som måste hålla sin källkod hemlig. Det finns tredjepartsleverantörer som levererar program för att lösa detta problem. I Microsofts IDE Visual Studio .NET 2003 finns ett gratisprogram, Dotfuscator Community Edition, integrerat. Preemptive Solution heter företaget som utvecklat Dotfuscator och DashO som är motsvarigheten för Java program. Kommersiella versionen av Dotfuscator använder en rad olika tekniker för att göra den dekompilerade MSIL koden obegriplig. En av de enklaste teknikerna kallas ”renaming” och går ut på att den döper om alla klasser till bokstavskombinationer.

4.5 Framework Class Library (FCL)

Ett välutvecklat klassbibliotek är ett måste för att ett programmeringsspråk ska bli framgångsrikt idag (jämför med Java). Klassbiblioteket i .NET framework är för en javaanvändare ganska lätt att sätta sig in i. Klasserna har andra namn än i java men tankesättet är ungefär detsamma.
Figur 4 visar i stort uppdelningen av FCL. ”Base Framework Classes” innefattar grundläggande funktionalitet såsom input/output, trådning, nätverkskommunikation och stränghantering. Data-klasserna hanterar anslutning till SQL samt manipulering av data. Dessa klasser brukar refereras som ADO.NET (se 4.9). XML klasserna tillåter bland annat manipulation och sökning i XML dokument. ASP.NET som består av klasser för XML Web Services och Web Forms beskrivs närmare i 4.8. Windows forms tillhandahåller klasser för

utveckling av grafiskt användargränsnitt (GUI) med ”drag and drop”-teknik i bland annat VS.NET.



Figur 4. *Uppdelningen av FCL.*

En skillnad mot t.ex. Java som är värd att notera är att åtkomsten till variabler i klasser oftast sker med hjälp av get och set funktioner. I exemplet nedan visas hur åtkomst till den privata variabeln private_count sker via ”egenskapen” Count som definierar på vilket sätt som private_count ska sättas (set) respektive hämtas (get). I detta exempel görs det på enklaste möjliga sätt men i t.ex. TextBox klassen i FCL så sker åtkomst via ”egenskapen” Text. Get funktionen i Text läser data från ViewState medan Set funktionen skriver till ViewState.

Exempel (C#):

```
class IntHolderClass
{
   private int private_count;
   public IntHolderClass(int count)
   {
      private_count = count;
   }
   public int Count
   {
      get
      {
         return(count);
      }
      set
      {
         count = value;
      }
   }
   public override string ToString()
   {
      return(count.ToString());
   }
}

//Instantiering av klassen
IntHolderClass tal1 = new IntHolderClass(99);

//läsning och skrivning till den privata variabeln private_count kan nu ske
på
//följande sätt
tal1.Count = 0;
tal1.Count++;  //Denna rad både hämtar värdet på private_count, ökar
               //detta med 1 och skriver slutligen tillbaka värdet
```

4.6 Assemblies och Metadata

Assemblies och Metadata är grundläggande för hela .NET framework. En hel del (partition II) av Ecma CLI [2] standarden behandlar just metadata.

4.6.1 Assembly

En assembly består av ett antal filer som installeras som en enhet. I Implementation 2 (se 8) i denna rapport består assemblyn av en enda .dll fil. Figur 5 visar ett exempel på en assembly.

4.6.2 Metadata

Metadata beskriver typer och medlemmar i MSIL koden som ingår i assemblyn. Egenskaper som beskrivs är t.ex. namn, synlighet (t.ex. private och public), basklasser och gränssnitt.

Med hjälp av ildasm.exe (se 4.4.1) kan man titta på metadatat och manifestet för en assembly.

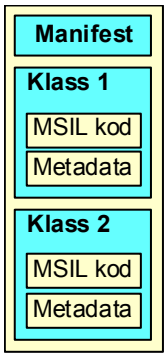
Fördelarna med metadata är många. En assembly blir självbeskrivande. Metadatat beskriver allt som behövs för att en modul[9] ska kunna kommunicera med en annan modul. Metadatat ersätter funktionaliteten som IDL (Interface Description Language) tillhandahåller för COM.

4.6.3 Manifest

Manifestet består av metadata som beskriver en assembly (Figur 5). Alla assemblys, oavsett om de innehåller en eller flera filer måste ha ett manifest.

Manifestet innehåller följande information om assemblyn:

- namn, version och säkerhetsbehov
- Vilka andra filer som tillhör assemblyn om denna består av fler än en fil
- Vilka typer som är definierade i andra filer än den som innehåller manifestet som skall exporteras
- Eventuellt en digital signatur samt en publik nyckel för denna.



Figur 5. *Exempel på en .NET assembly som innehåller två klasser.*

4.7 ”Managed code” och “Unmanaged code”

“Managed code” är programkod som kompilerats till MSIL. Det kallas för ”managed code” för att CLR:en ”hanterar” koden, d.v.s. exekverar den. ”Unmanaged code” är kod som

[9] En modul består av en ensam PE-fil (filformatet som används av exekverbara filer eller filer som länkas till exekverbara filer). Har den ett manifest så blir den en assembly. Har den ej ett manifest måste den tillhöra en assembly för att CLR:en ska kunna ladda den.

CLR:en ej hanterar, d.v.s. den är kompilerad direkt till maskinkod. Som exempel kan nämnas att program skrivna i C# och Visual Basic som standard kompileras till ”managed code” och program skrivna i C++ kompileras till ”unmanaged code”. Med inställningar till kompilatorn kan även C++ kompileras till ”managed code” då kallat ”managed C++”.

Fördelen med att använda ”managed code” och låta CLR:en exekvera programmet är bl.a: Hantering av säkerhet (överskridande av buffertar etc), enkel kommunikation till andra .NET assemblies, automatisk minneshantering med ”garbage collection” och typ-säkerhet[10] som innebär kontroll av att programmet endast får åtkomst till minnespositioner som det är auktoriserat att komma åt. En annan fördel med att använda ”managed code” är att .NET blir mer eller mindre programspråksoberoende med vilket menas att objekt kan skapas, klasser ärvas och metoder anropas mellan olika programmeringsspråk. T.ex. så kan en klass i C# ärva en klass från Visual Basic och vice versa. Detta är möjligt genom att alla program kompileras till det gemensamma språket MSIL.

4.8 ASP.NET

ASP.NET är en del av .NET som tillhandahåller klasser och funktionalitet för webbprogrammering. Man kan dock se ASP.NET snarare som en utökning av .NET eftersom alla andra klasser och tekniker fortfarande kan användas vid webbprogrammering. Detta gör ASP.NET mycket kraftfullt men ökar även inlärningströskeln avsevärt jämfört med traditionella ASP. Det som skiljer ASP.NET från andra .NET program är i huvudsak ”XML Web Services” och ”Web Forms”. ASP.NET är tänkt att ersätta ASP.

4.8.1 Web Forms

”Web Forms” är en del av ASP.NET. ”Web Forms” är en kombination av HTML och server komponenter även kallade ”Web Controls” som exekverar på servern och skapar vanlig html kod som skickas till klienten. I VS.NET kan ”Web Forms” användas på liknande sätt som Visual Basic används idag. D.v.s. man drar komponenter (knappar, textfält, rull-listor etc.) (se ”Web Controls” nedan) till sidan och sedan dubbelklickar man på komponenten för att skriva dess kod. Med ”Web Forms” blir skillnaden mot att programmera vanliga Windows program med ”Windows Forms” (liknar ”Web Forms” med ”drag and drop” av grafiska komponenter och händelsebaserad programmering) inte så stor.

4.8.2 Web Controls

Web Controls är grafiska komponenter på ASP.NET sidor (.aspx filer). Istället för att lagra html kod direkt i .aspx filen såsom det görs i gamla ASP lagras en beskrivning av knappen som sedan rendreras till HTML. Detta medför flera fördelar. Bland annat kan rendreringsmotorn anpassa sig för olika typer av webbläsare och nya HTML-standarder utan att ändringar behöver göras i .aspx sidan. Renderingsmotorn kan dessutom skapa javascript för att utföra kontroll av t.ex. inmatad data. Dessutom används namnen för Web Controls för att referera till deklaration i codebehind-filen (se 4.8.4 nedan). Nackdelen är förstås att prestanda blir lidande då rendrering förbrukar en del CPU-tid. Dock så minimeras tiden genom att följande förfrågningar på samma sida inte behöver rendreras om varje gång utan lagras i cachen.

[10] Typ-säkerhet (type safety) är ej att förväxla med datatyp-verifikation som används i många programmeringsspråk [17]

Exempel:
Beskrivningen av en OK-knapp i .aspx filen.

```
<asp:Button
   id="Button1"
   style="Z-INDEX: 105; LEFT: 72px; POSITION: absolute; TOP: 280px"
   runat="server"
   Width="120px"
   Height="40px"
   Text="OK">
</asp:Button>
```

OK-knappen renderad till HTML

```
<input type="submit" name="Button1" value="OK id="Button1"
style="height:40px; width:120px; Z-INDEX:105; LEFT:72px;
POSITION: absolute; TOP: 280px" />
```

OK-knappen renderad till HTML då RegularValidatorExpression[11] används på sidan

```
<input type="submit" name="Button1" value="OK" onclick="if
(typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); "
language="javascript" id="Button1" style="height:40px;width:120px;Z-INDEX:
105; LEFT: 72px; POSITION: absolute; TOP: 280px" />
```

4.8.3 XML Web Services

I .NET så är XML Web Services en grupp klasser och tekniker för att skapa och använda Web Services.

XML och Web Services är något som marknadsförarna av .NET ofta beskriver som lösningen på alla problem och den nya internetrevolutionen. Om de har rätt i detta återstår att se men helt klart är att stora delar av .NET är präglat av just XML. T.ex. så är objektet DataSet (se 4.9.1) helt serialiserbart i XML just för att det ska vara enkelt att utbyta information med hjälp av Web Services. XML och Web Services är tänkt att fungera som en bas för att utbyta information inom och mellan företag samt mellan olika system och plattformar.

W3C's definition av en Web Service (fritt översatt) [15]: "En Web Service är ett mjukvarusystem designat för interoperabilitet maskin-till-maskin över ett nätverk. Dess gränssnitt är beskrivet i ett format möjligt för en maskin att bearbeta (specifikt WSDL). Andra system interagerar med Web Servicen, på ett sätt som är föreskrivet av dess beskrivning, med SOAP-meddelanden som typiskt är överförda med HTTP, serialiserade med XML och i överensstämmelse med andra relaterade Internet-standarder."

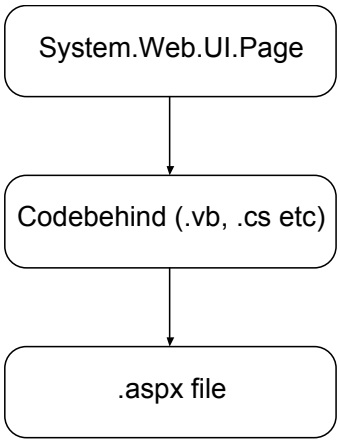
Web Services har tagits fram och utvecklats av flera olika företag och organisationer. Microsoft har varit en stor pådrivare av tekniken liksom bland annat SUN Microsystems, IBM och BEA Systems, vilka är några få av de ca 60 företag som ingår i Web Services Architecture Working Group tillhörande W3C.

Web Services Interoperability Organisation (WS-I) är en industrisatsning för att stödja Web Services interoperabilitet över olika platformar, applikationer och programspråk. WS-I tillhandahåller inga specifikationer utan kompletterar framställare av dessa såsom W3C genom att stödja och ge vägledning för implementering och testning av Web Services. Över 160 olika företag är anslutna till WS-I.

[11] En "Web Control" som kan kontrollera att inmatad text har vissa egenskaper. T.ex. att en e-mailadress är en giltig sådan. Med RegularValidatorExpression kan man själv skriva valideringsuttryck för att validera valfria textformat. Som synes så genererar renderingsmotorn förutom HTML-kod även javascript-kod.

4.8.4 Codebehind

Codebehind är en metod som används för att separera källkoden från html-koden. Codebehind medger ett mer strukturellt sätt att programmera som mer liknar traditionell programvaru-utveckling än ASP. Detta fungerar på så sätt att man låter .aspx-filen som innehåller html-koden (i orendrerad form) ärva av källkodsfilen som i sin tur ärver av Page-klassen, se Figur 6. Koden i ”codebehind-filen” måste alltid kompileras till ”managed code”. Dock kan anrop till ”unmanaged code” göras via så kallade wrappers (se 5.1.1) vilket är vanligt vid migrering till ASP.NET.



Figur 6. *Arvförhållande*

4.9 ADO.NET

ADO.NET är det dataaccess-API som används i .NET applikationer. I ADO.NET ingår ett antal objekt för att hämta och manipulera data. Det mest omskrivna och viktigaste objektet heter DataSet och beskrivs nedan. DataSet-objektet är motsvarigheten till RecordSet-objektet i gamla ADO. Andra objekt i ADO.NET är DataTable, DataColumn och DataRow. I Implementation 2 (kap. 8) så visas exempel på hur läsning till RecordSet kan gå till via en wrapper.

4.9.1 DataSet

Kort beskrivet innehåller DataSet-objektet ett antal tabeller, vilka består av ADO.NET-objektet DataTable som i sin tur består av ett antal rader, vilka består av ADO.NET-objektet DataRow.

En nyhet med DataSet-objektet jämfört med det gamla RecordSet-objektet, som används i ASP, är att DataSet-objektet kan serialiseras till XML och skickas med hjälp av Web Services (se 4.8.3 om XML Web Services).

Exempel på användning av DataSet (C#):

```
SQLConn.Open();
string QueryString = "SELECT DISTINCT CustomerName FROM PmcCustomer ORDER
                     BY CustomerName";
SqlDataAdapter myCommand = new SqlDataAdapter(QueryString, SQLConn);
DataSet ds = new DataSet();
myCommand.Fill(ds, "CustomerName");
CompanyDropDownList.DataSource = ds;
CompanyDropDownList.DataTextField = "CustomerName";
```

```
CompanyDropDownList.DataValueField = "CustomerName";
CompanyDropDownList.DataBind();
CompanyDropDownList.SelectedIndex = 0;
SQLConn.Close();
```

Första raden öppnar en anslutning till SQL-databasen. De tre efterföljande satserna läser ett antal poster från SQL-databasen. MyCommand.Fill fyller ds, som är ett DataSet, med posterna från databasen. DataSource i CompanyDropDownList, som är en webbkontroll (se 4.8.2 Web Controls) av typen DropDownList (rull-lista), sätts därefter till ds som är ett DataSet-objekt. När kodsnutten körts klart uppdateras webbkontrollen (CompanyDropDownList) på webbsidan (se Figur 11, sid 37) med värdena som lästs från databasen.

5 Jämförelse mellan ASP och ASP.NET

Detta kapitel kommer att belysa skillnaderna, som är nödvändiga att känna till vid migrering, mellan ASP och ASP.NET. Från början hade Microsoft tänkt att ASP.NET skulle vara 100% bakåtkompatibelt med ASP men till slut så valdes att acceptera vissa skillnader för att istället kunna satsa på nya bättre lösningar i ASP.NET. Detta skapar givetvis en hel del problem för gamla ASP system som ska migreras till ASP.NET och det är hur man på bästa sätt kan lösa dessa som Implementation 1 i kapitel 7 handlar om. Även i detta kapitel kommer vissa lösningar/metoder för migrering samt interoperabilitet att presenteras.

Huvudskillnaderna mellan ASP och ASP.NET är att programmeringsspråket för ASP är ett icke objektorienterat scriptspråk, som dock kan använda sig av så kallade COM-objekt, medan ASP.NET är helt objektorienterat oavsett vilket programmeringsspråk som används. I det vanliga fallet då VBScript används för ASP och Visual Basic för ASP.NET kan VBScript i många fall översättas relativt lätt till Visual Basic vilket visas i 5.2 nedan.

5.1 Komponent-typer COM / .NET Assemblys

Huvudtanken med komponenter är att man ska kunna återanvända kod. Komponenterna tillhandahåller ett gränssnitt som definierar klasser och metoder som andra program kan anropa. COM (Component Object Model) är komponentmodellen som används i ASP. Motsvarigheten i .NET heter .NET assembly.

5.1.1 COM Wrappers / Interop assemblies

För att kommunicera mellan de olika komponenttyperna så måste så kallade wrappers skapas som översätter anropen mellan de olika programmeringsmodellerna.
CLR:en i .NET kan anropa ett COM objekt genom att skapa en så kallad Runtime Callable Wrapper (RCW). Denna fungerar som en brygga mellan ”managed” .NET kod och ”unmanaged” COM kod. En så kallad ”interop assembly” fungerar som en mall som CLR:en använder för att skapa en RCW. En RCW skapas för varje COM objekt som man vill kommunicera med i .NET. Type Library Importer (tlbimp.exe) heter verktyget i .NET framework för att skapa en ”interop assembly”.

Det går även att omvänt anropa en .NET komponent (assembly) från en COM-klient. När detta sker så skapas en så kallad COM Callable Wrapper (CCW) som är uppbyggd av ”unmanaged” kod och refereras på liknande sätt som ett COM objekt.

5.2 Skillnader i VB.NET jämfört med VBScript

Som beskrivs i början av kapitlet så är ASP-sidor ofta skrivna i programmeringsspråket VBScript. VB.NET (Visual Basic .NET) är det språk i ASP.NET som till stor del liknar VBScript. Det är därför naturligt att använda VB.NET vid migration av ASP sidor skrivna i VBScript till ASP.NET.

Nedan följer några av de vanligast använda ändringarna för att VBScript ska kunna kompileras som VB.NET kod:

- Datatypen *Variant* finns ej längre. Den har ersatts med typen *Object*. *Object* måste ”castas” till den typ man vill använda.

- Ändring av betydelsen vid deklaration med *Dim*.
  Ex: `Dim a,b,c As Integer`
  I VB.NET så kommer samtliga variabler att deklareras som Integer medan endast ”c” kommer att deklareras som *Integer* i VBScript. a och b kommer att vara av typen *Variant*.

- Parenteser måste alltid användas vid funktionsanrop. I VBScript är det valbart.
  ```
  ADOconn.Open "DSN=TEST"
  ```
  Ändras till:
  ```
  objConn.Open("DSN=TEST")
  ```

- Så kallade ”default properties” är borttagna i VB.NET. Vill man t.ex. komma åt textinnehållet från en textruta måste detta specificeras explicit.

  Ex 1:
  ```
  Dim str As String = TextBox1
  ```
  Fungerar ej eftersom TextBox1 refererar till objektet TextBox1 och inte till standard-egenskapen TextBox.Text. Ändra till:
  ```
  Dim str As String = TextBox1.Text
  ```

  Ex 2:
  ```
  SortBy(“SortActiveOrders”)
  Ändras till:
  SortBy.Fields("SortActiveOrders").Value
  ```

- Set behövs ej längre eftersom namnet på ett objekt refererar till själva objektet och inte till “default properties”
  Ex:
  ```
  Set ADOconn = Server.CreateObject("ADODB.Connection")
  ```
  Ändras till:
  ```
  ADOconn = Server.CreateObject("ADODB.Connection")
  ```

- “Option Explicit” är förvalt vilket innebär att samtliga variabler måste deklareras innan de används.

- “Wend” har ersatts med ”End While”.

- ”IsNull” har ersatts med ”IsDBNull”.

5.3 Dataåtkomst ADO/ADO.NET

ASP använder sig av ADO (ActiveX Data Objects) för dataåtkomst (data accesss) medan ASP.NET använder sig av en nyare variant kallad ADO.NET. För bakåtkompatibilitet med ASP finns stöd i .NET för att läsa ADO objekt som använder sig av COM (Component Object Model).

ADO kräver att komponenterna som sänds och mottas är COM-komponenter. ADO.NET överför data i standard xml-format och då behövs t.ex. inte typ-konvertering utan typer definierade i .NET framework används.

I ADO läser man svaret från SQL-servern till ADO objektet RecordSet. I ADO.NET läser man till objektet DataSet. DataSet-objektet kan innehålla flera tabeller och relationer mellan dessa. Man kan se DataSet-objektet som en kopia av en minidatabas och komplexa förfrågningar kan göras i DataSet-objektet utan att information begärs från SQL-servern, vilket medför en prestandavinst jämfört med ADO's RecordSet.

5.4 Trådning

ASP.NET använder sig av MTA-modellen (Multiple Threading Apartment) i motsats till ASP (och tidigare VB applikationer) som använder sig av STA-modellen (Single Threading Apartment). Om ADO.NET komponenter (t.ex. DataSet) används i ASP.NET så kommer dessa som standard att följa båda modellerna. Om man använder sig av STA-komponenter i ASP.NET så måste attributet *aspcompat* sättas till *true* i en <%@ Page > tag. Ex: <%@Page aspcompat=true Language = VB%>

5.5 Autentisering

Autentisering kallas metoden för att kontrollera att en användare är den som han/hon utser sig vara. Det sker vanligen genom att användaren får logga in med användarnamn och lösenord. I ASP används ofta en inkluderingsfil (engelska: include file) som läggs in i början på varje ASP-sida. I .NET är autentiseringen enklare. .NET använder sig av så kallad cookie-baserad autentisering. Då räcker det med att lägga de filer som ska vara säkra i en speciell underkatalog. Skulle användaren försöka komma åt en fil som denne inte är autentiserad för så visas istället en förinställd sida där användarnamn och lösenord kan matas in på nytt. I .NET är det inte längre tillåtet med inkluderingsfiler vilket kan kräva en del omskrivning vid en migrering.

5.6 Utbyte av variabler

Det finns tre olika tillstånd (engelska: states) som kan användas för att utbyta variabler i ASP och ASP.NET. Tyvärr så är de tre metoderna ej kompatibla mellan ASP och ASP.NET. D.v.s. det går ej att läsa ett tillstånd från en ASP sida i ASP.NET och vice versa. Anledningen till detta är flera och några beskrivs nedan. Det finns dock tredjepartstillverkare [5, 6] av mjukvara som gör det möjligt att dela vissa tillstånd mellan ASP och ASP.NET. Använder man sig endast av textvariabler är det dock mycket lätt att skicka tillståndsvariablerna, med HTTP-metoden POST, från en .aspx sida till en .asp sida och vice versa. Fast då måste man själv se till att tillståndsvariablerna är synkroniserade i ASP respektive ASP.NET.

Svårigheten att utbyta variabler mellan ASP och ASP.NET kan vara ett av de största problemen vid migrering till ASP.NET. I alla fall så är användandet av t.ex. "Session State", som beskrivs nedan, viktigt att utreda innan man börjar migrera.

5.6.1 View State

View State finns endast i ASP.NET och innehåller information om en .aspx sida när den senast var processad av servern. Den innehåller t.ex. information om "Web Controls" (se 4.8.2), vilket innefattar dess olika attribut såsom enabled/disabled, position och innehåll. View State bidrar till en del av enkelheten för ASP.NET i och med att programmeraren slipper tänka på att lagra komponenternas egenskaper manuellt vilket medför att programmeringen mer liknar traditionell programmering.

View State lagras på varje sida i ett gömt textkodat fält, med HTML attributet ”type=hidden”, på webbsidan. Vill man själv spara variabler som är unika för just den aktuella sidan så kan även dessa läggas till i View State.

Med denna enkelhet medföljer oundvikligt ett antal nackdelar, i huvudsak prestanda och säkerhet. Eftersom View State lagras på webbsidan medför det att information skickas fram och tillbaka mellan servern och klienten. Har man en avancerad sida med ett flertal ”Web Controls” så upptar View State ofta över 40 kB, vilket kan medföra en lång hämtningstid för sidan, särskilt vid en långsam uppkoppling. Servern slipper dock att använda resurser för att spara View State i dess eget minne. Säkerhetsrisken uppstår genom att informationen lagras i klientens .aspx-sida, som med lite jobb skulle kunna läsas av en illvillig person. Vidare så kan informationen ändras och postas tillbaka till servern som processar denna. Det är därför viktigt att inte lagra några hemliga lösenord eller systemkommandon, som kan exekveras på servern, i View State.

För bästa prestanda bör man stänga av View State då denna ej är nödvändig. Detta kan antingen göras för hela sidan eller för en komponent i taget genom att sätta EnableViewState = ”false” i <%@Page %> direktivet respektive som attribut till den ”Web Control” (grafisk komponent) som avses.

Ex. hur det gömda ViewState-värdet lagras i den processade html sidan:

```
<input type="hidden" name="__VIEWSTATE"
value="dDw5NjE3MjI3MjI7dDw7bDxpPDE+Oz47bDx0PDtsPGk8MT47PjtsPHQ8QDA8Ozs7Ozs7
Ozs7Oz47Oz47Pj47Pj47Pp1V9ePQAxy68vQQF58bEESHUItl" />
```

Ex. på att lägga till och läsa ett eget string-objekt till View State i ASP.NET (C#)

```
ViewState[”UserName”] = ”Pelle”; // ”UserName” är nyckeln, ”Pelle” är värdet.
String username = ViewState[”UserName”]; // username == ”Pelle”
```

5.6.2 Session State

Objekt som sparas i Session State kan utbytas mellan olika sidor som nyttjas av samma användare under samma session vilken vanligtvis varar så länge användaren ej varit passiv (inga aktiviteter har förekommit under exempelvis 20 minuter). I ASP används ofta Session State för autentisering, vilket är fallet för PreCom Bud som migreras i Implementation 1 (kap. 7). När man loggar in på en webbplats så sätts en variabel i Session State till ett visst värde. T.ex. Session(”Auth”) sätts till ”Y”. Överst på alla andra sidor har man en rutin som kontrollerar om Session(”Auth”) == ”Y”. Om så ej är fallet så dirigeras man om till inloggningssidan. I annat fall visas den skyddade sidan. I ASP lagras alltid en cookie på klienten som innehåller en 120-bitars sessions ID. I ASP.NET är detta valbart och det finns lösningar som fungerar även om funktionen för att ta emot ”cookies” är avstängt på klienten.

Exempel på hur URL-strängen kan se ut då sessions-ID lagras i denna:
http://www.pocketmobile.se/(12mfju55vgblubjlwsi4dgjq)/NewOrder.aspx.

En annan svaghet i ASP är att variablerna i Session State måste ligga på samma server som sköter exekveringen av initieringssidan. I ASP.NET behöver inte detta vara fallet då Session State informationen kan skrivas direkt till en sql-databas eller lagras på en separat IIS-server. Detta innebär en stor fördel vid användning av så kallade ”Web Farms” där flera servrar sköter exekvering och minneshantering och resurserna kan fördelas jämnt över dessa (load balancing). Alla objekt i .NET är dessutom organiserade som XML datastrukturer och Session

State kan därmed lagra hela objekt som kan kommas åt av vilken sida som helst som öppnas i samma session.

5.6.3 Application State

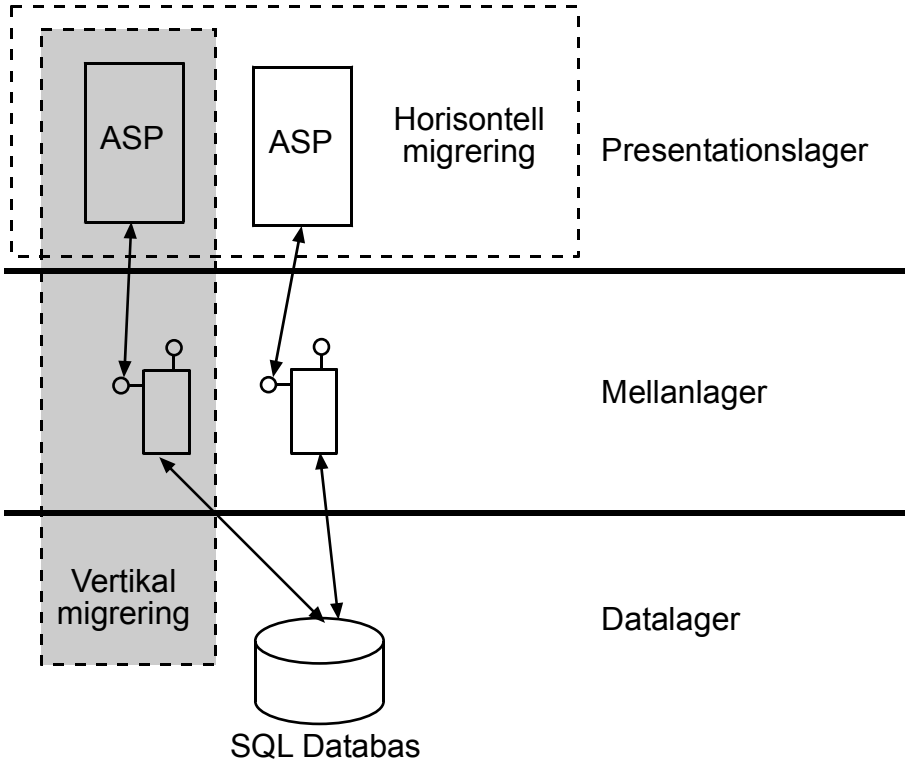
I ”Application State” lagras information som behöver vara tillgängligt för applikationen. Ofta kan en databas med fördel användas istället för ”Application State”, vilket medför en mer beständig lagring då ”Application State” försvinner bland annat vid omstart av webbservern. Då Application State ej förekommer i någon av implementationerna så kommer ingen vidare behandling av detta ”tillstånd” ske i rapporten.

6 Migreringsstrategi

Detta kapitel fungerar som en introduktion till nästföljande kapitel där migreringsstrategierna i detta kapitel testas.

För att inte behöva migrera hela programmet på en gång, undvika onödigt arbete och minimera risken att migreringsprojektet fallerar gör man bäst i att använda sig av någon strategi för att migrera mindre delar i taget. Olika migreringsstrategier passar olika bra beroende på hur applikationen är uppbyggd.

En applikation kan t.ex. ha väldigt mycket kod i mellanlagret medan den har mindre mängder kod i presentationslagret. Ibland kan det, på grund av applikationens konstruktion, vara svårt eller mer eller mindre omöjligt att endast migrera mindre delar av systemet.



Figur 7. *Åskådliggörande av vertikal respektive horisontell migrering.*

6.1 Horisontell migrering

Vid en horisontell migrering migreras ett helt lager i taget till .NET (se Figur 7). T.ex. så kan migrering av presentationslagret ske genom att migrera samtliga .asp filer (VBScript) till .aspx filer (VB.NET). Ett annat alternativ är att migrera mellanlagret. Fördelen med horisontell migrering är att man migrerar i steg istället för att göra allt med en gång. Trots detta kan stegen bli väldigt stora eftersom det som regel inte finns så många lager samtidigt som presentationslagret i många fall dominerar, vilket medför att skillnaden mot att migrera allt inte behöver vara så stor. En annan stor fördel är att hela systemet kommer att använda sig

av ASP.NET sessionsvariabler så att man inte behöver hantera utbyte av dessa mellan ASP och ASP.NET. Delad kod i inkluderingsfiler migreras som en enhet och dubbla kopior av denna behöver inte finnas.

6.2 Vertikal migrering

Vid en vertikal migrering gäller det att hitta en så isolerad bit av programmet som möjligt för att minimera behovet av interoperabilitet. För att göra detta används en metod som i denna rapport kallas kodvägsanalys och beskrivs nedan. Sedan migreras all kod som har anslutning till kodvägen (se Figur 8) till ASP.NET. Det kan även vara bra att göra en skiss på hur sidorna förhåller sig till varandra för att man lättare ska kunna se vilka sidor som kan vinna på en vertikal migrering (se Figur 9). Eftersom även mellanlagret migreras vid en vertikal migrering så innebär det i praktiken en omskrivning av koden genom alla delar i kodvägen. En vertikal migrering kan därför vara lämpligt för att göra ett småskaligt test av en ny programarkitektur.

6.2.1 Kodvägsanalys

Kodvägsanalys medför att man undersöker vilka komponenter som anropas vid körning av programmet (i detta fall laddning av en ASP-sida). Vanligtvis anropar programmet ett mellanlager och mellanlagret anropar i sin tur en databas. Mellanlagret kan även det bestå av olika komponenter som anropar varandra innan databasen nås. Kodvägen är den väg som löper från presentationslagret (ASP-sidans grafiska gränssnitt) genom en eller flera komponenter i mellanlagret och slutligen till datalagret.

6.3 Migrering sida för sida

Denna teknik går ut på att man migrerar befintliga sidor en i taget och bygger oftast på utbyte av sessionsvariabler mellan sidor i ASP och ASP.NET. Viktigt att tänka på vid användning av denna strategi är att välja metod för utbyte av sessionsvariabler (se 7.3), hantering av delad kod i ASP-sidor och anrop till mellanlager. Ett alternativ för hantering av delad kod är att denna migreras till ASP.NET och samtidigt bibehålls i ASP. Nackdelen med detta alternativ är att två kopior av liknande kod kommer att finnas både i ASP och ASP.NET. För att undvika flera kopior av samma kod, kan all delad kod lagras i ett .NET bibliotek som även anropas av ASP via en så kallad ”Com Callable Wrapper” (se 5.1.1).

Fördelen med denna strategi är att migrering kan ske i små steg i taget och att man kan rikta in sig på att migrera de sidor som medför en förbättring för användaren genom t.ex. bättre prestanda med .NET’s cachningsmöjligheter eller mer användarvänligt GUI. Andra sidor som inte vinner något på en migrering kan lämnas kvar som de är och man undviker på så sätt att introducera nya buggar i dessa genom onödig migrering.

6.4 Migrering allteftersom

Nya sidor som läggs till skrivs i ASP.NET medan de gamla lämnas kvar som de är. Förhållandena att ta hänsyn till, såsom sessionsvariabler, är desamma som i föregående strategi (”sida för sida”).

6.5 Kombinering av olika strategier

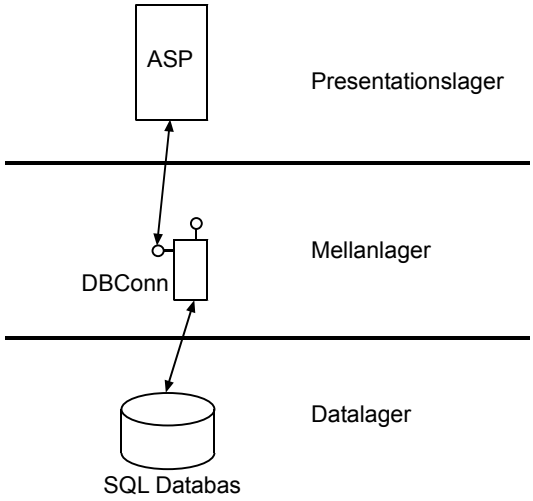
Olika migreringsstrategier kan ibland med fördel kombineras. De två senaste ”sida för sida” samt ”migrering allteftersom” kan som regel alltid kombineras med gott resultat.

7 Implementation 1 – Migrering av orderhantering till ASP.NET

För förståelse av denna implementation krävs att man läst igenom kapitel 6 om migreringsstrategier. Syftet med denna implementation är att testa olika migreringsstrategier på PocketMobiles befintliga ASP-program PreCom Bud, men innan detta kan genomföras behövs programstrukturen analyseras.

7.1 Kodvägsanalys

För att få en överblick över systemet samt senare kunna testa en vertikal migrering så inleddes arbetet med att göra en kodvägsanalys.
DBConn är den enda förekommande komponenten (COM-objektet) i mellanlagret vilket medför att kodvägsanalysen blir mycket enkel. Kodvägsanalysen för de ASP-sidor som har databasåtkomst via mellanlagret åskådliggörs i Figur 8.



Figur 8. *Kodvägsanalys.*

7.2 Sessionsvariabler

Ett problem vid migrering då inte all kod migreras till ASP.NET, t.ex. vertikal migrering, är sessionsvariabler som beskrivs i kapitel 5.6.2 Session State. Dessa kan nämligen inte på ett enkelt sätt delas mellan ASP och ASP.NET.

7.2.1 Identifiering av sessionsvariabler[12]

Nedan beskrivs de fyra sessionsvariabler som ingår i programmet. Samtliga sätts i samband med inloggningen och behåller därefter sitt värde tills användaren loggar ut. Man kan därmed konstatera att sessionsvariblerna är statiska efter det att användaren har loggat in. Detta underlättar avsevärt migrering då man väljer att migrera efter strategin ”sida för sida” eftersom det då räcker med att en gång per inloggning skicka över sessionsvariablerna till ASP.NET.

[12] Vid identifieringen användes programmet grep (samma som för Unix) för att skapa en fil som innehöll samtliga referenser till sessionsvariabler. Utifrån den filen var det lätt att undersöka vilka sidor som skrev respektive läste till sessionsvariablerna.

Session("Auth") anger om en användare är autentiserad eller ej. Sätts till *”Y”* om användaren är autentiserad. I annat fall sätts den till *”N”*.

Session("PmcCustomerID") anger kund-id.

Session("CompanyUserName") anger företags användarnamn.

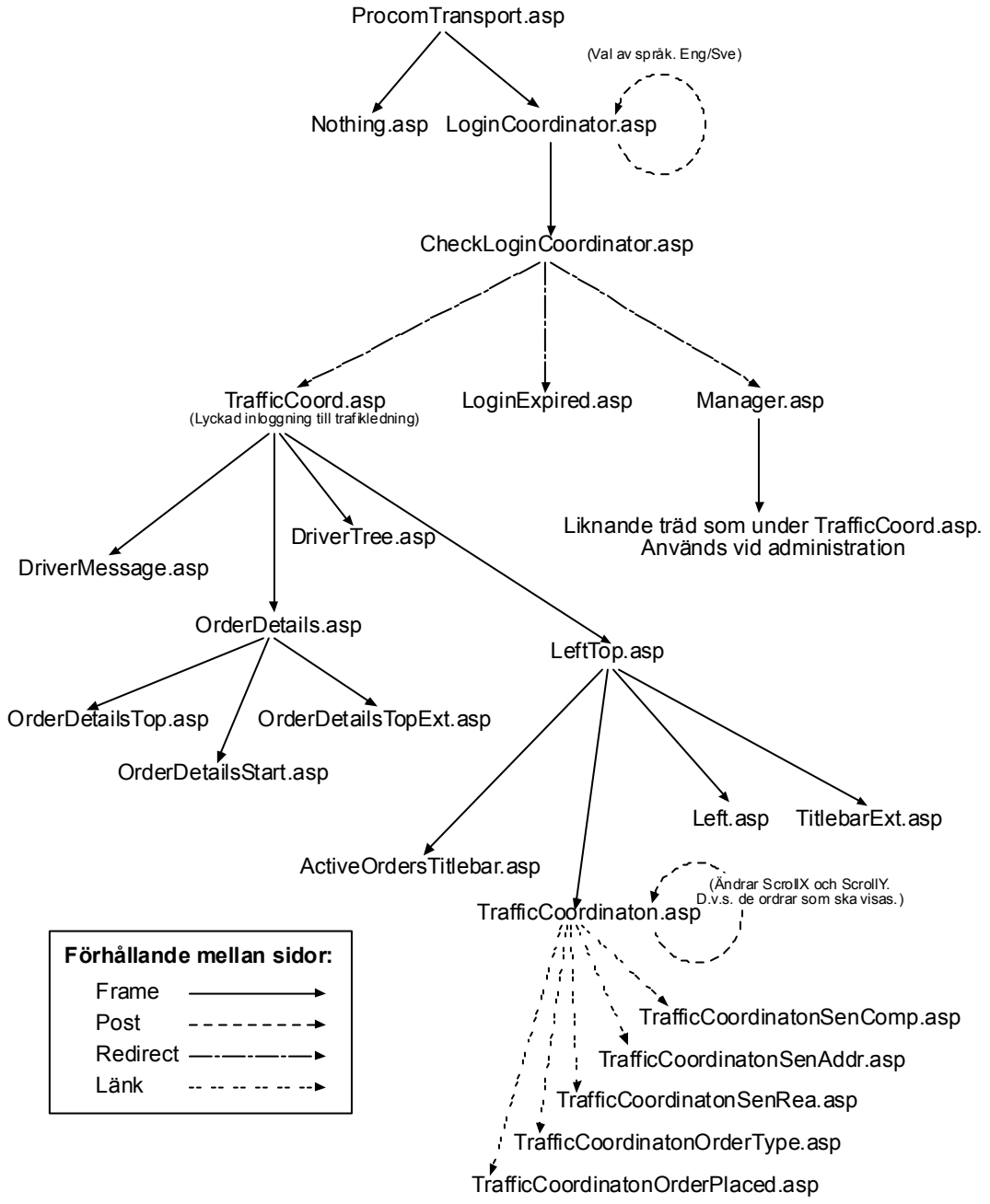
Session("Language") anger språk *(“Swedish”* eller *“English”*)

7.3 Möjliga migreringsstrategier

Utnyttjande av sessionsvariabler är inget hinder för horisontell migrering, vilket det däremot är för strategierna vertikal, ”migrering allteftersom” och ”sida för sida”. På grund av att sessionsvariablerna i PreCom Bud är statiska så var samtliga strategierna möjliga utan särskilt stora komplikationer.

Har man däremot dynamiska variabler kan man t.ex. ta hjälp av programmet StateStich från Cansonica [5] eller ASPBridge IncludeControl från IPrisma Software [6]. Ett annat alternativ som är gratis är att skapa ett eget sessionsobjekt både i ASP och ASP.NET som skriver till samma tabell i en SQL-databas och på så sätt kan delas, vilket Billy Yhen på Microsoft beskriver i en artikel [7]. Vill man ha ett annat enklare alternativ, även detta gratis, som endast kan överföra text, så kan man använda en metod för att posta sessionsinformationen mellan ASP och ASP.NET beskrivet av Peter A. Bromberg [12].

För att få en överblick av förhållandet mellan de olika ASP-sidorna, underlätta valet av migreringsstrategi, samt få en förståelse för funktionen hos de viktigaste ASP-sidorna gjordes en skiss av ASP-sidornas förhållande till varandra (se Figur 9). Genom att studera denna så kan man se vilka sidor som lätt kan migreras eller skrivas om i ASP.NET och dra stor nytta av den nya funktionaliteten. Man ser också vilka sidor som behöver ha kommunikation mellan varandra.



Figur 9. *Förhållandet mellan de viktigaste ASP-sidorna i PreCom Bud (tidigare Procom Bud). Sessionsvariablerna sätts vid inloggning av LoginCoordinator.asp och CheckLoginCoordinator.asp. PreComTransport.asp som synes högst upp i figuren är startsidan för applikationen.*

7.4 Test med horisontell migrering

För att se hur pass svårt det var att få presentationslagret att köra under .NET påbörjades ett småskaligt test av horisontell migrering av detta lager. Denna migreringsstrategi har fördelen att all hantering av sessionsvariabler sköts av ASP.NET och man behöver inte bestämma metod för att lösa detta problem.

7.4.1 Tillvägagångssätt

Ändelsen för .asp-filerna ändrades till .aspx och adderades till projektet i Visual Studio .NET. Därefter ändrades sådant i koden som var uppenbart att det inte skulle fungera. T.ex. ändrades alla referenser till andra .asp filer till .aspx filer istället. Vidare genomfördes ändringar som beskrivs i sektion 5.2 Skillnader i VB såsom att lägga till parenteser till funktionsanrop, ändra en variant-array till en objekt-array, ta bort set framför objekt-typer samt deklarera alla variabler. Även om det inte är nödvändigt att deklarera variablerna till en bestämd typ så rekommenderas att man gör så för att underlätta felsökning. Kodmigreringen i detta test utfördes helt för hand.

Även referenser till COM objekten CDBConn och ADODB lades till, vilket är Visual Studios sätt att skapa så kallade interop assemblies. Sedan kompilerades projektet (endast syntaxen kontrollerades på kod i .aspx filen) och sidorna testkördes i Internet Explorer 6. Det är först då som man upptäcker eventuella syntaxfel och andra fel som kan uppkomma vid migreringen.

Bland annat upptäcktes att Primary Interop Assembly (PIA) för namespace ADODB som innehåller objektet RecordSet inte fungerade direkt i skriptet på .aspx sidan. Däremot fungerade det i codebehind-koden. Det verkar som om Visual Studio inte inkluderar ADODB assemblyn vid kompilering av skriptkoden på .aspx sidan. Dessutom försöktes med att skapa referenser till olika versioner av COM-objektet ADODB. Inte heller detta fungerade vid kompilering av skriptkoden på .aspx sidan.

Slutligen så skapades en egen interop assembly med hjälp av tlbimp.exe (se 5.1.1) som sedan lades till som referens i Visual Studio, vilket fungerade utan problem. Helst vill man dock använda en PIA eftersom denna är signerad av utgivaren och CLR:en skiljer på typer i en PIA och den egentillverkade interop-assemblyn.

7.4.2 Slutsatser av horisontell migrering

Kodmigrering för hand är tidsödande. Även om det bara är enklare ändringar som behöver göras kan det bli ganska många ändringar. Detta tar tid och man kan behöva kompilera en sida, vilket innebär att öppna den i Internet Explorer eftersom det först är då som skriptkoden kompileras, många gånger för att hitta alla fel, vilket inte ens är säkert att man hittar. Hjälp finns dock att få från olika migreringsverktyg. De två som testades i detta arbete var gratisprogrammet ”Migration assistant tool” vars alpha version släpptes knappt två månader efter arbetets start samt ASP2ASPX från Netcoole. Migration assistant tool fungerade av okänd anledning inte. Förmodligen var det någon bugg i den tidiga versionen som förhoppningsvis kommer att ordnas. Test av ASP2ASPX visade att denna fungerar ganska bra och lyckades att översätta de flesta .asp sidor. Även vid användning av ett migreringsverktyg finns risk att nya buggar introduceras (se 9.2.2) och det finns vissa sidor där man behöver göra vissa manuella ändringar för att verktyget ska klara av att migrera dessa.

7.5 Vertikal migrering

Istället för att utföra ett fullskaligt test av denna migreringsstrategi kommer här istället ett exempel på hur denna skulle kunna utföras. Detta eftersom ett fullskaligt test skulle innebära en hel del implementeringsarbete som inte skulle vara av så stor nytta då detta alternativ valdes bort.

Precis efter man loggat in i PreCom bud visas en sida med aktiva order (order som ännu inte utförts. T.ex. att hämta ett paket från en viss adress och lämna det på en annan). Menyn för denna funktion kallas ”Aktiva order”. Andra funktioner i menyn är bland annat ”Lägg nya

order", "Inställningar" och "Historik". Var och en av dessa sidor är uppbyggd av ett antal andra sidor i ett antal ramar. Trädstrukturen under LeftTop.asp i Figur 9 ser olika ut beroende på vilken funktion som är vald. Väljs t.ex. historik-funktionen så kommer t.ex. LeftTop.asp att bytas ut mot HistoryLeftTop.asp och ActiveOrdersTitleBar.asp mot HistoryTitleBar.asp etc.

Vid vertikal migrering gäller det att hitta delar som är välisolerade från andra delar vilket dessa olika meny-funktioner är. I den följande texten så har historik-funktionen valts som exempel.

I kapitel 7.2.1 konstaterades att sessionsvariablerna i Precom Bud är statiska, vilket innebär att den enda kommunikationen som krävs mellan ASP och ASP.NET är att överföra sessionsvariablerna till den nya ASP.NET-sidan som ersätter den gamla historik-funktionen. Den stora mängden ramar som komplicerar utvecklingen av ASP-programmet PreCom Bud kan ersättas med olika så kallade "Web Controllers" i ASP.NET. De få funktioner i mellanlagret som anropas skrivs om till .NET.

Vertikal migrering innebär i praktiken vanligtvis en småskalig omskrivning av en isolerad del i programmet. Den delen kan då dra nytta av alla fördelar i .NET.

7.6 Tillvägagångssätt vid migreringsstrategierna "sida för sida" och "migrering allteftersom"

Migrering med strategin "sida för sida" kan ske på samma sätt som beskrivs i "Test med horisontell migrering" (7.4) ovan med skillnaden att tillvägagångssättet endast appliceras på en enda sida. Därefter tillkommer problemet att lösa delning av sessionsvariabler, vilket ej uppstod vid horisontell migrering då samtliga sidor i t.ex. presentationslagret (där sessionsvariblerna verkar) migrerades samtidigt. I kapitel 7.2 och 7.3 konstaterades att sessionsvariablerna för PreCom bud är statiska, vilket innebär att dessa kan skickas över med exempelvis POST till ASP.NET. Viktigt att tänka på är att se över de säkerhetsproblem som en överföring av sessionsvariablerna skulle kunna medföra, så att inte en användare kan bestämma värde på dessa själv genom att posta ett egentillverkat formulär.

Det enda problemet som uppstår vid "migrering allteftersom" är kommunikationen med gamla ASP med sessionsvariabler.

Har man löst problemet med överföring av sessionsvariabler är ingen av dessa strategier särskilt komplicerade och kan lätt kombineras.

7.6.1 Slutsats

Det stora problemet med dessa två strategier är överföringen av sessionsvariabler. I PreCom Bud är detta relativt enkelt att lösa då endast fyra olika sessionsvariabler används, vilka är statiska efter inloggningen. Några förslag till lösningar för både statiska och dynamiska sessionsvariabler disktuterades i kapitel 7.3. I detta arbete har ingen lösning valts för överföring av sessionsvariabler då det beslutades att göra en omskrivning av applikationen. Ett test med att "posta" (http metoden POST) variablerna mellan ASP och ASP.NET har utförts med lyckat resultat. Dock har ingen bedömning av säkerhetsriskerna för denna metod gjorts.

7.7 Kort om prestanda vid migrering

Vid en migrering utan omskrivning av koden kommer vanligtvis prestanda att minska något. Så länge som programkoden befinner sig inom <% %> taggarna så kommer denna att ej i förväg kompileras till MSIL kod utan detta görs då sidan laddas. Man får således ingen prestandavinst på grund av förkompilerad kod. För att öka prestanda krävs att man skriver om i alla fall vissa krävande delar av programmet för att kunna utnyttja de prestandahöjande metoder som finns i ASP.NET som t.ex. cachning och förkompilering till MSIL.

Något som man märker direkt är att den konverterade sidan laddar långsammare än i ASP-originalet vid körning för första gången. När den väl har laddats och sedan laddas om märks ingen skillnad. Att ASP.NET är långsammare vid första laddningen beror till stor del på att källkoden ej är förkompilerad utan kompileras till MSIL för att sedan JIT-kompileras (jämför med ”codebehind” i kapitel 4.8.4). I nya versionen av ASP.NET (version 2) så kommer även koden för själva sidan (.aspx filen) att kunna förkompileras och inte endast ”codebehind-filen” som idag.

Overhead som uppstår vid typkonvertering vid kommunikation med COM-objekt kan i vissa fall leda till en betydande prestandaförlust. En fördel är därför att så ofta som möjligt använda sig av så kallade ”blittable” data typer som innefattar de vanligaste standardtyper såsom integer, long, float och double. Görs detta behövs ingen typkonvertering vilket inte heller medför någon prestandaförlust.

Ett jämförande prestandatest för ASP och ASP.NET redovisas i kapitel 9.3.

7.8 Fördelar med migrering

Beroende på val av migrering kan olika fördelar uppnås. Målet är att i förlängningen kunna utföra all fortsatt utveckling i ASP.NET med alla dess fördelar (se 8.1).

Fördelarna med den migrerade koden är dock vanligtvis inte desamma som fördelarna med ett ASP.NET-program, eftersom en direkt kodöversättning ej resulterar i ett program uppbyggt med ASP.NET teknik såsom användande av Web Controllers och codebehind teknik.

7.9 Nackdelar med migrering

En nackdel är att prestanda kan komma att försämras bland annat på grund av den extra kommunikation och overhead som uppstår vid kommunikation med befintliga COM-objekt. Utvecklandet av nya ASP.NET sidor blir mer eller mindre komplicerat, beroende på hur migreringen gått till och på programstrukturen, då man måste tänka på att sidan ska fungera tillsammans med existerande ASP-sidor. Vid installation av systemet på en server så tillkommer det även extraarbete med att installera både ASP-systemet och ASP.NET-systemet eftersom båda dessa måste fungera då de kör parallellt med varandra och utbyter information.

7.10 Vald lösning för migrering av PreCom Bud

Då funktionaliteten av PreCom Bud är tänkt att integreras i den nya mera generella version av PreCom, vars prototyp implementerades i Implementation 2 (se 8), så valdes att inte migrera PreCom Bud utan att göra en omskrivning i ASP.NET med PreCom Bud som mall och inspirationskälla. Dock tillhandahåller PocketMobile andra ASP-applikationer där testerna av de olika migreringsstrategierna samt övrig kunskap om migrering som har erhållits under arbetets gång och publicerats i denna rapport kan komma till användning.

8 Implementation 2 – Omskrivning av orderhantering till ASP.NET

Syftet med denna implementation var att skapa en prototyp likt PreCom Bud, på vilken migreringsmöjligheter undersöktes i förra kapitlet, fast med stöd även för andra typer av order. Denna implementation kan sägas vara en omskrivning samt utökning av PreCom Bud med .NET teknik. Dessutom utvecklades/utformades ny funktionalitet i samarbete med Johnny Cardell som är yrkesutövande åkare.

Om man vill utnyttja ASP.NET-tekniken till fullo och utnyttja Web Forms och codebehind, vilka beskrivits i kapitel 4.8, kan man överväga en total omskrivning i ASP.NET med det gamla ASP programmet som mall (alt 4b i kapitel 9). Är systemet relativt litet och stora strukturella ändringar är planerade samtidigt som programmeraren är tränad i objektorienterad programmering och .NET så är det fullt tänkbart med en omskrivning.

Innan jag påbörjade denna implementation hade Fredrik Schmidt på PocketMobile konstruerat ett skelett att utgå ifrån med ett antal .aspx sidor samt några tabeller i databasen. Skelettet kom att behöva både modifieras och utökas för att få fram önskad funktionalitet.

Detta kapitel beskriver inte allt som gjordes vid skapandet av prototypen då mycket arbete var av implementeringskaraktär. Det försöker istället beskriva hur prototypen i stort är uppbyggd och hur dess viktigaste funktioner fungerar samt en fokusering på ASP.NET-tekniken som används för att skapa systemet.

IDE som användes i denna implementation var Visual Studio .NET 2003
Databashanteraren var MS SQL-server 7.0.

8.1 Fördelar med omskrivning

Vid en omskrivning kan man till fullo dra nytta av fördelarna med ASP.NET som beskrivs nedan.

- Smidigare att designa och programmera med ”Web Forms”, ”Web Controllers” och ”Codebehind” teknik vilket leder till kortare utvecklingstid och bättre återanvändande av kod.
- 2-3 gånger högre[13] prestanda (servade sidor per minut) jämfört med motsvarande program i ASP enligt Microsoft.
- Bättre säkerhet
- Enklare autentisering.
- Inbyggt stöd för att enkelt låta programmet stödja flera språk.
- Stöd för så kallade ”Web Farms”[14]
- Enklare installation (deployment) av komponenter (assemblies)
- Avancerade cache-funktioner
- Stöd för Web Services

[13] Vid egna mätningar i kapitel 9 blev dock prestanda för ASP.NET ungefär detsamma som för ASP. Däremot är prestanda för ett ASP-system omskrivet till ASP.NET betydligt högre än om systemet varit migrerat till ASP.NET. Under vissa omständigheter är det dock möjligt med denna prestandaökning då t.ex. cache-funktionerna i ASP.NET kan utnyttjas effektivt.

14 En grupp med servrar som sköter exekvering och minneshantering, och resurserna kan fördelas jämt över dessa (load balancing). Olika servrar kan ta hand om på varandra följande förfrågningar.

- Inkluderingsfiler ej längre nödvändiga.
- Enklare att uppdatera och underhålla färdiga program.

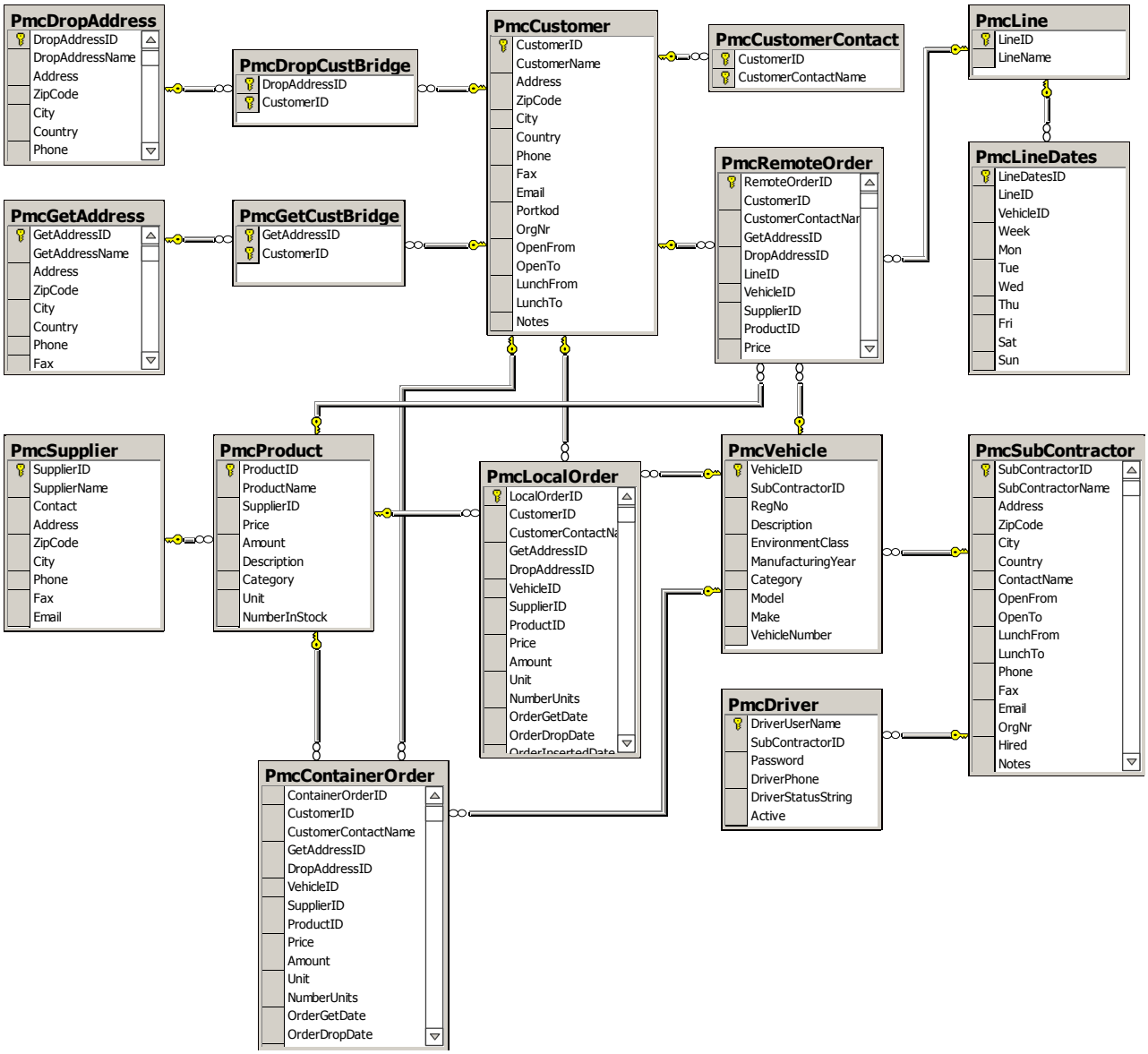
8.2 Nackdelar med omskrivning

Att göra en total omskrivning är i många fall inte rimligt. För ett stort och komplext system kan kostnaden i form av arbetstimmar bli väldigt stor. Samtidigt så tar det tid innan de nya buggarna som introducerats hittas. För att användaren skall märka någon förbättring så måste stort arbete läggas på att förbättra funktionalitet och användbarhet så att det inte upplevs som samma system som det gamla fast med fler buggar.

8.3 Databasdesign

All data som skall långtidslagras i programmet sparas i en SQL-databas. SQL databasen kan således ses som grunden i systemet kring vilken applikationen utvecklats. Det är viktigt att konstruera en bra databasstruktur så att det bland annat inte sker onödig dubbellagring av information, databasen inte blir onödigt stor samt att den medger bra prestanda för sökningar. Dessa faktorer kan ibland väga emot varandra. T.ex. så kan dubbellagring ibland medföra högre prestanda. I denna implementation har god struktur och normalisering[15] av databasen prioriterats framför prestanda då denna varit tillräckligt hög i alla fall. Figur 10 visar relationen mellan tabellerna i databasen. De tabeller som inte tillhörde Fredrik Schmidts ”skelett” utan konstruerades i denna implementation var PmcRemoteOrder, PmcContainerOrder, PmcLocalOrder, PmcLine och PmcLineDates. Dessutom behövdes betydande ändringar göras i de flesta andra tabeller för att få fram önskad funktionalitet.

[15] En process för att minimera mängden redundant data i databasen. Vanligtvis görs detta genom att dela upp informationen i flera olika databastabeller och skapa relationer mellan dessa (se Figur 10).



Figur 10. *Relationen mellan tabellerna i databasen. Relationen ”en till många” åskådliggörs med en nyckel samt ett oändlighetstecken. Bryggorna, t.ex. PmcDropCustBridge, används för att transformera relationen ”många till många” till ”en till många.”. T.ex. så kan en kund (ex Byggkoncernen AB) ha flera adresser där denna vill hämta samt lämna grus. Likväl kan en adress användas av flera olika kunder. Tabellen PmcDropCustBridge innehåller information som binder ihop vilka ”lämnadresser” (PmcDropAdress) som hör till vilka kunder (PmcCustomer).*

I

Tabell 1 nedan visas ett exempel på en databastabell från Figur 10. Denna är tänkt att underlätta förståelsen för hur databasen fungerar, särskilt för den läsare som inte sedan tidigare är bekant med databaser.

Tabell 1. *Exempel på hur tabellen PmcVechicle i databasen kan se ut (en del av kolumnnamnen har förkortats eller tagits bort av utrymmesskäl). Då fordon läggs till via webbgränssnittet i PreCom så är det i denna tabell som dess data lagras. Tabellen innehåller förutom sitt eget id (VehicleID) även id för vilket åkeri det tillhör (SubContractorID). I Figur 10 visas relationen mellan PmcVehicle och PmcSubContractor (innehåller SubContractorID) som en ”många till en” relation. Detta innebär att ett åkeri kan ha flera fordon men ett fordon kan bara tillhöra ett åkeri.*

VehicleID	SubContractorID	RegNo	Description	EnvClass	Year	Cat	Model	Make
1001	1	EBC 112	Lastbil	2	1999	stora fordon	P332	Volvo
1002	1	BFG 332	Pickup	1	2001	mellan fordon	Terrano	Nissan
1005	2	EED 987	Lastbil	3	2003	stora fordon	M442DC	Scania

8.4 Implementering av systemet

Implementeringen innebar programmering i C# för att skapa ”codebehind-filerna” (se 4.8.4) samt konstruktion av användargränssnittet (se Figur 11) med hjälp av Web Forms (se 4.8.1) och ”Web Controls” (se 4.8.2) i Visual Studio .NET 2003. Samtliga grafiska komponenter i Figur 11 är olika ”Web Controls” av klasserna Label, DropDownList, TextBox, TextField, Button, DataGrid och Image. DataGrid-komponenten (hela tabellen under senaste order) visar användbarheten hos mer komplexa ”Web Controls”. Denna binds till en databas så att posterna i denna enkelt kan visas och modifieras, se 4.9.1 för exempel på detta. Givetvis krävs en del kod i bakgrunden för alla finesser (t.ex. ”Ändra-knappen”) men det går betydligt enklare än att manuellt programmera HTML-kod blandad med både klient- och serverskriptkod som man är tvungen att göra i gamla ASP. Faktum är att man ofta inte behöver skriva någon klientskriptkod alls utan denna skapas automatiskt vid rendrering av ”Web Controls”, vilket utförs automatiskt vid körning.

8.4.1 Orderfunktion

En av de viktigaste funktionerna i PreCom är att kunna lägga order. Till skillnad från PreCom bud finns det i denna prototyp flera olika ordertyper att välja mellan. De olika ordertyperna har parametrar som är specifika för just dessa. Namnet inom parentes anger vilken databas i Figur 10 som innehåller data om respektive order.

8.4.1.1 Grus/Lokal (PmcLocalOrder)

En enklare form av order för bland annat lokala grustransporter till byggen.

8.4.1.2 Container (PmcContainerOrder)

Denna typ av order innehåller bland annat följande information för att hämta och lämna containrar:

- Containerhyra (pris per dag)
- Deponiavgift (sorterat medför lägre pris)
- Olika grupper med olika pris (träavfall, betong etc.)

8.4.1.3 Fjärr (PmcRemoteOrder)

Order som körs längre sträckor enligt förbestämda så kallade linjer, t.ex. från Stockholm till Göteborg. På undermenyn ”Linjer” (syns ej i Figur 11) till Åkerier (syns i Figur 11) kan man välja vilka fordon som kör vilka linjer vilka dagar och vilka veckor. T.ex. kan fordon med

registreringsnummer BFG 332 sättas att köra linjen ”Sthlm-Gbg” onsdagar och fredagar på jämna veckonummer.

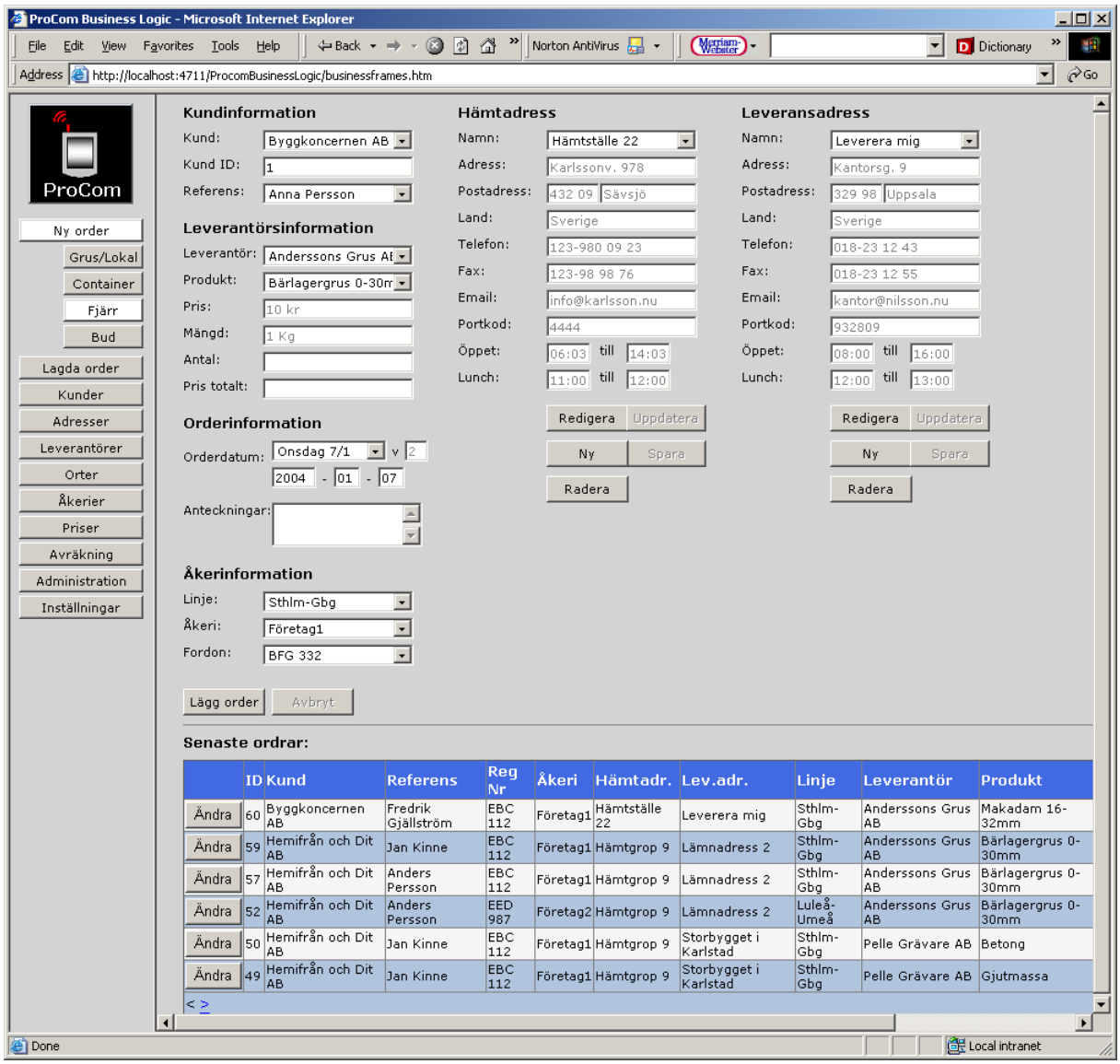
Väljer man sedan linjen ”Sthlm-Gbg” på fjärrordersidan så visas alla åkerier som har något fordon som kör den linjen den dagen och den veckan som valts. Väljer man sedan t.ex. åkeriet ”Företag1” som i Figur 11 så visas alla fordon som tillhör detta åkeri samt kör den valda linjen den dag och vecka som valts.

I Figur 10 visas tabellerna PmcLine och PmcLineDates som löser denna uppgift.

Denna ordertyp var den som krävde mest arbete för att implementera.

8.4.1.4 Bud (ingen databastabell skapad ännu)

Denna ordertyp används för att skicka enklare lokala försändelser. Implementeringen kommer att till stor del likna den för grus- och lokalorder.



Figur 11. *Skärmdump från systemet kört i Internet Explorer 6.0. Till vänster i bilden visas de olika huvudmenyerna samt undermenyn till ”Ny Order” (LeftControl.aspx). Till höger visas sidan som används för att lägga in en ny fjärrorder (NewOrder.aspx). En ”tracelogg” för NewOrder.aspx återges i Appendix B (se 13).*

8.4.2 Mellanlager

Ett mellanlager skulle kunna vara bra, speciellt om programmet växer och blir stort och komplext. Tyvärr räckte inte tiden till för att konstruera ett välgenomtänkt mellanlager som dessutom enkelt skulle kunna anpassas för andra databashanterare än Microsofts SQL server (t.ex. MySQL som är gratis). Istället anropas SQL-databasen direkt från olika metoder i klasserna för sidorna. Med tanke på att detta kommer att bli en första prototyp med en till en början ganska enkel konstruktion så är detta inget större problem. Vidare så har heller inga lagrade procedurer[16] använts, även detta med tanke på att andra databashanterare ska kunna användas. Det finns även en hel del nackdelar med mellanlager såsom ökad komplexitet, säkerhetsproblem och försämrad prestanda, som framgår i testet i kapitel 9.3. Information och riktlinjer för hur man designar ett program med flerlagersarkitektur (n-tier) finns i Microsofts ”Patterns and practices”.

8.5 Externa moduler

Från Pocketmobiles sida finns önskemål att kunna ansluta externa moduler[17]. De två moduler som skulle undersökas i detta arbete var en faktureringsmodul och en positioneringsmodul varav så gott som allt arbete lades på den senare.

8.5.1 Kort om faktureringsmodulen

En faktureringsmodul ska kunna tillhandahålla funktionalitet för att utbyta redovisningsdata med externa ekonomisystem.

För att modulen ska fungera med de flesta ekonomisystem på marknaden krävs att modulen genererar en så kallad SIE-fil.

SIE är en standard för att enkelt kunna flytta redovisningsdata mellan olika program, oavsett från vilka leverantörer programmen kommer [18].

Då utvecklingstiden var begränsad så utvecklades ingen faktureringsmodul utan arbetet inriktades istället på positioneringsmodulen som kan användas i flera andra av PocketMobile’s applikationer. Bland annat så används den för närvarande i ett pilotprojekt med skatteverket.

8.5.2 Positioneringsmodul

I det nya PreCom är det tänkt att det ska finnas en positioneringsfunktion. Denna ska fungera så att man väljer ett antal fordon varefter man klickar på knappen ”positionera”, varvid en karta med alla fordon utritade visas (se Figur 12). Kraven på modulen var förutom att den skulle vara enkel att kommunicera med, även att den utan modifikation skulle kunna användas i PocketMobile’s andra programvaror.

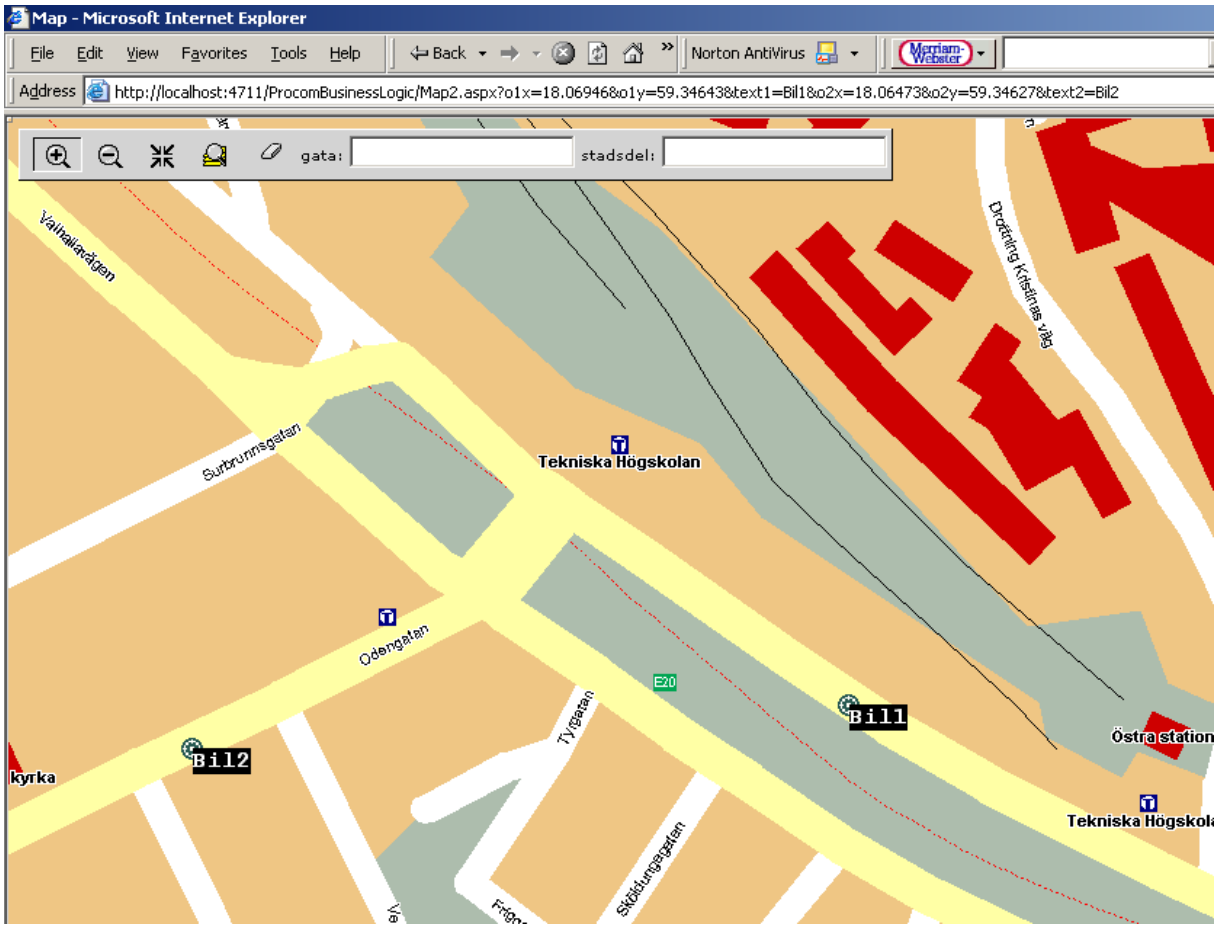
[16] Lagrade procedurer (eng: stored procedures) är subrutiner lagrade i databasen och exekverade av databashanteraren. Subrutinen är vanligtvis skriven i samma språk som databas-språket (här SQL). Olika databashanterare kan dock ha något olika syntax för koden i de lagrade procedurerna. En stor fördel med att använda lagrade procedurer är prestanda, då dessa kan förkompileras och reducera nätverkstrafiken avsevärt vid långa komplicerade förfrågningar och korta resultat.

[17] Modul i detta sammanhang är nödvändigtvis ej detsamma som definitionen för en modul i .NET (se fotnot 9 sid 14). Med modul i detta sammanhang menas istället ett externt program eller system som genom t.ex. Web Services kan anslutas till det aktuella programmet och tillhandahålla viktig information som t.ex. GPS-koordinater eller prisuppgifter från ett faktureringssystem.

För kommunikation till positioneringsmodulen användes ”querystring” (QS), som är strängen som matas in efter ett ”?” i URL:en. Användningen av QS beskrivs bäst med följande exempel.

Exempel 1: Användning av QS för att kommunikation
URL: http://localhost:4711/ProcomBusinessLogic/Map2.aspx?o1x=18.06946&o1y=59.34643&text1=Bil1&o2x=18.06473&o2y=59.34627&text2=Bil2

I ovanstående URL så är det strängen ”o1x=18.06946&o1y=59.34643&text1=Bil1&o2x=18.06473&o2y=59.34627&text2=Bil2” som är QS. Informationen i QS beskriver positionen (longitud och latitud) för kartans objekt samt en beskrivande text för dessa. I detta QS ovan finns information om två objekt. Det första är en bil som befinner sig på Valhallavägen i Stockholm. Det andra är även detta en bil i Stockholm som befinner sig på Odengatan.



Figur 12. *Kartan som genereras av positioneringsmodulen.*

Kortfattad beskrivning av postitioneringsmodulens funktion:
Inläsning av de olika objekten från QS eller SessionState (då någon av funktionerna i modulens verktygsfält används. T.ex. ”zoomin” eller ”zoomout”). Uträkning av vilken del av

kartan som ska visas samt generering av en ny QS som skickas till E-MAPI[18]. E-MAPI skickar då tillbaks en textfil som innehåller URL till kartbilden (i .gif-format) samt en mängd andra parametrar såsom kartans centerkoordinat och zoomfaktor, koordinater för kartans hörn samt de olika objektens positioner. Dessa sparas för att användas vid nästa anrop till E-MAPI, då någon av verktygsfunktionerna används.

Exempel 2: ”Switchsatsen” som utför kommando från verktygsfältet (knapparna med förstoringsglas etc i Figur 12)

```
switch (maptoolcmd) //Utför kommando från toolbar
   {
      case "zoomin":
         PartOfMapToShow = zoomin(startx, starty, endx, endy);
         break;

      case "zoomout": //Samma funktion som newcenter fast med ny zoomfaktor
         double newzoom = Convert.ToDouble(hz, nfi)*STANDARD_ZOOM_FACTOR;
         PartOfMapToShow = zoom(newzoom.ToString(nfi));
         break;

      case "newcenter":
         PartOfMapToShow = zoom(hz);   //Samma zoomfaktor som tidigare
         break;                        //medför endast omcentering.

      case "find": //Visa samma karta som innan om inget svar hittas
         PartOfMapToShow = "cx=" + hx + "&cy=" + hy + "&z=" + hz + "&sx=" +
                           sizex + "&sy=" + sizey + "&ss=" + street +
                           "&srm=101" + "&sc=" + city + "&srt=2";
         break;

      case "fullextent": //Visa hela sverige
         PartOfMapToShow = "cx=" + middle_swedenx + "&cy=" + middle_swedeny
          + "&z=" + zoom_whole_sweden + "&sx=" + sizex + "&sy=" + sizey;
         break;

      case "erasefind": //Tar bort alla sökpositioner på kartan
         if (Session["FindHT"] != null)
            Session.Remove("FindHT");
         break;
   }
```

Strängen ”PartOfMapToShow” i Exempel 2 innehåller den del av QS som E-MAPI använder för att generera en kartbild. Funktionerna zoom och zoomin skapades för att generera QS till E-MAPI för att generera en ny kartbild med en ny zoomfaktor respektive generera en ny kartbild med koordinaterna för en rektangel (startx, starty, endx, endy), utritad på den ursprungliga kartbilden med zoomin-verktyget, som begränsning för vad som ska visas efter inzoomningen.

En stor utmaning vid skapandet av positioneringsmodulen var konstruktionen av verktygsfältet som är skriven i javascript. Kod för detta fanns redan i en äldre tillämpning som PocketMobile använde. Denna var dock skriven i en blandning av javascript och javascript genererat från en ASP-sida, vilket komplicerade förfarandet att kopiera den del av koden som

[18] Easy-Mapper (E-MAPI) [19] är ett API som utvecklats av företaget Cartesia och används för att generera kartbilder. Några funktioner som finns implementerade i E-MAPI är: Centrering och in/utzoomning kring en viss koordinat, sökning på gata och eller stadsdel samt utritning av objekt.

kunde vara till nytta för just denna modul. Det var dock bara en liten del av den ursprungliga javascriptkoden som användas samt modifierades. I övrigt så är hela modulen utvecklad från grunden med C# och ASP.NET.

En annan stor utmaning var att hålla reda på alla objekt (dels från positionering och dels från adress-sökning). För att klara av detta konstruerades ett särskilt positionsobjekt (Position) som fick egenskaperna X för att läsa eller skriva x-koordinaten), Y för att läsa eller skriva y-koordinaten samt Name (beskrivning av positionen. T.ex. Bil1). Även metoder för att bland annat generera en delsträng av QS för E-MAPI och räkna ut inom vilket område samtliga objekt befann sig implementerades. Samtliga positionsobjekt som skapades lades i en ”Hashtable” som sedan skrevs till ”SessionState” för att senare kunna utritas vid en omladdning av sidan vid t.ex. en inzoomning.

8.6 Analys av systemets exekvering med ”Trace-funktionen”

Genom att aktivera ”trace-funktionen” i ASP.NET så kan man få väldigt detaljerad information om tidsåtgången för en .aspx sida. Denna information är mycket värdefull för att identifiera flaskhalsar och optimera där det behövs bäst. T.ex. så ser man hur lång tid det tar att rendrera olika ”Web Controls” (se 4.8.2).

Dessutom visas bland annat storleken för ViewState för samtliga ”Web Controls” på sidan. Detta görs även för så kallade ”child controls” som är byggstenarna i ”Web Controls” som t.ex. DataGrid. DataGrid som representerar en tabell har bland annat följande ”child controls”: DataGridTable, DataGridItem, TableCell och DataGridLinkButton. Var och en av dessa ”child controls” har sin egen ViewState vilket när de summeras kan bli väldigt stor.

I ”Appendix: B Trace” (kapitel 13) visas en ”tracelogg” för sidan NewOrder.aspx (se Figur 11). I denna logg under ”Trace Information” (13.1.2) kan man utläsa hur lång tid olika faser av exekvering och rendrering av sidan tar. Ur ”Trace Information” kan utläsas att 4 olika händelser tar betydligt längre tid än andra händelser. Dessa är ”LoadViewState”, ”Raise ChangedEvents”, ”SaveViewState” och “Render”. Av dessa tar “Render” (rendrering av komponenterna på sidan) längst tid med 27% av den totala programkörningstiden. Strax därefter kommer ”LoadViewState” och ”SaveViewState” med vardera ca 23% av den totala programkörningstiden. Tiden det tar att ladda och spara ”ViewState” tar alltså nästan 50% av den totala programkörningstiden. Detta verkar rimligt och bekräftar nackdelarna för ”ViewState” som nämndes i kapitel 5.6.1. Storleken av ViewState och den rendrerade storleken för de olika komponenterna på sidan framgår av ”Control Tree” (13.1.3).

Tiden för rendreringen är till stor del beroende på antal komponenter (”Web Controls”) och dess komplexitet, samt ”ViewState” (innehållet i komponenterna hämtas ofta från ViewState).

Vid optimering är det därför av stor vikt att rikta in sig på att begränsa storleken av ”ViewState” genom att begränsa eller stänga av denna då detta är möjligt samt att använda sig av cache som kan spara en hel sida och hindra att denna rendreras om på nytt vid en omladdning.

De övriga delarna i ”traceloggen” är även dessa av stor nytta vid t.ex. felsökning. De innehåller all annan information om sidan, såsom sessionsvariabler, innehållet i ”cookies” och servervariabler.

8.6.1 Aktivering av ”Trace-funktionen”

Aktivering av ”Trace-funktionen” kan göras på flera sätt. Ett är att lägga till attributet Trace=”true” i <%@Page … %> taggen överst på den sida man vill köra Trace på. Ett annat är att sätta isEnabled i Trace-objektet till sant (Trace.isEnabled = true).
Vill man använda Trace för alla sidor i en applikation, så kan man modifiera ”trace-taggen” i filen Web.config. Nedan demonstreras ett exempel på detta som tillåter Trace att visas i ett separat webbfönster.

```
<trace
   enabled="true"
   requestLimit="10"
   pageOutput="false"
   traceMode="SortByTime"
   localOnly="true"
/>
```

9 Analys, utvärdering och jämförelse

I detta kapitel undersöks hur pass stora fördelarna eller eventuella nackdelar med en migrering eller omskrivning till .NET är. Givetvis beror fördelarna eller nackdelarna på vilken typ av program som skrivs. I denna jämförelse kommer funktionalitet förekommande i ”Implementation 1 – Migrering av orderhantering till ASP.NET” och ”Implementation 2 – Omskrivning av orderhantering till ASP.NET” att jämföras i huvudsak tekniskt men i viss utsträckning även användarmässigt. Dessutom dras flera mer generella slutsatser om migrering med implementation 1 som underlag.

9.1 Alternativ vid migrering

I Implementation 1 i kapitel 7 framkom att följande fyra alternativ vid övervägning av migrering från ASP till ASP.NET är möjliga:

1. Migrera all kod.
2. Migrera ett lager i taget
3. Migrering efter hand. All fortsatt utveckling i ASP.NET
4. Inte migrera[19] alls, vilket kan delas upp i två delfall:
   a) Fortsätta all utveckling i ASP
   b) Göra en total omskrivning av systemet i ASP.NET med det gamla som mall

Vilket alternativ man väljer beror på flera faktorer. Bland annat hur mycket kod som det rör sig om, hur mycket man tänkt vidareutveckla applikationen i framtiden, vilka mellanlager som finns, speciallösningar och användandet av sessionsvariabler. För att avgöra vilket val som passar bäst måste man skaffa sig en bild av hur det nuvarande systemet ser ut och vad man kan vinna på att göra en migrering.

Alt 1 Det första alternativet är endast att tänka på om systemet är så pass stort att en total omskrivning skulle vara alltför tidsödande men samtidigt så enkelt att en total migrering på en gång inte blir för krångligt. Vidare så bör det finnas stora fördelar med en migrering. T.ex. stor fördel vid användning av någon ny .NET teknik som kan implementeras utan stora ändringar. Har man tänkt göra stora strukturella ändringar rekommenderas istället en omskrivning av programmet (se Implementation 2 kap. 8). Skillnaden mot en omskrivning är att man bevarar så mycket av den befintliga koden som möjligt istället för att skriva om allt från början. Det är inte säkert att man vinner särskilt mycket på detta då det krävs mycket omskrivning i både mellanlagret och presentationslagret för att byta ut t.ex. ASP’s RecordSet mot ASP.NET’s DataSet och de skillnader som då uppkommer. I praktiken riskerar man att skapa lika mycket jobb som med en omskrivning fast resultatet blir ett mycket sämre program eftersom man begränsas av hur systemet var byggt med den gamla tekniken.

Alt 2 Även detta alternativ innebär en mycket stor förändring. Är det kritiskt att systemet skall vara stabilt så bör man överväga detta alternativ extra noga. Största fördelen fås ofta med migration av presentationslagret där ny ASP.NET-teknik såsom förbättrad

[19] I denna rapport räknas en total omskrivning av ett system ej som en migrering eftersom man vanligtvis inte kan använda gammal kod särskilt mycket utan måste, om man vill lösa det på ett bra sätt, skriva om det mesta från grunden med användning av nya tekniker. Även om det mesta konstrueras om på nytt kan man använda det gamla systemet som mall vid utvecklingen, vilket kan vara tidsbesparande.
Notera att alternativet då fortsatt utveckling av applikationen sker i ASP.NET samtidigt som den gamla koden lämnas intakt räknas som en migrering (jämför migreringsstrategin ”Migrering allteftersom” i kapitel 6.4).

autentisering, databindning samt ”Web Forms” kan användas. Dessutom slipper man tänka på problem som uppstår med utbyte av variabler (ex. sessionsvariabler, se 5.6.2) eftersom alla sidor i presentationslagret körs under ASP.NET. Dock krävs viss (ibland betydande) omskrivning för anpassning för ”Web Controls” och konvertering av t.ex. ADO’s RecordSet till ASP.NET’s DataSet för användandet av databindning.

Alt 3 Resultatet av detta arbete visar att detta alternativ ofta är det bästa vid större komplexa färdigutvecklade system. För att åstadkomma denna typ av migrering använder man i praktiken en kombination av migreringsstrategierna ”6.3 Migrering sida för sida” och ”6.4 Migrering allteftersom”. Förekommer extensiv användning av sessionsvariabler så tillkommer valet att välja metod för att lösa detta problem. De nya sidorna utvecklas i ASP.NET. De gamla sidorna migreras efter hand/behov. Som visats i Implementation 1 så finns det dock en hel del saker som man måste analysera innan man kan börja. T.ex. det gamla systemets uppbyggnad och vilka sidor som kan migreras lättast och ger mest förbättring.

Alt 4

a) Om man bedömer att systemet inte behöver utvecklas särskilt mycket samt att man inte har nytta av de fördelar som ASP.NET har kan alternativet att bibehålla ASP vara det bästa. Som nämnts tidigare i rapporten går det bra att köra ASP.NET sidor parallellt på samma webbserver men ingen kommunikation mellan de separata systemen kommer att utföras.
b) Om systemet inte är alltför komplext och stora ändringar i programarkitekturen kan komma att ske, så kan en total omskrivning av programmet vara den bästa lösningen. Risk finns för att många nya buggar kommer att introduceras då ett väl testat system skrivs om på nytt. PreCom Bud behövde förbättras och integreras i ett mera generellt system vilket gjorde omskrivning till den bästa lösningen för detta system.

9.2 Risker och problem vid migrering

En stor risk vid översättningen av kod är introducerandet av nya buggar som enbart visar sig vid körning av programmet under speciella förutsättningar. För att minimera denna risk så rekommenderas att använda ett migreringsverktyg för att migrera koden från ASP’s vbscript till .Net’s Visual Basic eller C#. Det är också viktigt att göra extensiva tester av programmet innan man anser sig vara färdig med migreringen. Vid arbetet med implementation 1 så upptäcktes vilken tidsödande och riskfylld process det är att migrera för hand. Även om det som exemplet nedan visar kan finnas risker även med ett migreringsverktyg som översätter koden så kommer introducerandet av nya buggar att minimeras.

9.2.1 Exempel 1 – Migrering för hand

Ett exempel på fel som kan uppstå vid migrering för hand. (Kod i Visual Basic.)

I filen sida1.aspx finns följande kod.

```
Session(”KundID”) = LoginCheck(”KundID”)
```

På något ställe i sida2.aspx finns följande kod

```
Dim KundID As String = Session(”KundID”)
```

Felet som uppstår är att strängen KundID på sida2.aspx inte kan tilldelas det förmodade strängvärdet returnerat från Session(”KundID”) (Se 5.6.2 om Session State). Detta beroende

på att LoginCheck(”KundID”) returnerar ett ADO RecordSet objekt och inte en sträng. Detta noteras inte som något fel på sida1 eftersom valfritt objekt kan lagras i Session State. På sida2.aspx inträffar däremot ett fel att Session(”KundID”) inte kan konverteras (eng: cast) till String.

Felet som uppstår berodde på att kodraden på sida1.aspx glömts bort att modifieras. I ASP så kommer LoginCheck(”KundID”) returnera en sträng som är standardvärdet (default properties) för RecordSet objektet. För att det ska fungera i .NET så krävs att man använder egenskapen Value i ADO objektet, vilket innebär att en sträng returneras.

Nedan visas hur den korrekta raden i sida1.aspx ser ut.

```
Session(”KundID”) = LoginCheck(”KundID”).Value
```

9.2.2 Exempel 2 – Migrering med verktyg

Så länge som migreringsverktygen inte är perfekta finns även här en risk för att nya buggar introduceras som enbart märks vid körning under speciella omständigheter. Nedan visas två exempel med omständigheter som kan leda till programkörningsfel.

- Vill man ta reda på hur stor en cookie är i ASP (VBScript) så fungerar detta även om cookien är tom. En tom ”cookie” i ASP returneras nämligen som en tom sträng (””). En ”ren” översättning till ASP.NET (VB) skulle däremot inte fungera i samtliga fall eftersom en tom ”cookie” returnerar ”null”. Det som gör denna bugg svår att upptäcka är att programmet kanske för det mesta fungerar då ”cookien” ej är tom i 99% av körningarna. I de fall ”cookien” är tom uppstår ett programkörningsfel.

- Defaultvärdet för en variabel är Empty i ASP. I ASP.NET är det typ-specifikt men null för alla variabler. Vidare så är de båda uttrycken Empty == ”” och Empty == 0 sanna i ASP. Men i ASP.NET så är de båda uttrycken null != ”” och null != 0 sanna. Även detta måste migreringsverktyget lösa på korrekt sätt för att programkörningsfel inte ska uppstå.

9.2.3 Svårigheter med GUI

Även om man har lyckats bra med konverteringen av presentationslagret till ASP.NET så kan det bli problem då man vill vidareutveckla programmet. När man fortsätter utvecklingen i ASP.NET vill man gärna använda sig av ”Web Controls” och Codebehind vilket ger en viss standard på utseendet som kanske inte stämmer överrens med utseendet för de gamla ASP sidorna. Om det inte är något krav att sidorna ska se precis likadana ut som tidigare kan man göra dem så lika som det går och sedan steg för steg migrera de gamla sidorna till ”äkta” .NET teknik med ”Web Controls” och codebehind. I annat fall tillkommer en hel del extraarbete i och med att mycket kod kan behövas programmera manuellt i HTML precis som i ASP och då har man missat en del av poängen med migreringen.

9.3 Jämförelse av prestanda för ASP och ASP.NET vid olika specialfall

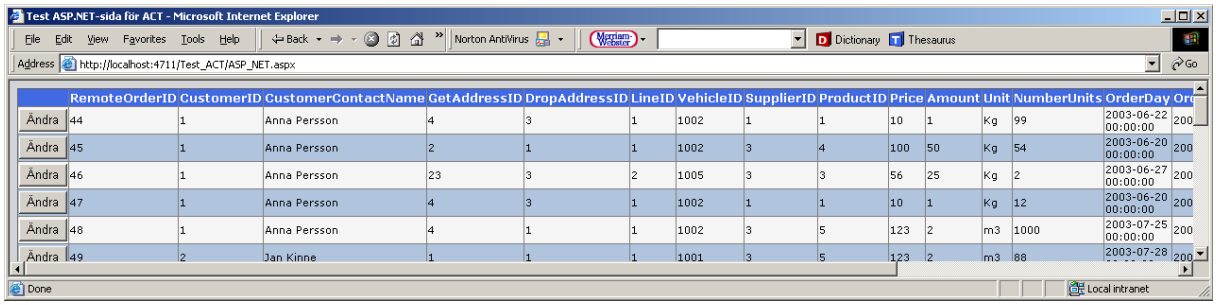
En av anledningarna för migrering till ASP.NET kan vara prestanda. Därför kommer en prestandajämförelse mellan ASP och ASP.NET att genomföras i detta kapitel. Omständigheterna vid testningen är vanliga i PocketMobiles webbapplikationer men samtidigt även i många andra webbapplikationer. Av praktiska skäl kunde endast vissa delar av systemen testas.

9.3.1 Metod för jämförelse

För att en jämförelse ska vara av nytta är det viktigt att använda en metod som återspeglar verkligheten så bra som möjligt.

Flera olika testmetoder övervägdes och utprovades innan det slutgiltiga testet med Microsoft Application Test Center (ACT) genomfördes (se 9.3.1.1 nedan). Det gällde att hitta en metod som på ett så rättvist sätt som möjligt kunde jämföra prestanda hos program skrivna i ASP och ASP.NET. Från början var det tänkt att göra test med en verklig sida från Implementation 2, t.ex. NewOrder.aspx som visas i Figur 11. Denna sida skulle dock vara väldigt tidskrävande att översätta till ASP samt att resultaten av ett prestandatest skulle vara svåra att jämföra. Därför bestämdes att göra ett test med en enklare sida. Den sida som då valdes var en sida som endast innehöll tabellen från senaste order i NewOrder.aspx. Sidan skapades i ASP.NET och en motsvarande sida byggdes upp i ASP. Vid några inledande testkörningar visade sig även denna sida vara för komplicerad eller rättare sagt databasåtkomsten tog för lång tid. Tabellen i sidan var uppbyggd av en databasvy (en tabell i databasen som skapats genom ett urval av andra tabeller) och detta medförde att databasåtkomsten tog mellan 70 och 90% av programkörningstiden vilket innebar att prestandaskillnaden med att använda ASP respektive ASP.NET skulle bli väldigt liten. Det slutliga valet föll därför på att använda endast en databastabell istället.

Olika fall som kan vara intressanta att jämföra togs fram. En sida där cachen skulle kunna utnyttjas var den då hela databastabellen lästes och en tabell skapades av denna. Den andra sidan som testades var en sida som skapades helt dynamiskt och där slumpen avgjorde vilka poster i databastabellen som skulle visas. Dessa fall implementerades som separata testprogram, vilka agerade på följande sätt: Ett antal poster lästes i en databastabell och därefter skapades en webbsida där samma tabell presenterades (se Figur 13). Slutresultatet blir i stort sett identiskt för samtliga testade program. D.v.s. den genererade html-sidan är uppbyggd på samma sätt och innehåller samma information med undantag för obetydlig dold dokumentinformation av begränsad storlek.



Figur 13. *Tabellen som skapades vid testkörningarna (i fallet då posterna valdes ut med slump så skapades tabellen endast med de rader som slumpmässigt valts ut).*

Prestanda hos följande olika program, speciellt skapade för detta test, har jämförts:

- **ASP**: En ASP-sida skriven i VBScript med direkt koppling till databasen.
- **MeASP**: Samma som **ASP** men med anslutning till databasen via ett mellanlager (DBConn) på samma sätt som i Figur 8. DBConn är ett COM objekt skrivet i Visual Basic.
- **MiASPX**: **ASP** migrerad till en ASP.NET-sida med programmet ASP2ASPX som genererar Visual Basic kod. I detta fall har en interop assembly (se 5.1.1) skapats för åtkomst till olika ADO-objekt (se 5.3 för jämförelse av ADO och ADO.NET)

- **MiMeASPX**: **MeASP** migrerad till en ASP.NET-sida med programmet ASP2ASPX som genererar Visual Basic kod. I detta fall tillkommer kommunikation med COM-objektet DBConn som i sin tur kommunicerar med ADO (se Figur 8). Denna version borde vara det långsammaste med tanke på mängden COM-kommunikation och den overhead som läggs till av .NET för detta. Även typkonverteringen vid COM-kommunikationen kommer ha en negativ inverkan på prestanda.
- **ASPX**: **ASP** omskriven till ASP.NET med utnyttjande av ASP.NET teknik såsom ”Codebehind” och ”Web Controls” (DataGrid i detta fall). Precis som **ASP** har detta program en direkt koppling till databasen. Eftersom endast läsning testades så stängdes View State av för hela applikationen för att begränsa datamängden. Programmet är skrivet i C#, men detta bör inte ha någon betydelse eftersom all källkod oavsett programmeringsspråk kompileras till den för .NET gemensamma MSIL-bytekoden (se 4.4).

Följande olika testfall genomfördes för de olika programmen:

- **Läsning av slumpmässiga poster**: Sannolikheten för att post nr *i* ska visas i tabellen på webbsidan är 50%. Detta medför att olika poster i databasen väljs ut varje gång och den skapade tabellen ser olika ut för varje körning. Databashanteraren får i detta fall arbeta mera och det finns ingen möjlighet för sidan att lagras i något cacheminne då den ser olika ut varje gång. Dessutom blir tabellen i genomsnitt hälften så stor vilket minskar tiden för att skapa denna samt överföringstiden för att hämta sidan.
- **Läsning av ej slumpmässiga poster**: I detta fall läses hela tabellen från databasen varje gång och skrivs ut på webbsidan. Cache kan då komma att användas flitigare både hos databashanteraren och på webbsidan.

De två ovanstående testfallen genomfördes med både 1 och 20 samtidiga användare. Detta för att se om resultaten skilde sig beroende på hur många användare som samtidigt försöker att hämta sidorna. För att undvika alltför många ”http-errors” (Response code 403) så gör de simultana användarna en paus (med ACT-funktionen Test.sleep()) på 300 ms innan nästa sida efterfrågas. I tabellen så visas endast lyckade förfrågningar vilket har räknats ur genom att ta det totala antalet RPS minus antalet ”http-errors” per sekund.

Testet gick till på följande sätt:
Samtliga testprogram kördes i ACT under 1 minuts tid. ”Warmup time”, d.v.s. den tid som testet körs innan mätning påbörjas sattes till 30 sekunder. Nedan visas ACT skriptet som användes och anropas kontinuerligt under hela testkörningstiden (anropet till sleep utfördes endast då flera simultana användare simulerades).

```
Call Test.Sleep(300) 'sleep for 300 ms before request
Test.SendRequest("http://localhost:4711/Test_ACT/namn på testat program")
```

9.3.1.1 Microsoft Application Center Test (ACT)

“Microsoft Application Center Test” är ett program för att göra prestandamätningar för webbapplikationer [16]. Programmet kan användas för att se hur en webbapplikation klarar olika uppgifter under hög belastning med många simultana användare även kallat ”stress test”. Till grund för testningen ligger ett visualbasic- eller javaskript. I detta script kan en mängd http-kommandon köras. T.ex. finns det objekt för att skapa en anslutning (Connection-objektet), skapa en förfrågan (Request-objektet), ta emot svar (Response-objektet), skapa HTML-headers (Header-objektet) samt läsa cookies (Cookies-objektet). Detta tillsammans med alla möjligheter i skriptspråket samt inställningsmöjligheter i ACT gör det möjligt att

utföra mycket avancerade prestandatest av webbapplikationer under omständigheter som så mycket som möjligt efterliknar verkligt användande av riktiga användare.

Viktigt att notera är att ACT mäter både webbserverns, eventuell databasserver och webbapplikationens prestanda. Något som man får tänka på då resultat från ACT jämförs och analyseras.

9.3.2 Resultat och diskussion

Resultaten från prestandatestet redovisas i Tabell 2 och Tabell 3.

Tabell 2. *Siffrorna i tabellen har enheten RPS (Requests Per Second) som anger hur många förfrågningar (hämtningar) per sekund som är möjliga för en webbsida. Antalet användare som anges är antalet användare som samtidigt ansluter eller försöker att ansluta till sidan. Procenttalet inom parentes anger hur stor andel förfrågningar som lyckades (Response code 200). Övriga förfrågningar har ”Response code 403”, vilket betyder att servern har uppfattat förfrågan men vägrar att genomföra den, t.ex. på grund av att den får fler förfrågningar än den klarar av.*

	ASP	MeASP	MiASPX	MiMeASPX	ASPX
Läsning av slumpmässiga poster. 1 användare.	28,6 (100%)	17,8 (100%)	17,4 (100%)	13,8 (100%)	28,1 (100%)
Läsning av ej slumpmässiga poster. 1 användare.	25,5 (100%)	15,5 (100%)	12,3 (100%)	9,8 (100%)	24,9 (100%)
Läsning av slumpmässiga poster. 20 användare.	25,0 (55%)	13,0 (27%)	14,1 (31%)	11,2 (26%)	24,6 (55%)
Läsning av ej slumpmässiga poster. 20 användare.	24,6 (85%)	11,8 (35%)	9,9 (22%)	8,1 (21%)	21,5 (48%)

Tabell 3. *Antalet mottagna/skickade kilobytes per sekund samt summan av dessa för de olika programmen vid körning med 1 användare. Siffrorna inom parentes avser värdet för läsning av slumpmässiga poster.*

	Skickade kB/s	Mottagna kB/s	Total bandbredd kB/s
ASP	10,0 (11,3)	377,2 (227,7)	387,2 (239,0)
MeASP	6,1 (7,1)	230,0 (141,7)	236,0 (148,7)
MiASPX	4,9 (7,0)	187,3 (142,9)	192,3 (150,0)
MiMeASPX	3,9 (5,6)	149,0 (111,8)	152,9 (117,4)
ASPX	8,3 (9,5)	381,0 (231,9)	389,3 (241,4)

Av Tabell 2 kan man dra slutsatsen att för detta test så är prestandaskillnaden mellan ett program skrivet i ASP och ett skrivet med ASP.NET teknik obetydliga. Faktiskt är det så att ASP är en aning snabbare. Vidare ser man att mellanlagret har en stor negativ inverkan på prestanda. Detta bland annat beroende på COM-kommunikationen och den ökande beräkningstid som går åt i mellanlagret.

Prestandaskillnaden mellan **MiASPX** och **MiMeASPX** är inte alls lika stor som mellan **ASP** och **MeASP** troligen beroende på att **MiASPX** redan kommunicerar med COM och att det bara är lite ytterliggare kommunikation med COM för **MiMeASPX**. En stor prestandaförlust kommer från läsning av ADO’s RecordSet-objekt som läses via COM-interoperabilitet. Detta på grund av att anrop till funktionen **MoveNext**, som anropas en gång för varje rad i tabellen, i RecordSet-Objektet skapar overhead som försämrar prestanda [3].

I Tabell 3 syns att den totala bandbredden för slumpmässiga respektive ej slumpmässiga läsningar förhåller sig ganska bra till RPS-värdena i Tabell 2. Några skillnader kan dock observeras. Bland annat så har programmet **ASPX** en något högre bandbredd än **ASP** trots att det har något lägre RPS. Detta kan förklaras med att sidan som ASP.NET programmet genererar har några fler rader ej synlig information. För den slumpmässiga sidan så noteras en något högre bandbredd på skickade bytes vilket beror på dess högre RPS och att alla förfrågningar för samma sida är lika stora. Den mottagna bandbredden är dock ganska mycket lägre än för den icke slumpmässigt genererade sidan. Detta beroende på att åtkomsttiden till databasen har ökat. Däremot har sidstorleken krymp med omkring 50% på grund av att endast hälften av posterna i genomsnitt skrivs till HTML-tabellen, vilket medför att RPS ändå kan bli högre.

Eftersom funktionaliteten i ASP är betydligt mer begränsad än i ASP.NET så behövs ofta kommunikation till olika COM-objekt vilket bevisligen minskar prestanda drastiskt.

9.3.3 Påverkande faktorer/felkällor

Faktorer som påverkar RPS är givetvis hårdvaran men även andra processer som körs på processorn. Därför är det viktigt att stänga ner så många andra processer som möjligt innan testet körs. Trots att data endast läses ur en tabell så är data-acesstiden ej obetydlig särskilt för läsningarna av slumpmässiga poster.

9.4 Produktivitet

Efter erfarenheter med både ASP och ASP.NET framkom det att programutvecklingen för det mesta går betydligt snabbare och smidigare med ASP.NET. Huvudanledningen är uppdelningen av källkod och HTML samt de hjälpmedel som finns i Visual Studio .NET 2003 för att skapa grafiska gränssnitt med hjälp av ”Web Forms” och ”Web Controls”. Möjligheten att slippa kommunikation med COM-objekt som är fallet vid t.ex. utveckling av mellanlager i ASP är också en stor fördel.

Som exempel för att visa detta tas programmen **ASP** och **ASPX** från prestandatestet i kapitel 9.3 som läste data från en databas och presenterade resultatet i en tabell på webbsidan.

ASPX: I detta ASP.NET-program finns objektet DataGrid (se 4.9.1) som är en ”Web Control” som betydligt underlättar tillvägagångssättet för att åstadkomma skapandet av en HTML-tabell. I Visual Studio går det till så att man skapar den grafiska komponenten på .aspx sidan genom drag and drop samt gör vissa inställningar för den. Sedan skapar man en funktion i codebehind-filen som läser data från databasen till ett DataSet-objekt. Därefter sätts källan för DataGrid-objektet (DataSource) till DataSet-objektet. Efter körning av DataBind() så visas innehållet i det data som lästes från början som en tabell på hemsidan. Se 4.9.1 för exempelkod. Vill man utföra mer avancerad visning så finns det inbyggda funktioner i DataGrid-objektet för ”paging” vilket innebär att man t.ex. kan visa 15 poster i databasen och därefter klicka på ”nästa” för att se de 15 nästa posterna. Ytterligare möjligheter finns för att lägga in t.ex. ändra/tabort-knappar för respektive post så att man lättare kan ändra eller ta bort denna. All denna ytterliggare funktionalitet får man programmera själv i ASP vilket tar mycket tid.

ASP: I ASP så får man själv programmera tabellen på det sätt man vill med blandning av skriptkod och html. Detta kan ta mycket tid och försvårar återanvändandet av kod samt ökar risken för buggar.

Ändringar av tabell i ASP respektive ASPX
Vill man t.ex. läsa in samtliga poster i en annan databastabell och visa den på samma sätt så är det enda man behöver göra i ASP.NET-applikationen att ändra sql-select-satsen så att den läser in en annan databastabell. Eftersom DataGrid läser ett DataSet och sedan automatiskt genererar en html-tabell vid rendreringen behövs inga ytterliggare ändringar.

I ASP-applikationen kan det däremot behövas betydligt större ändringar i koden för att åstadkomma samma sak. Dessa ändringar kan innebära, beroende hur man programmerat programmet, förutom ändringar i programkoden en hel del tillägg och ändringar i HTML-koden. I ASP.NET undviker man som regel att skriva HTML-kod direkt utan överlåter detta till rendreringsmotorn som anpassar koden till valbar webbläsare.

9.5 Webbadministration / Installation av systemen (Deployment)

Web administration och installation av system är betydligt enklare med .NET system då dessa endast behöver kopieras till servern. I fallet med ASP så krävs ofta flera manuella steg och diverse specialkonfigurering av systemet. Särskilt då COM-komponenter i olika dll-filer ska användas.

9.6 Utvecklingsmöjligheter (expanderbarhet)

På grund av de nya teknikerna, bland annat ”Web Forms” och ”Codebehind”, som tillhandahålls i ASP.NET så underlättas återanvändandet av kod vilket innebär en stor tidsbesparing.

Som visats i prestandatestet (se 9.3) så fås inte alltid någon automatisk prestandaökning vid övergång till ASP.NET. Obetänksamt användande av ”View State” kan få sidor att ladda extremt långsamt över långsamma internetförbindelser. Dock så finns det bättre förutsättningar i ASP.NET för att bygga program med högre prestanda än motsvarande program i traditionella ASP. Två huvudfaktorer är den kompilerade koden samt avancerade cachningsmöjligheter.

9.6.1 Skalbarhet

En av bristerna i ASP var avsaknaden av stöd för så kallade ”Web Farms”. I ASP.NET finns stöd för detta, bland annat genom att sessionsobjektet kan sparas i en databas. Denna möjlighet är för tillfället inte aktuell för den nya versionen av PreCom. Dock så kan det vara bra att vid utveckling tänka på att konstruera programmet så att inga hinder mot ”Web Farms” byggs in.

10 Slutsatser

10.1 Projektets första mål

Det första målet var att utreda hur en migrering av det gamla ASP-systemet PreCom Bud skulle kunna migreras till ASP.NET. Implementation 1 visade att migreringsstrategierna ”sida för sida” kombinerat med ”migrering allteftersom” samt horisontell migrering var möjliga realistiska alternativ. För fallet med horisontell migrering behövs ytterliggare testning för att avgöra om det inte tillför alltför många buggar. Annars verkade detta alternativ fungera väldigt bra. Om systemet är större än det som testats och väl utprövat från början rekommenderas kombinationen av strategierna ”sida för sida” och ”migrering allteftersom” då denna inte i onödan migrerar befintlig kod.

Det finns flera problem som kan uppstå vid migrering. Ett av de största är utbyte av variabler (vanligtvis så kallade sessionsvariabler). Problemet uppstår då variablerna ej är kompatibla med varandra i ASP och ASP.NET. Vid i stort sett statiska sessionsvariabler som i fallet med PreCom Bud behöver variablerna endast överföras till ASP.NET, via exempelvis POST metoden, efter inloggningen och sedan behåller de sitt värde. Ändras sessionsvariablerna dynamiskt under programmets gång så behövs andra lösningar som t.ex. programmen StateStich från Cansonica eller ASPBridge Include Control från Iprisma software. Andra lösningar då man bygger ett eget sessionsobjekt som lagrar informationen i en databas är också möjliga.

När det gäller kodmigrering (översättning från VBScript till VB.NET) av presentationslagret så visade sig det enda realistiska alternativet vara att utföra detta med hjälp av ett migreringsverktyg som t.ex. ASP2ASPX. Kodmigrering för hand var väldigt tidsödande och riskerade att introducera mängder med nya buggar (se 9.2.1).

Ingen slutlig migreringsstrategi för PreCom Bud valdes då dess funktionalitet genom omskrivning istället integrerades i den nya mer generella versionen av PreCom, vars prototyp utvecklats i implementation 2 (kapitel 8).

För att ge svar på frågan om prestanda vid migrering och i viss utsträckning även i övriga fall togs metoder fram för detta i kapitel 9.3. Metoden som valdes var att prestandatesta en sida som läste poster från en databas och presenterade dessa i en HTML-tabell på en webbsida (.asp-sida för ASP respektive .aspx-sida för ASP.NET). Denna sida prestandatestades sedan med Microsoft Application Center Test (se 9.3.1.1) med olika parametrar. Resultatet från detta test visade att ASP och ASP.NET hade likvärdiga prestanda. Vid användning av mellanlager försämrades prestanda med ca 40% och en ytterliggare prestandaförsämring tillkom vid migrering. Är webbservern lätt belastad så kommer dock denna försämring knappt att märkas av användaren. Vid en hårt belastad server kan däremot prestandaförsämringen vara tillräckligt stor för att vägas in då migrering övervägs. För att få en mer heltäckande bild av hur prestanda kommer att påverkas för det egna systemet så behöver givetvis flera tester utföras.

Mer allmänt kan sägas att migrering hela tiden handlar om att väga fördelar mot nackdelar. T.ex. vad den vanliga användaren vinner på en migrering till .NET, i förhållande till vad utvecklarna vinner på en migrering. Det gäller att försöka minimera mängden arbete för utvecklaren och samtidigt helst förbättra funktionalitet, användarvänlighet och prestanda för användaren, dels för migreringen samt den fortsatta utvecklingen av systemet.

10.2 Projektets andra mål

Det andra målet var att ta fram en prototyp för hantering av flera olika ordertyper. Detta resulterade i webbsystemet som beskrivs i Implementation 2 i kapitel 8. Systemet kan sägas vara en omskrivning samt utökning av PreCom Bud som beskrivs i Implementation 1 i kapitel 7. Vid en total omskrivning av ett system kan man till fullo dra nytta av samtliga fördelar med ASP.NET. Dock är det i många fall inte rimligt att göra en total omskrivning, på grund av den stora mängden arbetstimmar som måste läggas ned.

Vid arbetet med Implementation 2 designades en databas och relationer mellan databastabellerna skapades. Ett användargränssnitt skapades med hjälp av ”Web Forms” och ”Web Controls” i Visual Studio .NET 2003. Dessutom krävdes programmering i C# för att skapa ”codebehind-filerna” för de olika webbformulären. C#-koden binder bland annat ihop ”Web Controls” med databasen.

Dessutom utvecklades en positioneringsmodul, passande även andra av PocketMobiles applikationer. Med hjälp av denna kan ett antal fordons positioner visas på en karta. Andra funktioner som implementerades är bland annat sökning på gata eller stadsdel. Kommunikation med postitioneringsmodulen sker med ”QueryString”. Positioneringsmodulen skapar kartan med hjälp av instruktioner som skickas till företaget Cartesias kart-API, Easy-Mapper, som genererar kartan i form av en .gif-bild.

Det som återstår innan systemet kan nyttjas kommersiellt är en del implementeringsarbete, för t.ex. funktioner som avräkning (sammanställning av kostnader och inkomster) och inställningar, samt testning.

11 Referenser

1. Ecma-334. C# Language Specification. 2nd edition (December 2002). <http://www.ecma-international.org/publications/files/ecma-st/Ecma-334.pdf>

2. Ecma-335. Common Language Infrastructure (CLI). 2nd edition (December 2002). <http://www.ecma-international.org/publications/files/ecma-st/Ecma-335.pdf>

3. Microsoft .NET/COM Migration and Interoperability. Steve Busby and Edward Jezierksi. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cominterop.asp>

4. The Microsoft Shared Source CLI Implementation. David Stutz. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Dndotnet/html/mssharsourcecli.asp>

5. Moving to ASP.NET – White Pages. Utgivare Consonica, besökt 2003-09-03. <http://www.consonica.com/information/movingtoaspdotnet/movingtoaspdotnet.pdf>

6. ASPBridge IncludeControl. IPrisma Software. Besökt 2003-12-01. <http://www.iprisma.com/aspbridge/>

7. How to Share Session State Between Classic ASP and ASP.NET. Billy Yuen. February 2003 <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/ConvertToASPNET.asp>

8. Com Interoperability Tutorial. Besökt 2003-10-22. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmex/html/vcgrfintroductionnetremoting.asp>

9. Make your .NET apps interoperable with existing COM assets. Tim Landgrave. 2003-05-07 <http://builder.com.com/5102-6387-5034862.html>

10. ”User’s Guide” och “Language Reference” för “Visual Basic Scripting Edition”. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vtorivbscript.asp>

11. ASP From A to Z. Nancy Winnick Cluts. 1998-10-22. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnasp/html/aspatoz.asp>

12. Transfer Session Variables from Classic ASP to ASP.NET. Peter A. Bromberg, Ph.D. Besökt 2003-11-11. <http://www.eggheadcafe.com/articles/20021207.asp>

13. RFC-2616. Hypertext Transfer Protocol -- HTTP/1.1. Juni 1999. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

14. Web Services Interoperability Organisation.
<http://www.ws-i.org/>

15. Web Services Architecture. W3C Working Draft 8 August 2003.
<http://www.w3.org/TR/ws-arch/>

16. Hjälpfil till Microsoft Application Center Test version 1.0.536.0

17. Type-Safety in .NET. Don Kiely. 2001-05-11.
<http://www.itworld.com/nl/nt2k_sec/11052001/>

18. SIE-Gruppen. Besökt 2004-04-27.
<http://www.sie.se>

19. Easy-Mapper API (E-MAPI) Reference Guide and Specifications. Anders Haraldsson. 2001-03-22 (Revision 1.22).

20. Mono. A free implementation of the .NET Development Framework.
<http://www.go-mono.com/>

12 Appendix: A Förkortningar

ACT	(Microsoft) Application Center Test
ADO	ActiveX Data Object
API	Application Programming Interface
ASP	Active Server Pages
CCW	COM Callable Wrapper
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
COM	Component Object Model
CTS	Common Type System
EE	Execution Engine
E-MAPI	Easy-Mapper API
GAC	Global Assembly Cache
GC	Garbage Collector/Collection
GUI	Graphical User Interface
GUID	Globally Unique Identifier
IDE	Integrated Development Environment
IDL	Interface Definition Language
IIS	Internet Information Services
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
MSDN	Microsoft Developer Network
MSIL	Microsoft Immediate Language
PE	(Common Language Runtime) Portable Executable
PIA	Primary Interop Assembly
QS	Query String
RCW	Runtime Callable Wrapper
RFC	Requst For Comments
RPS	Requests Per Second
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
VB.NET	Visual Basic .NET
VS.NET	Visual Studio .NET
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WS-I	Web Services Interoperability Organization
XML	eXtensible Markup Language

13 Appendix: B Trace

Följande system har använts.
Dator: Pentium III 500Mhz, 256 MB RAM.
Programvara: Windows 2000 SP3, MS SQL-server 7.0, ASP.NET version 1.1 och MS Internet Explorer 6.0.

13.1 ”Tracelogg” för NewOrder.aspx

”Tracelogg” vid exekvering av NewOrder.aspx (se Figur 11, sid 37).

13.1.1 Request Details

Session Id:	55t2sn55iblqus55gmzc5n45	**Request Type:**	POST
Time of Request:	2004-01-14 11:14:22	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

13.1.2 Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin Init		
aspx.page	End Init	0,000329	0,000329
aspx.page	Begin LoadViewState	0,000440	0,000111
aspx.page	End LoadViewState	0,033848	0,033408
aspx.page	Begin ProcessPostData	0,034012	0,000164
aspx.page	End ProcessPostData	0,038538	0,004526
aspx.page	Begin ProcessPostData Second Try	0,038844	0,000306
aspx.page	End ProcessPostData Second Try	0,038944	0,000100
aspx.page	Begin Raise ChangedEvents	0,039036	0,000092
aspx.page	End Raise ChangedEvents	0,069200	0,030164
aspx.page	Begin Raise PostBackEvent	0,069332	0,000132
aspx.page	End Raise PostBackEvent	0,069445	0,000113
aspx.page	Begin PreRender	0,069536	0,000091
aspx.page	End PreRender	0,070029	0,000493
aspx.page	Begin SaveViewState	0,103835	0,033806
aspx.page	End SaveViewState	0,105177	0,001342
aspx.page	Begin Render	0,105284	0,000107
aspx.page	End Render	0,144818	0,039534

13.1.3 Control Tree

Control Id	Type	Render Size Bytes (including children)	Viewstate Size Bytes (excluding children)
__PAGE	ASP.NewOrder_aspx	41216	20
_ctl0	System.Web.UI.ResourceBasedLiteralControl	528	0
NewOrder	System.Web.UI.HtmlControls.HtmlForm	40667	0
_ctl1	System.Web.UI.LiteralControl	5	0
CustomerInfoLabel	System.Web.UI.WebControls.Label	191	0
TextBox22	System.Web.UI.WebControls.TextBox	0	48
Label35	System.Web.UI.WebControls.Label	184	0
RemoteOrderIDTextbox	System.Web.UI.WebControls.TextBox	0	0
ChangeOrderButton	System.Web.UI.WebControls.Button	0	0
DropTimeTextbox	System.Web.UI.WebControls.TextBox	0	36
Label54	System.Web.UI.WebControls.Label	0	0
GetTimeTextbox	System.Web.UI.WebControls.TextBox	0	36
Label53	System.Web.UI.WebControls.Label	0	0
Label52	System.Web.UI.WebControls.Label	0	0
Label51	System.Web.UI.WebControls.Label	0	0
Label50	System.Web.UI.WebControls.Label	0	0
DropDayTextbox	System.Web.UI.WebControls.TextBox	0	32
DropMonthTextbox	System.Web.UI.WebControls.TextBox	0	32
GetDayTextbox	System.Web.UI.WebControls.TextBox	0	32
GetMonthTextbox	System.Web.UI.WebControls.TextBox	0	32
DropYearTextbox	System.Web.UI.WebControls.TextBox	0	32
GetYearTextbox	System.Web.UI.WebControls.TextBox	0	32
Label49	System.Web.UI.WebControls.Label	148	0
Label48	System.Web.UI.WebControls.Label	0	0
Label47	System.Web.UI.WebControls.Label	0	0

CustomerIDTextbox	System.Web.UI.WebControls.TextBox	285	28
Label46	System.Web.UI.WebControls.Label	154	0
WeekTextbox	System.Web.UI.WebControls.TextBox	224	28
Label43	System.Web.UI.WebControls.Label	148	0
Label45	System.Web.UI.WebControls.Label	148	0
Label44	System.Web.UI.WebControls.Label	0	0
OrderYearTextbox	System.Web.UI.WebControls.TextBox	286	32
OrderMonthTextbox	System.Web.UI.WebControls.TextBox	287	32
OrderDayTextbox	System.Web.UI.WebControls.TextBox	281	32
HiddenOrderDayTextbox	System.Web.UI.WebControls.TextBox	0	28
OrderDayDropdownlist	System.Web.UI.WebControls.DropDownList	1417	736
Label31	System.Web.UI.WebControls.Label	158	0
LineDropdownlist	System.Web.UI.WebControls.DropDownList	387	180
HiddenUnitTextBox	System.Web.UI.WebControls.TextBox	0	32
HiddenAmountTextBox	System.Web.UI.WebControls.TextBox	0	28
HiddenPriceTextBox	System.Web.UI.WebControls.TextBox	0	32
NewButton1	System.Web.UI.WebControls.Button	205	0
Label41	System.Web.UI.WebControls.Label	0	0
VehicleDropDownList	System.Web.UI.WebControls.DropDownList	259	132
SubContractorDropDownList	System.Web.UI.WebControls.DropDownList	408	196
TextBox24	System.Web.UI.WebControls.TextBox	210	32
Label40	System.Web.UI.WebControls.Label	159	0
TextBox21	System.Web.UI.WebControls.TextBox	252	0
Label39	System.Web.UI.WebControls.Label	153	0
TextBox4	System.Web.UI.WebControls.TextBox	222	36
TextBox3	System.Web.UI.WebControls.TextBox	221	32
Label38	System.Web.UI.WebControls.Label	154	0
TextBox2	System.Web.UI.WebControls.TextBox	208	0
TextBox1	System.Web.UI.WebControls.TextBox	208	0
Label37	System.Web.UI.WebControls.Label	152	0
Label34	System.Web.UI.WebControls.Label	153	0
Label29	System.Web.UI.WebControls.Label	184	0
Label14	System.Web.UI.WebControls.Label	191	0
Label13	System.Web.UI.WebControls.Label	154	0
Label11	System.Web.UI.WebControls.Label	154	0
Label10	System.Web.UI.WebControls.Label	155	0
DropDownList3	System.Web.UI.WebControls.DropDownList	526	316
Label9	System.Web.UI.WebControls.Label	158	0
DropDownList2	System.Web.UI.WebControls.DropDownList	526	324
EarlierAvrakningarLabel	System.Web.UI.WebControls.Label	199	0
OrdersDataGrid	System.Web.UI.WebControls.DataGrid	3639	1684
OrdersDataGrid:_ctl0	System.Web.UI.WebControls.DataGridTable	3639	0
OrdersDataGrid:_ctl1	System.Web.UI.WebControls.DataGridItem	0	0
OrdersDataGrid:_ctl1:_ctl2	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl1:_ctl0	System.Web.UI.WebControls.Label	0	0
OrdersDataGrid:_ctl1:_ctl3	System.Web.UI.LiteralControl	0	0
OrdersDataGrid:_ctl1:_ctl1	System.Web.UI.WebControls.DataGridLinkButton	0	0
OrdersDataGrid:_ctl2	System.Web.UI.WebControls.DataGridItem	327	0
OrdersDataGrid:_ctl2:_ctl0	System.Web.UI.WebControls.TableCell	17	0
OrdersDataGrid:_ctl2:_ctl1	System.Web.UI.WebControls.TableCell	11	0
OrdersDataGrid:_ctl2:_ctl2	System.Web.UI.WebControls.TableCell	13	0
OrdersDataGrid:_ctl2:_ctl3	System.Web.UI.WebControls.TableCell	17	0
OrdersDataGrid:_ctl2:_ctl4	System.Web.UI.WebControls.TableCell	15	0
OrdersDataGrid:_ctl2:_ctl5	System.Web.UI.WebControls.TableCell	15	0
OrdersDataGrid:_ctl2:_ctl6	System.Web.UI.WebControls.TableCell	18	0
OrdersDataGrid:_ctl2:_ctl7	System.Web.UI.WebControls.TableCell	17	0
OrdersDataGrid:_ctl2:_ctl8	System.Web.UI.WebControls.TableCell	14	0
OrdersDataGrid:_ctl2:_ctl9	System.Web.UI.WebControls.TableCell	20	0
OrdersDataGrid:_ctl2:_ctl10	System.Web.UI.WebControls.TableCell	16	0
OrdersDataGrid:_ctl2:_ctl11	System.Web.UI.WebControls.TableCell	17	0
OrdersDataGrid:_ctl2:_ctl12	System.Web.UI.WebControls.TableCell	19	0
OrdersDataGrid:_ctl3	System.Web.UI.WebControls.DataGridItem	481	0
OrdersDataGrid:_ctl3:_ctl1	System.Web.UI.WebControls.TableCell	83	0
OrdersDataGrid:_ctl3:_ctl0	System.Web.UI.WebControls.Button	72	0
OrdersDataGrid:_ctl3:_ctl2	System.Web.UI.WebControls.TableCell	11	32
OrdersDataGrid:_ctl3:_ctl3	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl3:_ctl4	System.Web.UI.WebControls.TableCell	59	96
OrdersDataGrid:_ctl3:_ctl5	System.Web.UI.WebControls.TableCell	16	36
OrdersDataGrid:_ctl3:_ctl6	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl3:_ctl7	System.Web.UI.WebControls.TableCell	20	44
OrdersDataGrid:_ctl3:_ctl8	System.Web.UI.WebControls.TableCell	30	56
OrdersDataGrid:_ctl3:_ctl9	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl3:_ctl10	System.Web.UI.WebControls.TableCell	27	52
OrdersDataGrid:_ctl3:_ctl11	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl3:_ctl12	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl3:_ctl13	System.Web.UI.WebControls.TableCell	28	52

OrdersDataGrid:_ctl4	System.Web.UI.WebControls.DataGridItem	485	0
OrdersDataGrid:_ctl4:_ctl1	System.Web.UI.WebControls.TableCell	83	0
OrdersDataGrid:_ctl4:_ctl0	System.Web.UI.WebControls.Button	72	0
OrdersDataGrid:_ctl4:_ctl2	System.Web.UI.WebControls.TableCell	11	32
OrdersDataGrid:_ctl4:_ctl3	System.Web.UI.WebControls.TableCell	25	48
OrdersDataGrid:_ctl4:_ctl4	System.Web.UI.WebControls.TableCell	59	96
OrdersDataGrid:_ctl4:_ctl5	System.Web.UI.WebControls.TableCell	16	36
OrdersDataGrid:_ctl4:_ctl6	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl4:_ctl7	System.Web.UI.WebControls.TableCell	24	48
OrdersDataGrid:_ctl4:_ctl8	System.Web.UI.WebControls.TableCell	30	56
OrdersDataGrid:_ctl4:_ctl9	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl4:_ctl10	System.Web.UI.WebControls.TableCell	27	52
OrdersDataGrid:_ctl4:_ctl11	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl4:_ctl12	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl4:_ctl13	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl5	System.Web.UI.WebControls.DataGridItem	481	0
OrdersDataGrid:_ctl5:_ctl1	System.Web.UI.WebControls.TableCell	83	0
OrdersDataGrid:_ctl5:_ctl0	System.Web.UI.WebControls.Button	72	0
OrdersDataGrid:_ctl5:_ctl2	System.Web.UI.WebControls.TableCell	11	32
OrdersDataGrid:_ctl5:_ctl3	System.Web.UI.WebControls.TableCell	25	48
OrdersDataGrid:_ctl5:_ctl4	System.Web.UI.WebControls.TableCell	59	96
OrdersDataGrid:_ctl5:_ctl5	System.Web.UI.WebControls.TableCell	16	36
OrdersDataGrid:_ctl5:_ctl6	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl5:_ctl7	System.Web.UI.WebControls.TableCell	24	48
OrdersDataGrid:_ctl5:_ctl8	System.Web.UI.WebControls.TableCell	30	56
OrdersDataGrid:_ctl5:_ctl9	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl5:_ctl10	System.Web.UI.WebControls.TableCell	27	52
OrdersDataGrid:_ctl5:_ctl11	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl5:_ctl12	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl5:_ctl13	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl6	System.Web.UI.WebControls.DataGridItem	485	0
OrdersDataGrid:_ctl6:_ctl1	System.Web.UI.WebControls.TableCell	83	0
OrdersDataGrid:_ctl6:_ctl0	System.Web.UI.WebControls.Button	72	0
OrdersDataGrid:_ctl6:_ctl2	System.Web.UI.WebControls.TableCell	11	32
OrdersDataGrid:_ctl6:_ctl3	System.Web.UI.WebControls.TableCell	25	48
OrdersDataGrid:_ctl6:_ctl4	System.Web.UI.WebControls.TableCell	59	96
OrdersDataGrid:_ctl6:_ctl5	System.Web.UI.WebControls.TableCell	16	36
OrdersDataGrid:_ctl6:_ctl6	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl6:_ctl7	System.Web.UI.WebControls.TableCell	24	48
OrdersDataGrid:_ctl6:_ctl8	System.Web.UI.WebControls.TableCell	30	56
OrdersDataGrid:_ctl6:_ctl9	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl6:_ctl10	System.Web.UI.WebControls.TableCell	27	52
OrdersDataGrid:_ctl6:_ctl11	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl6:_ctl12	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl6:_ctl13	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl7	System.Web.UI.WebControls.DataGridItem	472	0
OrdersDataGrid:_ctl7:_ctl1	System.Web.UI.WebControls.TableCell	83	0
OrdersDataGrid:_ctl7:_ctl0	System.Web.UI.WebControls.Button	72	0
OrdersDataGrid:_ctl7:_ctl2	System.Web.UI.WebControls.TableCell	11	32
OrdersDataGrid:_ctl7:_ctl3	System.Web.UI.WebControls.TableCell	25	48
OrdersDataGrid:_ctl7:_ctl4	System.Web.UI.WebControls.TableCell	59	96
OrdersDataGrid:_ctl7:_ctl5	System.Web.UI.WebControls.TableCell	16	36
OrdersDataGrid:_ctl7:_ctl6	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl7:_ctl7	System.Web.UI.WebControls.TableCell	24	48
OrdersDataGrid:_ctl7:_ctl8	System.Web.UI.WebControls.TableCell	21	44
OrdersDataGrid:_ctl7:_ctl9	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl7:_ctl10	System.Web.UI.WebControls.TableCell	27	52
OrdersDataGrid:_ctl7:_ctl11	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl7:_ctl12	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl7:_ctl13	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl8	System.Web.UI.WebControls.DataGridItem	476	0
OrdersDataGrid:_ctl8:_ctl1	System.Web.UI.WebControls.TableCell	83	0
OrdersDataGrid:_ctl8:_ctl0	System.Web.UI.WebControls.Button	72	0
OrdersDataGrid:_ctl8:_ctl2	System.Web.UI.WebControls.TableCell	11	32
OrdersDataGrid:_ctl8:_ctl3	System.Web.UI.WebControls.TableCell	25	48
OrdersDataGrid:_ctl8:_ctl4	System.Web.UI.WebControls.TableCell	59	96
OrdersDataGrid:_ctl8:_ctl5	System.Web.UI.WebControls.TableCell	16	36
OrdersDataGrid:_ctl8:_ctl6	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl8:_ctl7	System.Web.UI.WebControls.TableCell	24	48
OrdersDataGrid:_ctl8:_ctl8	System.Web.UI.WebControls.TableCell	21	44
OrdersDataGrid:_ctl8:_ctl9	System.Web.UI.WebControls.TableCell	18	40
OrdersDataGrid:_ctl8:_ctl10	System.Web.UI.WebControls.TableCell	27	52
OrdersDataGrid:_ctl8:_ctl11	System.Web.UI.WebControls.TableCell	29	56
OrdersDataGrid:_ctl8:_ctl12	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl8:_ctl13	System.Web.UI.WebControls.TableCell	28	52
OrdersDataGrid:_ctl9	System.Web.UI.WebControls.DataGridItem	0	0

OrdersDataGrid:_ctl9:_ctl0	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl1	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl2	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl3	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl4	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl5	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl6	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl7	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl8	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl9	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl10	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl11	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl9:_ctl12	System.Web.UI.WebControls.TableCell	0	0
OrdersDataGrid:_ctl10	System.Web.UI.WebControls.DataGridItem	178	0
OrdersDataGrid:_ctl10:_ctl2	System.Web.UI.WebControls.TableCell	123	0
OrdersDataGrid:_ctl10:_ctl0	System.Web.UI.WebControls.Label	17	0
OrdersDataGrid:_ctl10:_ctl3	System.Web.UI.LiteralControl	6	0
OrdersDataGrid:_ctl10:_ctl1	System.Web.UI.WebControls.DataGridLinkButton	76	0
CancelOrderButton	System.Web.UI.WebControls.Button	243	0
OrderButton	System.Web.UI.WebControls.Button	215	0
NotesLabel	System.Web.UI.WebControls.Label	163	0
NotesTextBox	System.Web.UI.WebControls.TextBox	149	0
RaderaButton3	System.Web.UI.WebControls.Button	215	0
Label36	System.Web.UI.WebControls.Label	181	0
Label12	System.Web.UI.WebControls.Label	154	0
Label18	System.Web.UI.WebControls.Label	152	0
TextBox9	System.Web.UI.WebControls.TextBox	228	44
TextBox10	System.Web.UI.WebControls.TextBox	224	36
Label19	System.Web.UI.WebControls.Label	158	0
TextBox11	System.Web.UI.WebControls.TextBox	225	36
Label20	System.Web.UI.WebControls.Label	153	0
TextBox12	System.Web.UI.WebControls.TextBox	227	36
Label21	System.Web.UI.WebControls.Label	156	0
TextBox13	System.Web.UI.WebControls.TextBox	232	44
Label22	System.Web.UI.WebControls.Label	152	0
Label23	System.Web.UI.WebControls.Label	154	0
TextBox14	System.Web.UI.WebControls.TextBox	232	44
TextBox15	System.Web.UI.WebControls.TextBox	237	52
TextBox23	System.Web.UI.WebControls.TextBox	0	44
SaveCustomerButton3	System.Web.UI.WebControls.Button	246	0
NewButton2	System.Web.UI.WebControls.Button	205	0
UppdateraButton3	System.Web.UI.WebControls.Button	244	0
RedigeraButton3	System.Web.UI.WebControls.Button	221	0
Label28	System.Web.UI.WebControls.Label	154	0
TextBox20	System.Web.UI.WebControls.TextBox	224	36
Label27	System.Web.UI.WebControls.Label	151	0
TextBox19	System.Web.UI.WebControls.TextBox	224	36
TextBox18	System.Web.UI.WebControls.TextBox	224	36
Label26	System.Web.UI.WebControls.Label	151	0
TextBox17	System.Web.UI.WebControls.TextBox	224	36
Label25	System.Web.UI.WebControls.Label	155	0
Label24	System.Web.UI.WebControls.Label	156	0
TextBox16	System.Web.UI.WebControls.TextBox	226	36
DropDownList1	System.Web.UI.WebControls.DropDownList	394	232
RaderaButton2	System.Web.UI.WebControls.Button	215	0
Label33	System.Web.UI.WebControls.Label	151	0
Label32	System.Web.UI.WebControls.Label	151	0
Label8	System.Web.UI.WebControls.Label	153	0
Label7	System.Web.UI.WebControls.Label	154	0
Label5	System.Web.UI.WebControls.Label	157	0
Label4	System.Web.UI.WebControls.Label	155	0
Label3	System.Web.UI.WebControls.Label	152	0
Label1	System.Web.UI.WebControls.Label	177	0
Label2	System.Web.UI.WebControls.Label	151	0
Label6	System.Web.UI.WebControls.Label	153	0
GetAddressZipTextBox2	System.Web.UI.WebControls.TextBox	247	36
GetAddressCountryTextBox2	System.Web.UI.WebControls.TextBox	258	36
GetAddressPhoneTextBox2	System.Web.UI.WebControls.TextBox	260	44
GetAddressCodeTextBox2	System.Web.UI.WebControls.TextBox	249	32
GetAddressOpenTextBox3	System.Web.UI.WebControls.TextBox	250	36
GetAddressOpenTextBox4	System.Web.UI.WebControls.TextBox	250	36
GetAddressLunchTextBox3	System.Web.UI.WebControls.TextBox	252	36
GetAddressLunchTextBox4	System.Web.UI.WebControls.TextBox	252	36
GetAddressDropDownList2	System.Web.UI.WebControls.DropDownList	434	220
TextBox5	System.Web.UI.WebControls.TextBox	230	48
Label15	System.Web.UI.WebControls.Label	154	0

TextBox6	System.Web.UI.WebControls.TextBox	223	40
TextBox7	System.Web.UI.WebControls.TextBox	229	44
Label16	System.Web.UI.WebControls.Label	152	0
Label17	System.Web.UI.WebControls.Label	156	0
TextBox8	System.Web.UI.WebControls.TextBox	233	48
Label30	System.Web.UI.WebControls.Label	0	0
SaveCustomerButton2	System.Web.UI.WebControls.Button	246	0
UppdateraButton2	System.Web.UI.WebControls.Button	244	0
RedigeraButton2	System.Web.UI.WebControls.Button	221	0
ReferenceLabel	System.Web.UI.WebControls.Label	162	0
ReferenceDropDownList	System.Web.UI.WebControls.DropDownList	522	356
_ctl2	System.Web.UI.LiteralControl	11	0
CompanyDropDownList	System.Web.UI.WebControls.DropDownList	421	232
CompanyLabel	System.Web.UI.WebControls.Label	156	0
_ctl3	System.Web.UI.LiteralControl	101	0
Debug	System.Web.UI.WebControls.TextBox	0	36
_ctl4	System.Web.UI.LiteralControl	21	0

13.1.4 Session state

Session Key	Type	Value
TestVar	System.String	Detta är ett test!

13.1.5 Cookies Collection

Name	Value	Size
.ASPXFORMSAUTH	2D47994B783288D1400D388BDE51ED29462DA0BC4E0027690CEDFAE457C0ECF728F5AD45EB261F0EB62A1F1943B39B11D796C7BD2DB72151D5F5C6A3BF8A3D99F3F9476EEF56A7C3196C23D76DAD980E	175
ASP.NET_SessionId	55t2sn55iblqus55gmzc5n45	42

13.1.6 Headers Collection

Name	Value
Cache-Control	no-cache
Connection	Keep-Alive
Content-Length	13211
Content-Type	application/x-www-form-urlencoded
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Accept-Encoding	gzip, deflate
Accept-Language	en-gb,sv;q=0.5
Cookie	.ASPXFORMSAUTH=2D47994B783288D1400D388BDE51ED29462DA0BC4E0027690CEDFAE457C0ECF728F5AD45EB261F0EB62A1F1943B39B11D796C7BD2DB72151D5F5C6A3BF8A3D99F3F9476EEF56A7C3196C23D76DAD980E; ASP.NET_SessionId=55t2sn55iblqus55gmzc5n45
Host	localhost:4711
Referer	http://localhost:4711/PreComBusinessLogic/NewOrder.aspx
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; FunWebProducts-MyWay; .NET CLR 1.0.3705; .NET CLR 1.1.4322)

13.1.7 Form Collection

Name	Value
__EVENTTARGET	SubContractorDropDownList
__EVENTARGUMENT	
__VIEWSTATE	dDwtNjc5NjkzOTE5O3Q8cDxsPFBlbGxlO05pc3NlOz47bDx0am9obztwbGluZ3Bsb25nOz4+O2w8aTwxPjs+O2w8dDw7bDxpP * Resten är borttaget p.g.a. platsbrist. Totalt ca 40 kB. *
CustomerIDTextbox	1
OrderYearTextbox	2004
OrderMonthTextbox	01
OrderDayTextbox	14
OrderDayDropdownlist	Onsdag 14/1
LineDropdownlist	Sthlm-Gbg
VehicleDropDownList	BFG 332
SubContractorDropDownList	Företag2
TextBox21	
DropDownList3	Bärlagergrus 0-30mm
DropDownList2	Anderssons Grus AB
NotesTextBox	
DropDownList1	Leverera mig
GetAddressDropDownList2	Hämtställe 22
ReferenceDropDownList	Anna Persson
CompanyDropDownList	Byggkoncernen AB

13.1.8 Server Variables

Name	Value
ALL_HTTP	HTTP_CACHE_CONTROL:no-cache HTTP_CONNECTION:Keep-Alive HTTP_CONTENT_LENGTH:13211 HTTP_CONTENT_TYPE:application/x-www-form-urlencoded HTTP_ACCEPT:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */* HTTP_ACCEPT_ENCODING:gzip, deflate HTTP_ACCEPT_LANGUAGE:en-gb,sv;q=0.5 HTTP_COOKIE:.ASPXFORMSAUTH=2D47994B783288D1400D388BDE51ED29462DA0BC4E0027690CEDFAE457C0ECF728F5AD45EB261F0EB62A1F1943B39B11D796C7BD2DB72151D5F5C6A3BF8A3D99F3F9476EEF56A7C3196C23D76DAD980E; ASP.NET_SessionId=55t2sn55iblqus55gmzc5n45 HTTP_HOST:localhost:4711 HTTP_REFERER:http://localhost:4711/PreComBusinessLogic/NewOrder.aspx HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; FunWebProducts-MyWay; .NET CLR 1.0.3705; .NET CLR 1.1.4322)
ALL_RAW	Cache-Control: no-cache Connection: Keep-Alive Content-Length: 13211 Content-Type: application/x-www-form-urlencoded Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */* Accept-Encoding: gzip, deflate Accept-Language: en-gb,sv;q=0.5 Cookie: .ASPXFORMSAUTH=2D47994B783288D1400D388BDE51ED29462DA0BC4E0027690CEDFAE457C0ECF728F5AD45EB261F0EB62A1F1943B39B11D796C7BD2DB72151D5F5C6A3BF8A3D99F3F9476EEF56A7C3196C23D76DAD980E; ASP.NET_SessionId=55t2sn55iblqus55gmzc5n45 Host: localhost:4711 Referer: http://localhost:4711/PreComBusinessLogic/NewOrder.aspx User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; FunWebProducts-MyWay; .NET CLR 1.0.3705; .NET CLR 1.1.4322)
APPL_MD_PATH	/LM/w3svc/1/root/PreComBusinessLogic
APPL_PHYSICAL_PATH	f:\inetpub\wwwroot\PreComBusinessLogic\
AUTH_TYPE	
AUTH_USER	
AUTH_PASSWORD	
LOGON_USER	
REMOTE_USER	
CERT_COOKIE	
CERT_FLAGS	
CERT_ISSUER	
CERT_KEYSIZE	
CERT_SECRETKEYSIZE	
CERT_SERIALNUMBER	
CERT_SERVER_ISSUER	
CERT_SERVER_SUBJECT	
CERT_SUBJECT	
CONTENT_LENGTH	13211
CONTENT_TYPE	application/x-www-form-urlencoded
GATEWAY_INTERFACE	CGI/1.1
HTTPS	off
HTTPS_KEYSIZE	
HTTPS_SECRETKEYSIZE	
HTTPS_SERVER_ISSUER	
HTTPS_SERVER_SUBJECT	
INSTANCE_ID	1
INSTANCE_META_PATH	/LM/W3SVC/1
LOCAL_ADDR	127.0.0.1
PATH_INFO	/PreComBusinessLogic/NewOrder.aspx
PATH_TRANSLATED	f:\inetpub\wwwroot\PreComBusinessLogic\NewOrder.aspx
QUERY_STRING	
REMOTE_ADDR	127.0.0.1
REMOTE_HOST	127.0.0.1
REMOTE_PORT	1114
REQUEST_METHOD	POST
SCRIPT_NAME	/PreComBusinessLogic/NewOrder.aspx
SERVER_NAME	localhost
SERVER_PORT	4711
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.1
SERVER_SOFTWARE	Microsoft-IIS/5.0
URL	/PreComBusinessLogic/NewOrder.aspx
HTTP_CACHE_CONTROL	no-cache
HTTP_CONNECTION	Keep-Alive
HTTP_CONTENT_LENGTH	13211
HTTP_CONTENT_TYPE	application/x-www-form-urlencoded
HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	en-gb,sv;q=0.5
HTTP_COOKIE	.ASPXFORMSAUTH=2D47994B783288D1400D388BDE51ED29462DA0BC4E0027690CEDFAE457

	C0ECF728F5AD45EB261F0EB62A1F1943B39B11D796C7BD2DB72151D5F5C6A3BF8A3D99F3F9476EEF56A7C3196C23D76DAD980E; ASP.NET_SessionId=55t2sn55iblqus55gmzc5n45
HTTP_HOST	localhost:4711
HTTP_REFERER	http://localhost:4711/PreComBusinessLogic/NewOrder.aspx
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; FunWebProducts-MyWay; .NET CLR 1.0.3705; .NET CLR 1.1.4322)