Datavetenskap

Fredrik Fahlstad och Michael Kihlén

Utvärdering av alternativ programutvecklingsmiljö

En jämförelse mellan Microsoft Visual Studio 6.0 och Visual Studio .NET med avseende på databasstöd.

Examensarbete, C-nivå

2003:16

Utvärdering av alternativ programutvecklingsmiljö

En jämförelse mellan Microsoft Visual Studio 6.0 och Visual Studio .NET med avseende på databasstöd.

Fredrik Fahlstad och Michael Kihlén

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Fredrik Fahlstad och Michael Kihlén

Godkänd, 2003-06-03

Handledare: Per Strömgren

Examinator: Stefan Lindskog

Sammanfattning

Den här tekniska rapporten har utarbetats på uppdrag av Midroc Electro, i syfte att utvärdera Visual Studio Dotnet (VS Dotnet). Bakgrunden till utvärderingen är att Midroc Electro upplever programutvecklingen i nuvarande utvecklingsmiljö Visual Studio 6 (VS 6) som tidskrävande och komplex. Utvärderingen har utförts genom att implementera en alternativ lösning i VS Dotnet av en befintlig referensapplikation utvecklad i VS 6. Dotnet erbjuder flera nya arbetssätt som underlättar och effektiviserar programutvecklingen, speciellt mot databaser som den här utvärderingen främst berör. Resultatet av utvärderingen visar att VS Dotnet underlättar utveckling av flerskiktade system, men VS Dotnet innehåller en del barnsjukdomar och programfel som gör att produkten inte känns redo för att utveckla stora system i, utan Midroc Electro bör invänta en uppdatering.

Evaluation of alternative software development environment

Comparison between Microsoft Visual Studio 6.0 and Visual Studio .NET with regard to database support.

Abstract

This technical document is based on a request from Midroc Electro, in order to evaluate Visual Studio Dotnet (VS Dotnet). The reason for this evaluation is that Midroc Electro feels that program development in their present development environment Visual Studio 6 (VS 6) are time consuming and complex. To perform the evaluation, a Dotnet implementation is made of an existing reference project developed in VS 6. Dotnet offers several new ways that makes program development easier and more efficient especially when working with databases, which this evaluation mainly is about. The result of this evaluation shows that Dotnet simplifies development of multi-layered solutions. But VS Dotnet still contains several program errors making Dotnet not yet suitable for large system development. Midroc Electro is recommended to wait for the next improved release.

Innehållsförteckning

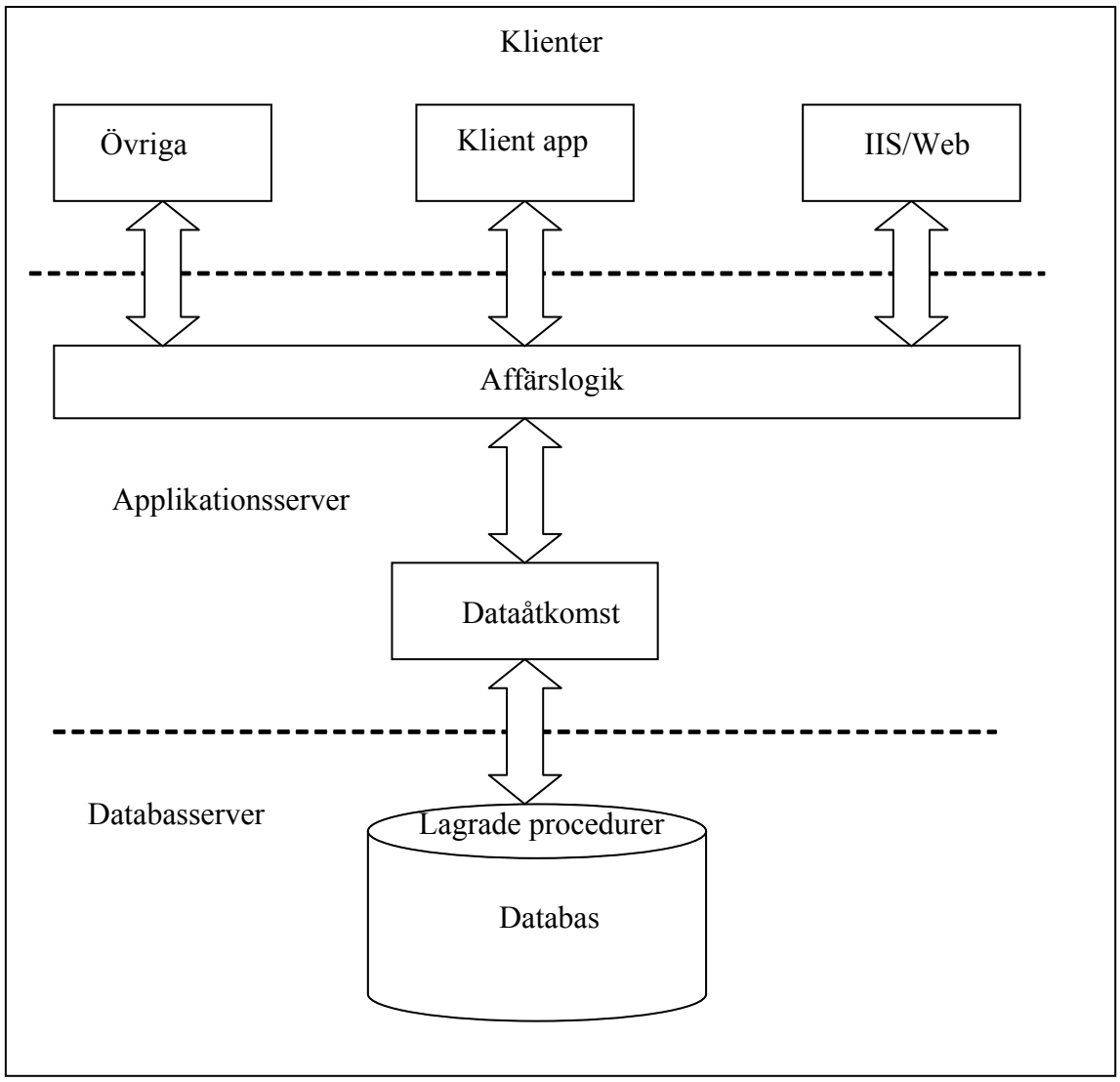
Figurförteckning

Tabellförteckning

1 Bakgrund

Midroc Electros IT avdelning i Karlstad är ett företag som utvecklar IT baserade lösningar för industriellt bruk. För närvarande sker utvecklingen av programvaror med hjälp av Microsofts Visual Studio 6 (VS 6). Midroc Electros önskan att utreda hur Visual Studio Dotnet (VS Dotnet) kan förenkla och effektivisera systembyggnad är grunden för den här rapporten. Med nuvarande verktyg upplever Midroc Electro att utvecklingen är tidskrävande och komplex. För att möjliggöra utveckling av stora komplexa system, vilka ofta kompletteras och förändras under livscykeln, används en skiktad och modulär arkitektur. Systemen delas upp i ett flertal nivåer enligt Figur 1-1 för att åstadkomma önskad skiktning. I de flesta system förekommer:

- Databas med lagrade procedurer.
- Dataåtkomst för att isolera databasen från direkta programanrop.
- Affärslogik, som innehåller systemspecifika regler.
- Klientapplikationer, vilket innefattar alla applikationer som läser eller skriver data i systemet.



Figur 1-1: Treskikts arkitektur

Denna arkitektur har visat sig vara väl fungerande och stabil. För att förbereda införandet av nästa generation av utvecklingsverktyg har vi utrett på vilket sätt Microsoft VS Dotnet kan bidra med förbättrade möjligheter till effektiv systembyggnad för Midroc Electro.

2 Företagspresentation

Midroc Electro är ett teknikföretag med konsult, entreprenad- och serviceverksamhet inom områdena el-installation, kraft, automation, industriell IT, datakommunikation, säkerhet och telefoni. Inom industriell IT är Midroc Electro specialiserade på datorbaserade lösningar för produktions- och labbmiljö. Verksamheten spänner från rena konsultuppdrag till kompletta

systemleveranser. Midroc Electro har 900 anställda och är en del av Midroc Scandinavia AB med totalt 2500 anställda.

3 Uppdragsbeskrivning

För att utreda hur VS Dotnet kan förenkla systembyggnad följer här en lista på punkter som är intressanta. Till hjälp har vi en referensapplikation, d.v.s. en fullt fungerande produkt byggd i VS 6, vilken kommer att tas upp mer i detalj i kapitel 9.

I uppdraget ingår flera delmoment enligt nedan.

3.1 Befintlig arkitektur

För att kunna utvärdera Dotnets möjligheter till effektivisering är det viktigt att sätta sig in i den befintliga arkitekturen som används idag. Som tidigare nämnts finns en referensapplikation till förfogande. Dess implementation och dokumentation blir en referens att jämföra de nya verktygen mot.

3.2 Alternativ utvecklingsmiljö

Med den befintliga arkitekturen som grund undersöks om det finns möjligheter i Dotnet till förenklad/förbättrad systembyggnad utan att ge avkall på:

- Funktionalitet
- Modularitet
- Förändringstålighet

3.3 Dokumentation

En avgörande egenskap vid val av framtida verktyg är möjligheten till att effektivt dokumentera programvara. Detta kan vara inbyggda verktyg eller genom integration med program från tredjepart.

3.4 Implementation

För att utvärdera effektiviteten av implementation i Dotnet har en applikation med motsvarande funktionalitet som den ursprungliga referensapplikationen implementerats.
Intressanta parametrar är:

- Mängden kod som måste produceras manuellt.
- Kodens komplexitet.
- Möjlighet till debuggning.
- XML stöd och hantering. En detaljerad beskrivning av XML återfinns i kapitel 7.

4 Förutsättningar och krav

Microsoft VS Dotnet är en ny utvecklingsmiljö som varken vi eller Midroc Electro har någon större erfarenhet av. Midroc Electro använder för tillfället Microsofts VS 6.0 för att utveckla programvara och målet med vårt arbete var som rubriken säger att utvärdera Microsoft VS Dotnet med avseende på databasstöd. Inte för att hitta brister utan för att undersöka hur de nya verktygen i Dotnet skulle kunna effektivisera utvecklingsarbetet på Midroc Electro. Tillsammans med vår handledare på Midroc Electro har vi gått igenom hur vårt referensprojekt konstruerats. På så sätt har vi fått en inblick i vilka delar i utvecklingsprocessen som Midroc Electro är missnöjda med samt vilka delar som de vill föra över till nästa version av utvecklingsverktyg. Eftersom uppgiften är av utredande karaktär så finns inga direkta krav fastställda utan alla förslag till förbättring får utvärderas i samarbete med Midroc Electro och kan i slutändan vara till grund för framtida val av utvecklingsmiljö.

5 Utförande

Vår referensapplikation, som är utvecklad i VS 6 är ett orderhanteringssystem där ordrar, orderdetaljer och produkter ligger lagrade i en databas. Referensapplikationen använder en skiktad arkitektur med tre lager: ett lager för databasåtkomst, ett lager för affärslogik och ett lager för presentation (GUI). För att kunna se fördelar/nackdelar med att bygga systemet i VS Dotnet har vi valt att implementera alla tre skikten med Visual Basic Dotnet som programspråk. Databasen är uppbyggd av ett flertal tabeller med inbördes relationer, vi har

dock valt att inte använda oss av alla tabeller utan har begränsat oss till tre stycken. Dessa är som följer:

- T_Order - Här lagras alla ordrar. En order kan innehålla noll eller flera orderrader.
- T_OrderLine - Orderrader.
- T_Item - Produkter

För mer detaljerad beskrivning av tabellerna och dess relationer se bilaga A.

Under implementationen av den alternativa applikationen har vi tittat på:

- Kommunikationen mellan lagren.
- Hur mycket beroende som finns mellan lagren.
- Om kommunikationen med databasen har förenklats i ADO Dotnet (ett klassbibliotek för databasåtkomst) och på vilket sätt ADO Dotnet bidrar till att förenkla implementationen av den här typen av system. ADO och ADO Dotnet förklaras närmare i kapitel 7.
- Möjlighet att koppla databasresultat till GUI-komponenter etc.
- Fördelar i att arbeta med cachad data.

Eftersom vi har valt att implementera varje lager i skiktmodellen så har vi fått en god insyn i:

- Mängden manuellt skriven kod.
- Debuggningsmöjligheter.
- Kodens komplexitet.
- XML-stöd i Dotnet.

Databasmodelleringen har vi inte gjort själva, däremot har vi skapat lagrade procedurer för de olika kommandona: Insert, Select, Update och Delete. Dessa har vi skapat genom en guide i VS Dotnet som i sin tur automatiskt skapade dessa i databasen.

6 Terminologi

Här förklaras de mest frekvent återkommande akronymerna. De viktigaste förklaras mer ingående senare i rapporten.

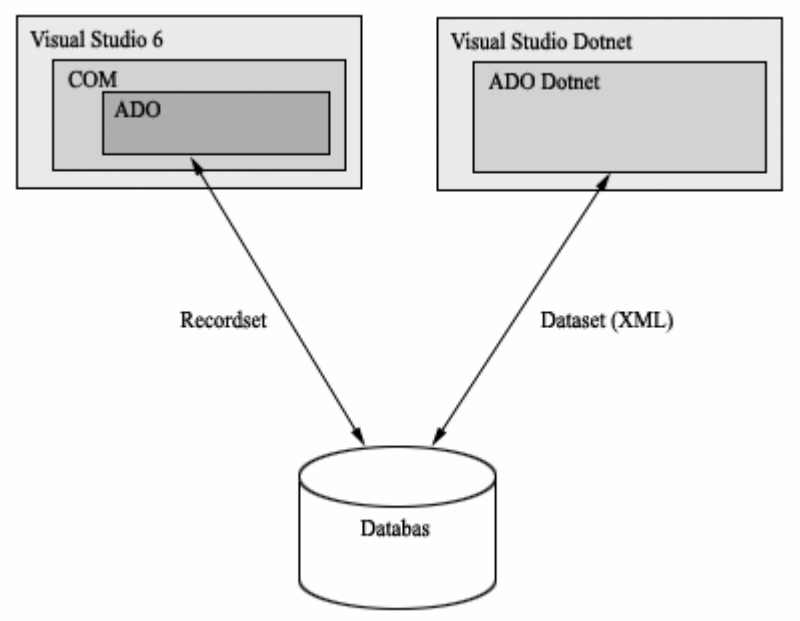
- ADO - ActiveX Data Objects: Microsoft API för access till datakällor.
- COM - Component Object Model: En objektorienterad programmeringsstruktur för kommunikation mellan programkomponenter.

- SCADA - Supervisory Control And Data Acquisition: Ett samlingsnamn för utvecklingsmiljöer avsedda för att bygga operatörsgränssnitt inom industrin.
- ActiveX - Microsofts teknologi att distribuera körbar kod över Internet.
- IIS - Internet Information Services: Microsofts webserver.
- XML - Extensible Markup Language: Ett språk för att beskriva strukturerad data och används frekvent för att skicka data över Internet.
- ASP - Active Server Pages: Microsofts scriptspråk för att skapa dynamiska webbsidor.
- SOAP - Simple Object Access Protocol: Denna teknik möjliggör anrop mellan olika program på ett standardsätt, vilket i sin tur gör det möjligt att bygga program som distribueras över Internet. Precis som XML ingår SOAP i W3C:s standardiseringsprocess, och lämnades in av Microsoft och ett antal andra företag inklusive IBM. Både XML och SOAP har officiellt stöd av i stort sett alla större programvaruföretag i branschen, inklusive Microsoft, IBM, HP, Sun och Oracle.
- DOM - Document Object Model: DOM är en W3C-standard som innehåller ett gränssnitt för programmässig åtkomst av XML-data.
- CLR - Common Language Runtime: Dotnets kärna där program exekverar.
- IL - Intermediate Language: Ett språk som alla Dotnet applikationer kompileras till oavsett programmeringsspråk.
- Unmanaged / Managed code: Konventionellt kompilerad kod / kod kompilerad för CLR
- Dataset: ADO Dotnets XML-baserade objekt för lagring av data och relationer från databaser.
- Recordset: ADOs objekt för lagring av data från databaser.

7 Teknologier

För att förstå olika typer av teknologier följer här en kort beskrivning av de delar som ingår antingen som integrerade delar av utvecklingsverktygen eller som fristående komponenter. Även komponenter som inte är någon del av verktygen utan rena programmeringsstrukturer behandlas. Alla dessa komponenter är nödvändiga för att den skiktade arkitekturen skall kunna upprätthållas. ADO och ADO Dotnet (ActiveX Data Objects) för kommunikation med datakällor samt skillnaden de emellan, XML (Extensible Markup Language) för ett

gemensamt språk mellan applikationer och COM (Component Object Model) för att möjliggöra kommunikation mellan lagren i den befintliga arkitekturen.

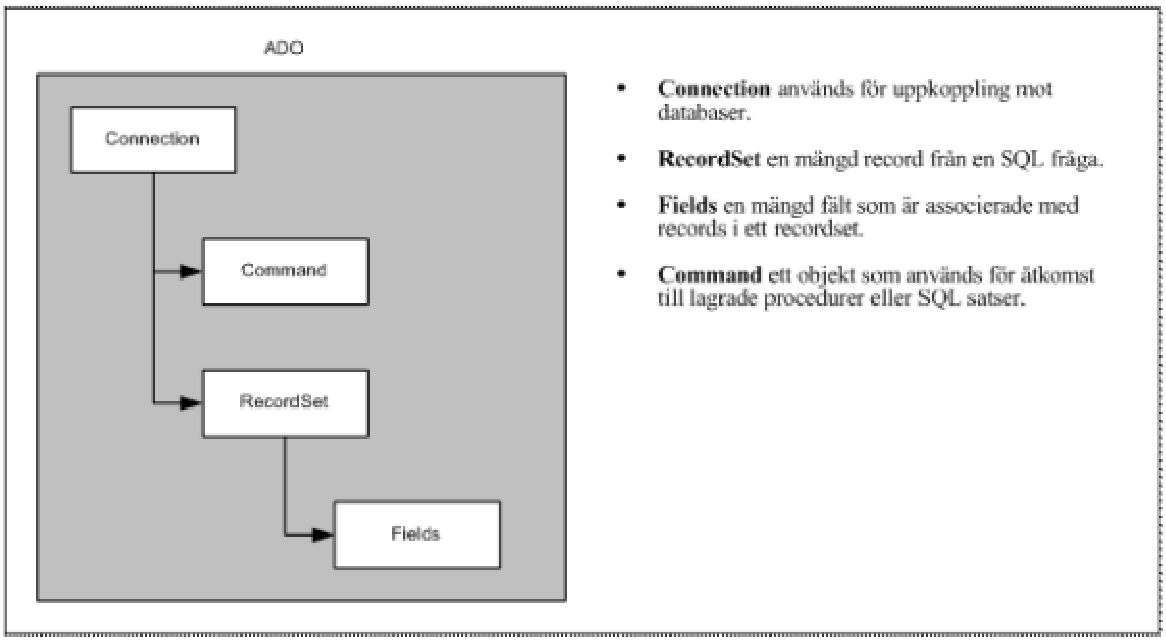


Figur 7-1: Komponentöversikt

Förhållandet mellan dessa komponenter och skillnaden mellan VS 6 och VS Dotnet visas i Figur 7-1. En mer detaljerad beskrivning av ovan nämnda termer följer nedan

7.1 ADO - ActiveX Data Objects

ADO är ett av Microsoft framtaget gränssnitt för dataåtkomst. För kommunikation med databaser använder ADO s.k. dataproviders, som kan ses som specialanpassade drivrutiner för olika databaser. Genom att anropa ADO:s API möjliggörs operarationer mot databasen. Svaret från databasen kommer sedan som ett recordset. Ett recordset är ett objekt som innehåller den data som efterfrågats i en SQL-sats. Recordsetet kan användas för operationer mot databaser eller för presentation, tex. i ett GUI. En av nackdelarna med ADO är hanteringen av cachad data, d.v.s. ADO har ingen inbyggd mekanism för typkontroll av indata utan överlåter typhanteringen till databasen, vilket kan vara ineffektivt och tidskrävande. Nedan i Figur 7-2 ses ADO:s objektmodell.



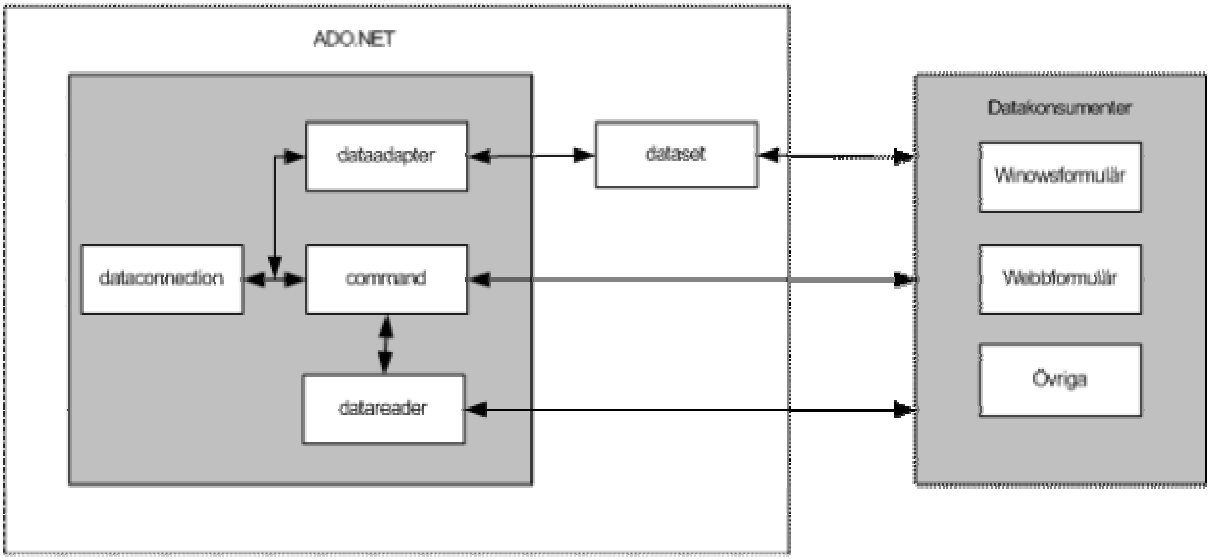
Figur 7-2: Objektmodell ADO

7.2 ADO Dotnet - ActiveX Data Objects Dotnet

ADO Dotnet är ett gränssnitt för dataåtkomst som används framförallt för databasaccess men kan användas för access till alla typer av datakällor. ADO Dotnet är ett klassbibliotek som tillhandahåller tjänster för just dataåtkomst. Klasserna i ADO Dotnet består av två olika typer av komponenter:

- Dataleverantörer (Data Providers) - hanterar kommunikationen med fysiska datakällor.
- Objektet Dataset - representerar aktuell data.

Båda dessa komponenter kan kommunicera med datakonsumenter (Data Consumers se Figur 7-3), som oftast är webformulär eller windowsformulär. ADO Dotnet har stöd för en mängd olika databaser. För att kommunikationen skall kunna ske med olika databaser så krävs det att olika dataleverantörer finns att tillgå. Om en databas av någon anledning behövs ersättas av en databas av annan typ så räcker det med att i källkoden byta den dataleverantör som ADO Dotnet arbetar mot. SQL-syntaxen kan dock skilja sig mellan olika databaser, men ligger SQL-satserna som lagrade procedurer (SQL-satser som ligger lagrade i databasen istället för att dom är hårdkodade i programkoden) så kan de bytas ut till nya som stödjer den nya databasens syntax.



Figur 7-3: Objektmodell ADO Dotnet

ADO Dotnet är motsvarigheten till ADO som används i VS 6 men den enda likheten mellan de två är egentligen bara namnet och syftet. ADO Dotnets dataleverantörer levererar endast ett begränsat API med de grundläggande funktionerna och låter sedan programmeraren komplettera API:t med önskad funktionalitet. Dotnet använder inte COM-objekt för dataåtkomst utan använder istället klassbibliotek. ADO Dotnet skiljer sig sedan markant från ADO när det gäller kommunikationen mot datakällan. ADO kommunicerar hela tiden med datakällan som i sin tur håller reda på status och metadata (datatyper) medan ADO Dotnet plockar ut de data som efterfrågas inklusive metadata som ett Dataset (ADO Dotnets motsvarighet till ADO:s recordset) och bearbetar sedan Datasetet frånkopplat från datakällan (cachad data).

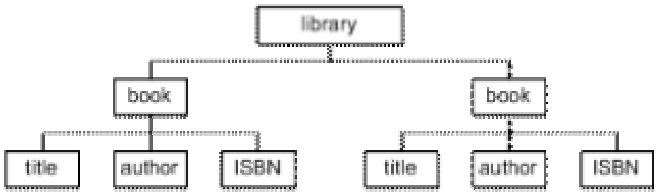
7.3 XML - Extensible Markup Language

Anledningen till att vi tar upp XML är att ADO Dotnet bygger mycket på XML och att XML bidrar till ett visst oberoende mellan program som kan läsa och hantera XML-data. Midroc Electros nuvarande utvecklingsmiljö bygger på XML för att skicka data mellan de olika skikten. XML är ett språk för att beskriva data och används frekvent för att skicka data över Internet. Syntaxen för XML liknar HTML men har inga fördefinierade taggar. XML-dokument måste dock uppehålla en viss struktur för att vara giltiga. Detta gör XML till ett lämplig sätt att beskriva strukturerad data som t.ex. tabelldata från en databas. Ett exempel på ett XML-dokument som beskriver böcker i ett bibliotek kan se ut som i Figur 7-4.

```
<library>
        <book>
        <title>Name of the book</title>
        <author>Author name</author>
        <ISBN>Number</ISBN>
        </book>
        <book>
        <title>Name of another book</title>
        <author>Author name</author>
        <ISBN>Number</ISBN>
        </book>
</library>
```

Figur 7-4: Exempel på XML struktur

Naturligtvis kan ett XML-dokument vara mer komplext än vad exemplet visar, här visas bara strukturen. Attribut kan också infogas i taggarna för att beskriva olika element. Inga av taggarna library, book, title, author eller ISBN är fördefinierade utan är valda för att på ett lättförståligt sätt beskriva innehållet i ett bibliotek. Varje tagg måste dock till skillnad mot HTML avslutas med motsvarande snedstreck följt av taggens namn. Som synes har XML en trädliknande struktur med ett enda topelement som innehåller alla andra element. Strukturen i ett XML-dokument kan representeras genom ett träddiagram vilket kan ses i Figur 7-5.



Figur 7-5: Trädhierarki XML

7.4 COM - Component Object Model

Microsoft COM är en objektorienterad språkoberoende programmeringsstruktur där programmeraren kan använda allt från C, C++ till ADA och Basic för att skapa COM-objekt. Förutsättningen är att språket klarar av att generera den binära layout som definierar ett COM-objekt, d.v.s. att objektets struktur och gränssnitt följer de regler som finns uppsatta för COM-objekt. COM medger kommunikation mellan programkomponenter i olika program. Det är vanligt är att COM-objekten ligger i ett s.k. Dynamic Link Library (DLL). En DLL-fil kan

innehålla många COM objekt och när en DLL-fil laddats i ett program kan sedan de objekt som DLL-filen exponerar ut anropas. DLL-filer kan ses som samlingsplatser för COM objekt där objekten kan laddas in och ur andra program dynamiskt. Det finns tre olika typer av COM objekt:

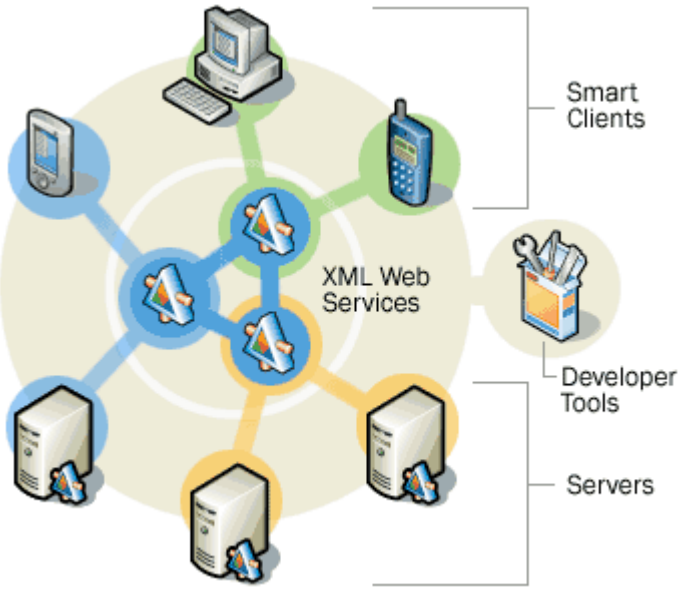
- In-process - objekten ligger i en DLL-fil som inte själv är körbar.
- Local - objekten ligger i en EXE-fil som en separat körbar process.
- Remote - objekten ligger i en DLL-fil eller som EXE-fil på en annan maskin, distibuted COM (DCOM).

Varje COM-objekt får ett unikt nummer (GUID) som definierar var objektet ligger i filsystemet. Denna GUID hanteras av operativsystemet genom Windowsregistret. Detta medför programmering mot ett objekt utan att behöva veta var objektet finns. Det här är vad Midroc Electro utnyttjar för att skapa system i flera skikt för att isolera t.ex. databasåtkomsten från affärslogik och användargränssnitt.

7.5 Vad är Dotnet?

Dotnet är Microsofts plattform för XML-baserade webbtjänster, en programvara som skall hjälpa till att sammanlänka datorer och människor på ett enhetligt sätt. Kommunikationen sker med ett World Wide Web Consortium (W3C) standardiserat XML-protokoll, Simple Object Access Protocol (SOAP), se Figur 7-6. Denna teknik gör det möjligt att skapa och använda XML-baserade program, processer och webbplatser som delar och kombinerar information och funktioner med varandra på alla plattformar. Dotnet innehåller en omfattande uppsättning produkter som är byggda enligt bransch- och Internet standard. Dotnet utgör eller kommer att utgöra en del av de verktyg och servrar som Microsoft kommer att tillhandahålla i framtiden.

Dotnet-applikationer har stöd för "remoting" för att skapa täta, snabba, binära kommunikationskanaler eller "lösare förbindelser" mellan webbtjänster och konsumenter/klienter via SOAP-protokollet. Kommunikationsspråket i SOAP är XML.



Figur 7-6: SOAP - XML

I Dotnet finns ett ramverk (Dotnet framework), som är grunden för alla Dotnet-baserade applikationer. För att kunna köra en Dotnet-applikation krävs att Dotnet framework finns installerat på datorn. I Windows XP är Dotnet framework förinstallerat och finns att ladda ner till ett flertal av Microsofts tidigare operativsystem. Alla Dotnet-baserade applikationer oavsett vilket programspråk de är skrivna i, använder Dotnet framework. Dotnet frameworks hjärta heter Common Language Runtime (CLR), denna fungerar ungefär som Javas virtuella maskin och sköter grovjobbet som t.ex. minneshantering, automatisk skräphantering (garbage collection) och typkontroll (type check). I Dotnet kompileras källkoden inte direkt till maskinkod utan kompileras istället till ett mellanliggande språk ett så kallat Intermediate Language (IL) som hanteras av CLR. Kod skriven för CLR kallas för managed code (kod skriven för Dotnet ramverket).

CLR är anledningen till att Dotnet kan kallas för språkoberoende. En applikation kan vara skriven i ett språk och använda klasser, ActivX:er eller DLL:er skrivna i andra språk. Till exempel kan en VB-klass ärva funktionalitet från en klass skriven i C#. Dotnet framework använder en så kallad Just In Time (JIT) kompilator för att kompilera programmet till

maskinkod. JIT kompilatorn plockar vid körning in okompilerade delar av programmet och kompilerar dem allt eftersom dom behövs. VS Dotnet har genom att använda CLR raderat ut COM-objektens berättigande i Dotnet-applikationer. VS Dotnet har fortfarande stöd för hantering av COM-objekt eftersom många väl fungerande fortfarande finns att tillgå.

Microsoft har implementerat tre fullvärdiga språk, Visual Basic Dotnet (som till skillnad från tidigare är helt objektorienterat), C++ Dotnet samt ett nytt språk som heter C# (C sharp) och som starkt påminner om Java d.v.s. en uppstädad C/C++ syntax med skräphantering och hårdare typning. För övrigt är hela Dotnet framework och VS Dotnet skrivet i C#. Dotnet är även öppet för fler programspråk såsom J++ och Borland Delphi.

8 Utvecklingsverktyg

För att skapa förståelse för vad utvecklingsverktyg är och hur de används inom programvaru-utveckling förklaras här kortfattat de utvecklingsverktyg som berörs i rapporten.

8.1 Visual Studio 6.0

VS 6 kan ses som föregångaren till VS Dotnet och är en uppsättning verktyg från Microsoft för utveckling av programvara. VS 6 stödjer en rad olika programspråk som t.ex. Visual Basic, Visual J++ (Java) och Visual C++. VS 6 tillåter byggnad av system direkt mot databaser eller mot Internet. Genom att använda COM underlättar det att bygga flerskiktade webbaserade lösningar, eller genom att använda ADO utveckla databasapplikationer. På grund av att VS 6 stödjer flera olika språk tillhandahåller miljön olika möjligheter att utveckla en mängd olika typer av applikationer. I Visual Basic finns ett kraftigt verktyg för att utveckla grafiska användargränssitt. Alternativt är att använda MFC (Microsoft Foundation Classes, ett API för att skapa GUI) i Visual C++. Eftersom COM-tekniken är integrerad i VS 6 tillåts utveckling av objekt i ett språk för att sedan användas i andra applikationer skrivna i ett anndra språk. VS 6 tillhandahåller ett flertal andra tjänster för att underlätta programmering t.ex. SourceSafe som är ett versionshanteringsystem, debuggning i realtid och webbutveckling med Visual InterDev med mera. VS 6 är det verktyg som Midroc Electro för närvarande använder för att utveckla sina system. I bilaga B visas exempel på vilka verktyg VS 6 innehåller som underlättar utveckling av programvara. Som synes är Visual Basic, Visual C++ och Visual J+ inte bara programspråk utan hela utvecklingssystem vilket innebär att

språken är hårt knutet till sina respektive utvecklingsmiljöer och att koden blir plattformsberoende.

8.2 Visual studio Dotnet

Microsoft VS Dotnet är en efterföljare till VS 6. VS Dotnet är hårt knutet till Dotnet framework. VS Dotnet erbjuder flera nya sätt att bygga system bl.a. genom det nya programspråket C# (C sharp) och det nya och förbättrade ADO Dotnet för dataaccess. Dotnet är även hårt knutet till XML som tillhandahåller ett standardspråk för textbaserad kommunikation mot t.ex. databaser och webbtjänster. Precis som VS 6 har VS Dotnet debuggning i realtid och program för versionshantering mm. Samtliga språk i Dotnet kan användas i en gemensam utvecklingsmiljö, vilket innebär att verktyget är oberoende vilket språk programmeraren föredrar. Det finns även stöd för att integrera flera språk. Kod som är skriven för att köras i CLR blir s.k. managed code och innebär att koden inte är interpreterande utan kompileras under runtime (se JIT kapitel 7.5). Det går även att skriva unmanaged code (konventionellt kompilerad kod) i Visual C++ för användande i äldre standarder, t.ex. COM eller DLL:er men då förloras de fördelar som managed code erbjuder med avseende på Dotnet:s runtime services som t.ex. garbage collection etc. Dotnet tillåter dock användandet av både managed och unmanaged kod i samma applikation. Dotnet har ett gemensamt klassbibliotek för alla typer av programutveckling. T.ex. webbformulär, XML-baserade webbtjänster, kommandoradsapplikationer och grafiska Dotnet applikationer. VS Dotnet erbjuder en utvecklingsmiljö för utveckling av applikationer i ett s.k. designläge, d.v.s. utveckling sker genom ett grafiskt gränssnitt vilket gör att den manuellt producerade koden hålls nere. Mycket av den kod som produceras av VS Dotnet döljs från programmeraren som istället kan koncentrera sig på den kod som han/hon själv skrivit. Dock är VS Dotnet bakåtkompatibel med ADO och andra tjänster som finns i tidigare versioner av VS. I bilaga B ses en tabell över de verktyg som ingår i VS Dotnet. Till skillnad mot VS 6 använder VS Dotnet inte längre COM-objekt utan klassbibliotek t.ex. för dataåtkomst via ADO Dotnet eller för Windows grafiska komponenter vid utveckling av GUI.

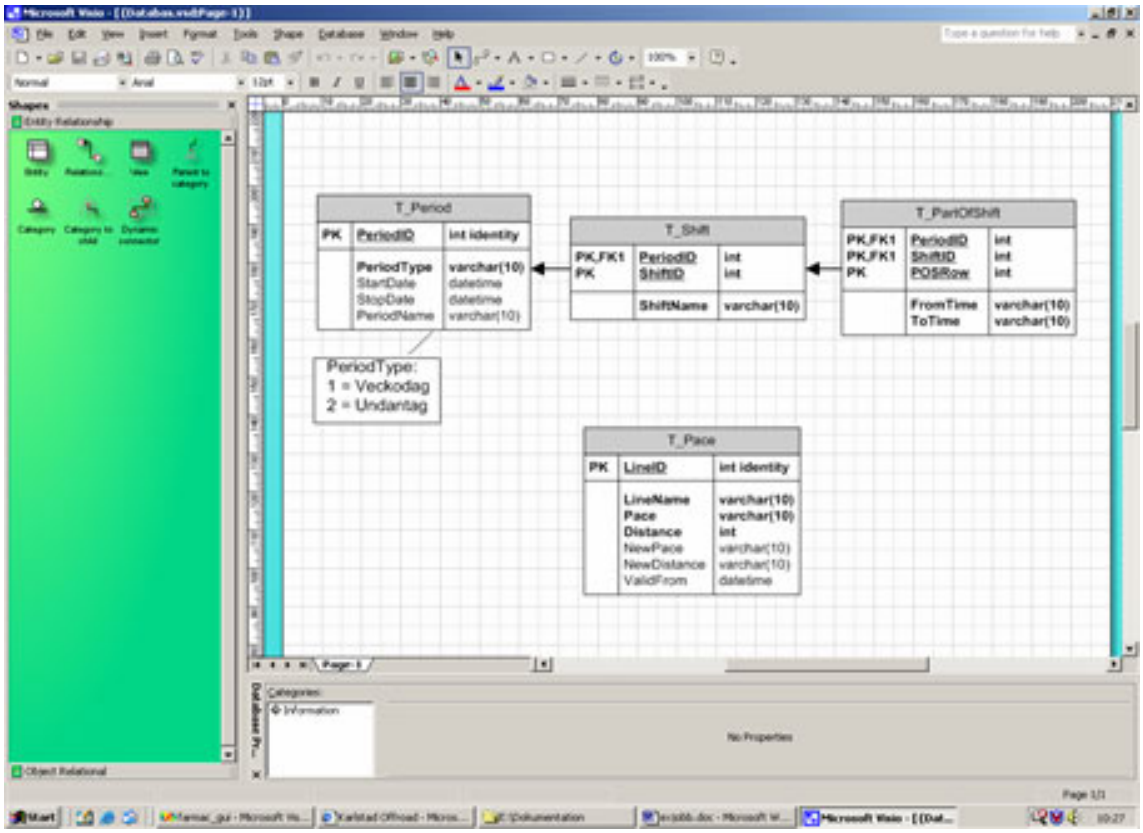
8.3 Övriga utvecklingsverktyg (Midroc Electro)

För systembyggnad använder Midroc Electro ett flertal andra verktyg förutom VS 6. T.ex. verktyg för databasmodellering, versionshanterare och en kodgenerator, denna kodgenerator har Midroc Electro själva skrivit som ett macro till Microsoft Visio som beskrivs i nästa

kapitel, för att skapa lagrade procedurer, script för skapande av databaser och tabeller samt generera en viss mängd kod i form av moduler i Visual Basic till sina applikationer. Nedan följer en kort beskrivning av en del av dessa verktyg.

8.3.1 Microsoft Visio

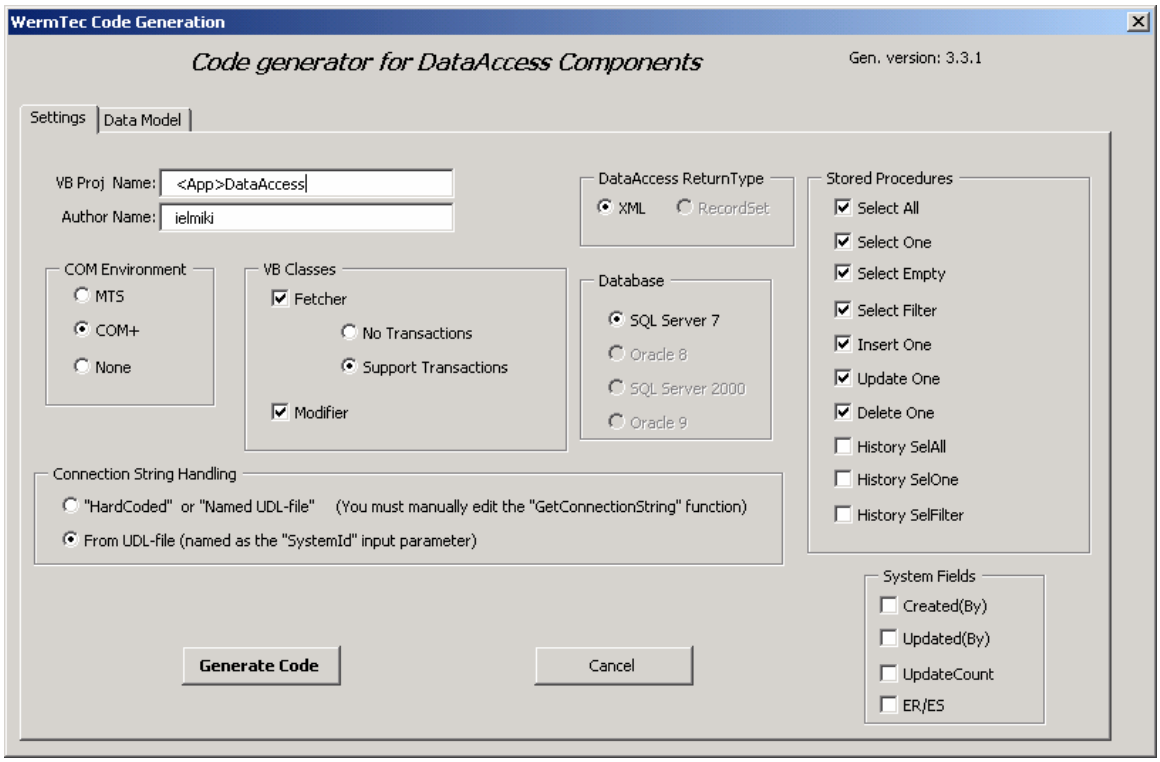
Microsoft Visio är ett program för att skapa flödesscheman, modellera databaser och Unified Modeling Language (UML) diagram. Visio medger skapandet av ritningar över nätverk, byggnader etc. Fördelen med detta program är just stödet för databasmodellering (se Figur 8-1) kompletta med tabeller, datatyper och relationer i ett grafiskt gränssnitt. Efter att en modell av en databas har skapats finns det via ett macro (beskrivs mer detaljerat i nästa kapitel) möjlighet att generera SQL-script avsedda att köra mot databasservern. Även kod och definitioner i form av moduler (Visual Basic) för de skikt (dataåtkomst, affärslogik och presentation) som skall användas i systemet skapas. Detta medför att en del tid kan sparas i utvecklingsarbetet. Dessa moduler kan sedan öppnas i VS för fortsatt utveckling. Användningsområdet för Visio är ganska stort men det är till databasmodellering som Midroc Electro använder det till största delen.



Figur 8-1: Databasmodellering i Visio

8.3.2 Kodgenerator

Midroc Electros kodgenerator är en egenutvecklad applikation som genererar lagrade procedurer för databasen. Villkoret för att kodgeneratorn skall fungera är att tabellerna är korrekt skapade i Visio. Kodgeneratorns gränssnitt låter programmeraren välja vilka procedurer som skall genereras för databasen. Kodgeneratorn genererar också ett SQL-skript för hela databasen inklusive dess lagrade procedurer. En fetcher klass och en modifier klass genereras samtidigt för varje tabell. Fetcher klassen handhar all läsning av data från databasen medan modifier klassen handhar all skrivning till databasen. Alla tabeller i databasen får på så vis två klasser var som kan läsa och modifiera databasen med hjälp av dess lagrade procedurer. Svaret från databasen görs om till XML i fetcher-klassen för att kunna skickas till ovanliggande skikt i plattformen. Klassernas metoder återfinns sedan publikt i DataAccess.dll.



Figur 8-2: Skärmbild Midroc Electros kodgenerator

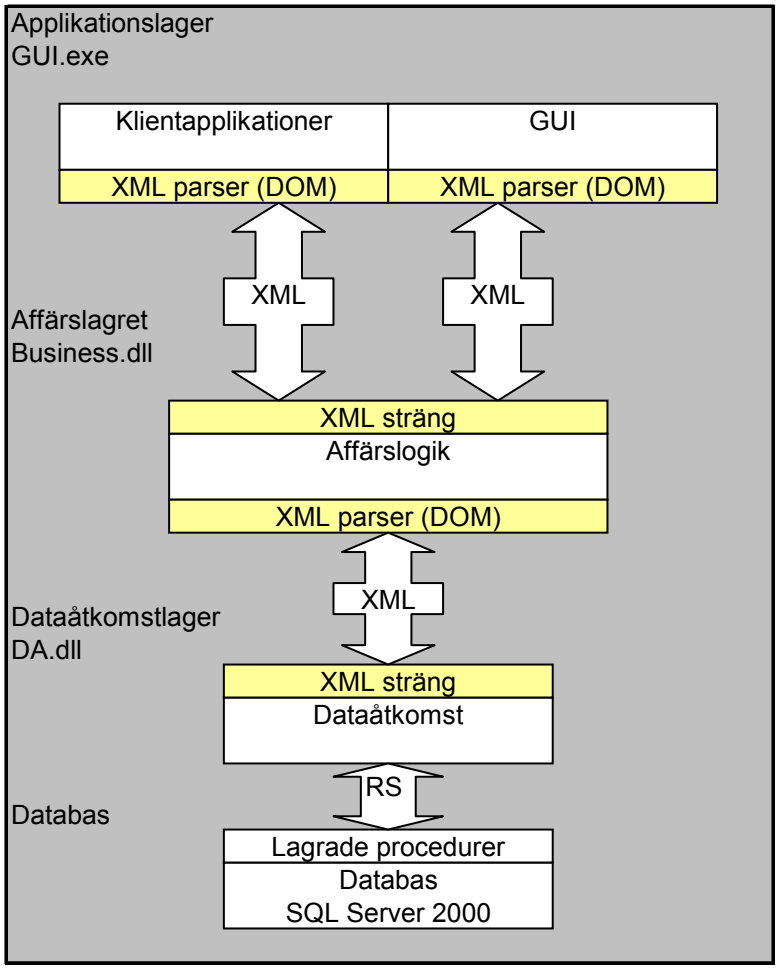
9 Befintlig referensapplikation utvecklad i Visual studio 6

Referensapplikationen är ett orderhanteringssystem med tillhörande användargränssnitt för hantering av ordrar. Från det grafiska gränssnittet kan användaren lägga till, ta bort och ändra

ordrar. Referensapplikationen bygger på trelagerprincipen och datatransporten mellan lagren sker med hjälp av XML. Nedan följer en mer detaljerad beskrivning av de olika lagren i referensapplikationen.

9.1 Datatransport mellan lagren

För att skicka data med hjälp av XML mellan skikten så används W3C Document Object Model (DOM). W3C DOM är en standard som innehåller ett gränssnitt för programmässig åtkomst av XML-data. W3C DOM är en språk- och plattformsoberoende standard som gör det möjligt att tolka och bearbeta koden i ett XML-dokument på ett standardiserat sätt. Svaret från databasen skickas till dataåtkomstlagret i form av ett ADO-recordset som sedan med hjälp av DOM konverteras till en XML-sträng. XML strängen tas sedan emot i affärslagret av en DOM-parser. DOM-parsern kan sedan läsa och modifiera XML-strängen efter behov. XML strängen skickas sedan vidare upp till applikationslagret där ännu en DOM-parser tar hand om strängen. Efter att en post ändrats, infogats eller tagits bort i användargränssnittet så måste XML-strängen manuellt uppdateras för att sedan kunna skickas ner till databasen. Uppdateringen av XML-strängen är en kritisk och tidskrävande operation som börjar med att en ny ”tom” fråga ställs mot databasen för att få tillbaks XML-strängens struktur. Denna struktur används sedan för den nya strängen som skall skickas ner för uppdatering i databasen. Nackdelen med detta förfarande anser vi vara att den manuella uppbyggnaden av XML-strängen är tidskrävande både accessmässigt och implementationsmässigt. Fördelen ligger i att en XML-sträng kan skickas till och tas emot från vilken annan applikation som helst bara applikationen i fråga kan hantera XML. Modellen tillåter också att yttre applikationer interagerar med applikationen direkt mot affärslagret.



Figur 9-1: Skiktmodell referensapplikation i VS 6

9.2 Dataåtkomstlagret (DataAccess)

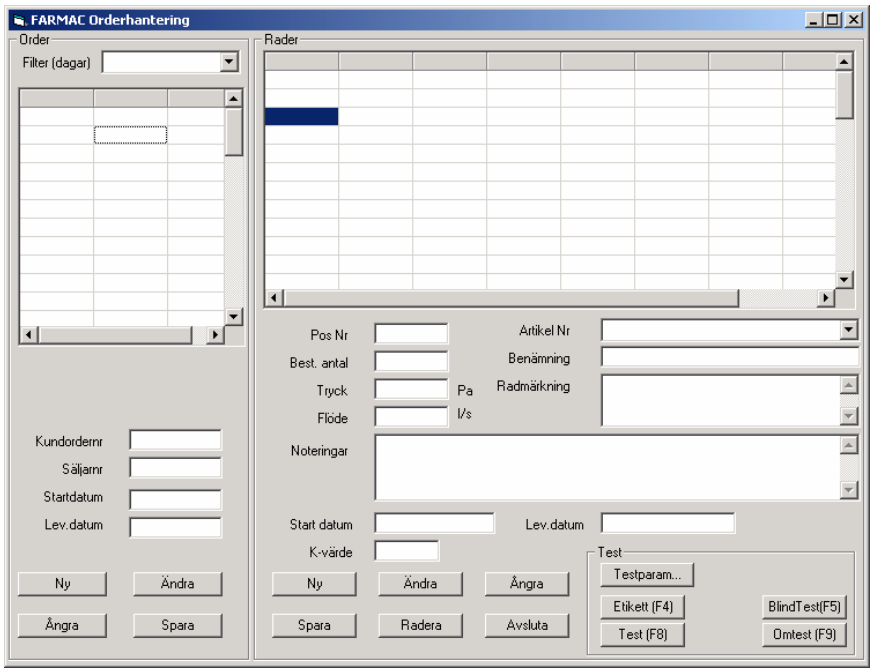
Dataåtkomstlagret är den del av applikationen som har kontakt med databasen. Det är i dataåtkomstlagret som de klasser som skapats av kodgeneratorn för att nå de lagrade procedurerna ligger. Dataåtkomstlagret blir specialanpassat mot en viss typ av databas, m.a.o. måste ett nytt dataåtkomstlager skapas vid byte av databas t.ex. från SQLserver till Oracle. Kommunikationen mellan dataåtkomstlagret och databasen sker med hjälp av ADO-recordset. Ett recordset konverteras m.h.a. DOM till en XML-sträng innan den skickas vidare uppåt i modellen och på motsvarande sätt så byggs recordsetet ihop av en XML-sträng som kommer ner från det överliggande lagret, som i detta fall är affärslagret. Koden för dataåtkomstlagret kompileras till en DLL-fil som sedan refereras till från affärslagret.

9.3 Affärslogiklagret (BusinessRules)

I affärslagret sker de s.k. affärsspecifika operationerna som t.ex. matematiska beräkningar, datautbyte med externa utrustningar, operationer som att räkna antalet nya poster och kontroll av inmatad data. Affärslagret kommunicerar både uppåt och nedåt i modellen via XML. Detta innebär att DOM-objektet används flitigt för att parsa inkommande XML-strängar samt till att generera nya XML-strängar för vidare transport inom modellen eller till yttre enheter.

9.4 Presentationslagret (GUI)

Presentationslagret har hand om interaktionen med användaren i form av ett grafiskt gränssnitt. I presentationslagret tas XML-strängen emot och DOM-parsern plockar ut valda delar av strängen och placerar in rätt data på rätt plats i gränssnittet.



Figur 9-2: Skärmbild referensapplikation i VB 6

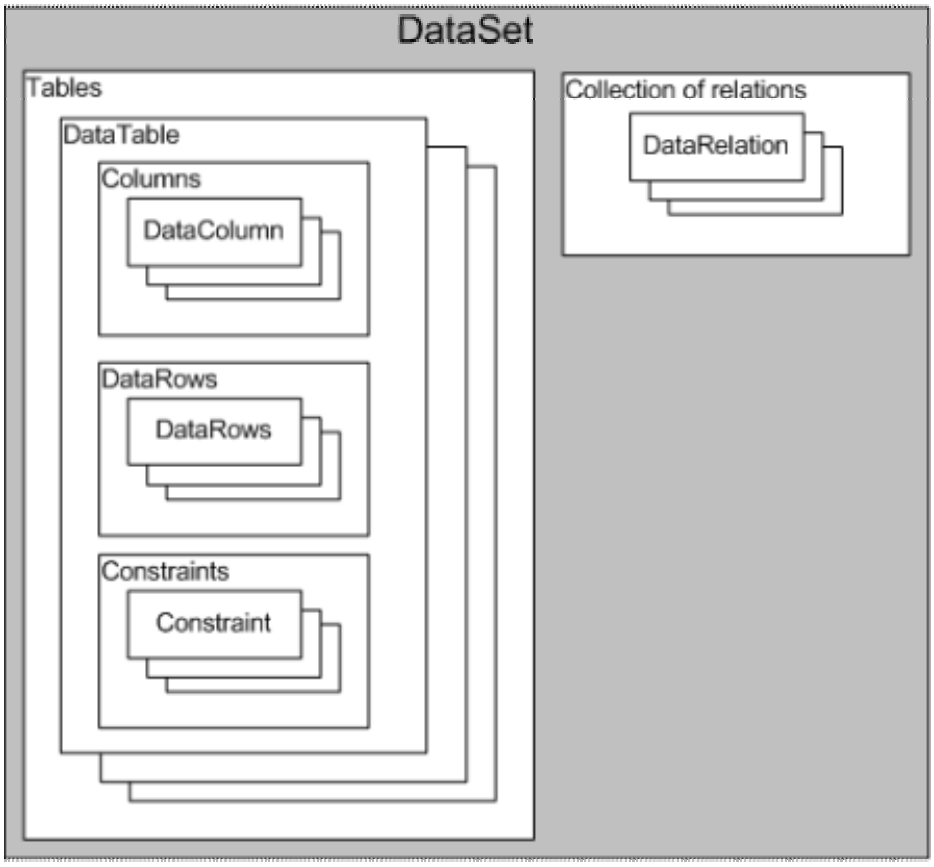
10 Alternativ lösning utvecklad i Visual studio Dotnet

Här beskrivs hur vi valde att implementera den alternativa lösningen i verktyget VS Dotnet. Det här är grunden till att vi skall kunna se skillnader, likheter och förbättringar/försämringar i VS Dotnet jämfört med VS 6. Och att kunna se vilket stöd för databasmodellering, automatiskt genererad kod, dokumentation etc. som finns tillgänglig. För att skapa förståelse

för de objekt som används i applikationen följer här en beskrivning av ADO Dotnets Dataset och DataAdapter.

10.1 Dataset

Ett Dataset är en struktur för minneslagring som representerar både tabellers data och relationer. Ett Dataset kan fyllas/användas med flera tabeller samtidigt ihop med deras inbördes relationer och begränsningar. Ett Dataset fylls lämpligast med hjälp av en DataAdapters inbyggda funktion *fill(dataset)*. Ett Dataset bygger i grunden på W3C kompatibel XML kod vilket gör ett Dataset till ett utmärkt medium för transport av data mellan olika lager över Internet och genom brandväggar. Ett Dataset består av en eller flera tabeller representerade som XML samt deras eventuella relationer. Det är ett Datasets arkitektur som utgör grunden för cachad och distribuerad data i ADO Dotnet. När ett Dataset fylls med data från tabeller från en databas så är det inte enbart tabellernas innehåll som sparas utan även tabellers relationer och regler (constraints). Tack vare att tabellernas regler (metadata) finns lagrade i Datasetet tillsammans med dess tabelldata så kan ett Dataset fungera som en cachad kopia av en begränsad del av datakällan. Flera operationer som t.ex. Insert, Delete eller Update kan sedan utföras mot Datasetet som i sin tur kan kontrollera att operationerna inte bryter mot de regler som specificerats för tabellerna i databasen. För att sedan uppdatera databasen med de nya ändringarna så används DataAdapterns inbyggda funktion *update(dataset)*. Att jobba mot ett Dataset är tidsbesparande eftersom kommunikationen mot databasen kan hållas nere, samtidigt som databasservern avlastas.



Figur 10-1: Objektmodell Dataset

ADO Dotnet har stöd för två typer av Dataset - typade och otypade. Otypade Dataset är en direkt instans av Dotnet frameworks *System.Data.Dataset* medan typade Dataset är en klass som ärver från *System.Data.Dataset*. Skillnaden ligger i att ett typat Dataset synliggör sina tabeller och kolumner som objektegenskaper. Typade Dataset underlättar hanteringen på så sätt att kolumner och tabeller kan refereras till efter deras namn. En annan fördel är att de är s.k. hårt typade och medger kontroll av datatyper redan under kompileringen, d.v.s. om programmeraren i sin kod försöker att tilldela en tabellcell ett värde med felaktig datatyp så genereras ett fel av kompilatorn. Med otypade Dataset saknas möjligheten att referera till kolumner och tabeller via namn och får istället indexeras med nummer (*ex. tabell(1) rad(4) kolumn(6)*). Otypade Dataset används oftast när en applikation tar emot ett Dataset från en mellankomponent eller en webbtjänst och då strukturen är okänd. För exempel på Datasets objektmodell se Figur 10-1.

10.2 DataAdapter

Objektet DataAdapter är som ett ”klister” mellan datakällan och objektet Datset. I abstrakta ordalag kan man säga att en DataAdapter tar emot data från ett Connection-objekt

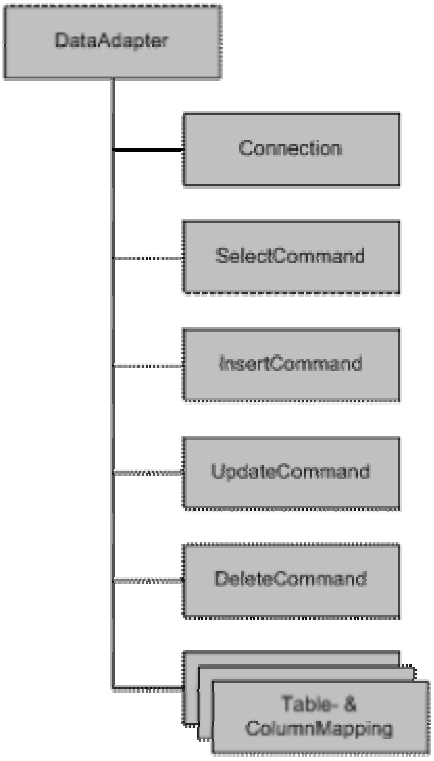
(representerar den fysiska anslutningen till datakällan) och skickar datan vidare till ett Dataset. Samma gäller åt andra hållet då datan skickas från Datsetet till DataAdaptern när en uppdatering av datakällan ska ske. Datakällan kan vara vilken typ av data som helst, inte bara databaser. För att klara av att uppdatera en datakälla så har en DataAdapter referenser till fyra olika Command-objekt (en klass för att skapa kommandon som exekveras mot datakällan, antingen som SQL-satser eller som lagrade procedurer) en för varje tillgänglig åtgärd. Följande Command-objekt finns att tillgå:

- SelectCommand – Hämta data
- InsertCommand – Lägg till data
- UpdateCommand – Uppdatera data
- DeleteCommand – Ta bort data

Precis som objekten Command och Connection (se Figur 10-2) är DataAdapter en del av Data Provider och det finns en specifik version av DataAdapter i de olika Data Providers. Komponenter av typen Data Provider är speciellt anpassade för en viss datakälla. Det ingår två stycken Data Providers i Dotnet framework:

- OleDBDataAdapter - en generell Data Provider för kommunikation med alla typer av datakällor.
- SqlDataAdapter - en Data Provider som är speciellt anpassad för kommunikation med Microsoft SQL server 7.0 eller senare.

Andra Data Providers som kan kommunicera med andra typer av databaser som DB2, Oracle etc. finns att tillgå från tredjepart. Om en datakälla som en applikation arbetar byts ut så behövs inte hela programmet skrivas om utan det räcker i många fall med att byta ut DataAdaptrarna mot nya som kan hantera den nya datakällan. Ytterligare en fördel är att en programmerare kan via en guide skapa DataAdaptrar i VS Dotnet och kan då välja att använda SQL-satser eller lagrade procedurer. Guiden låter programmeraren välja om han/hon vill använda redan befintliga lagrade procedurer eller om den skall skapa nya. Det räcker med att användaren anger en SELECT-sats så kan guiden generera de resterande kommandona. Dessa kan sedan skapas direkt i databasen eller så kan SQL-script genereras för att köras mot databasen vid ett senare tillfälle. När uppdatering av datakällan skall ske anropas den aktuella DataAdapterns funktion Update med ett Dataset som parameter. Funktionen Update väljer själv ut lämplig Command-objekt som skall utföras mot datakällan. Detta innebär att programmeraren inte behöver skriva olika funktioner de olika kommandona som skall utföras.



Figur 10-2: Objektmodell DataAdapter

Som nämnts tidigare kan en DataAdapter fylla tabeller i ett Dataset och läsa ändringar i det samma för att uppdatera datakällan. För att hålla reda på vad som ska var i datakällan har DataAdaptern en del supportegenskaper. Bl.a. en kollektion av TableMappings, som är en egenskap för mappningen mellan tabeller i Datasetet och datakällan. Varje TableMapping har en likadan egenskap för mappning mellan kolumnerna i Datasetet och datakällan. Dessa mappningar är till för att hålla reda på vilken tabell eller kolumn i databasen som avbildas i Datasetet.

10.3 Dotnet applikationen

Applikationen är precis som referensapplikationen ett orderhanteringssystem, där användaren kan lägga till, ta bort och uppdatera ordrar. Det grafiska gränssnittet presenterar datan genom två stycken DataGrids. En DataGrid är en grafisk komponent för att presentera data ur ett Dataset. Ena DataGriden presenterar ordrarna som finns i databasen medan den andra DataGriden presenterar alla orderrader som finns i den aktuella ordern. I Datasetet finns tre tabeller som är fyllda med data från databasen:

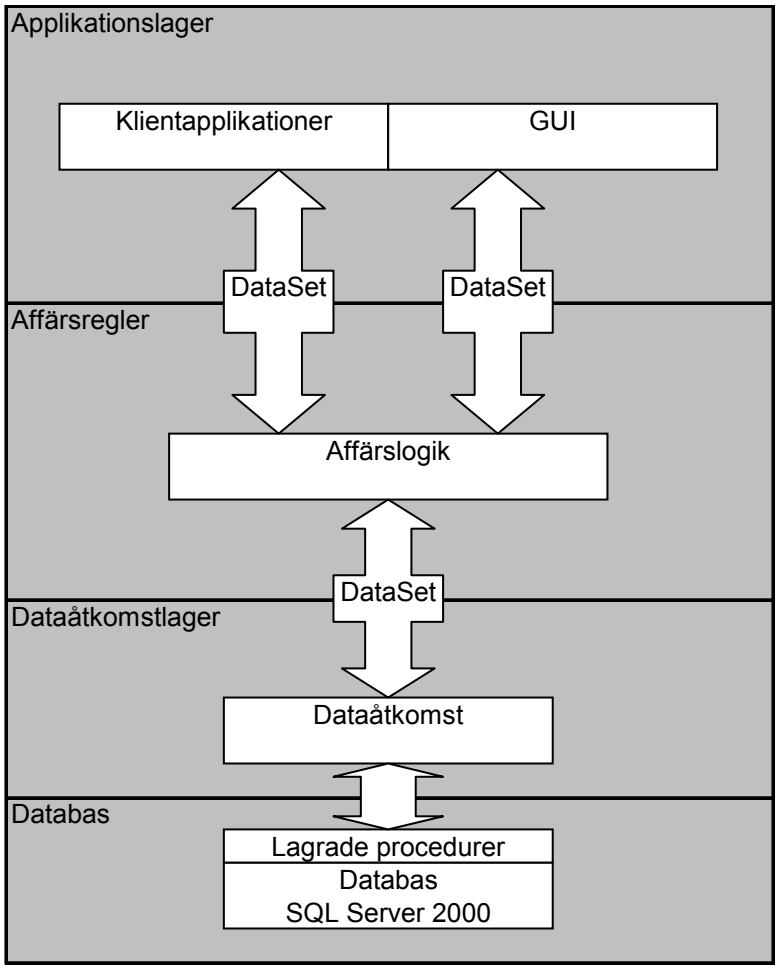
- T_Order – Ordrar.
- T_OrderLine – Orderrader för varje order.

- T_Item – Produkter.

Mellan T_Order och T_OrderLine finns en relation (se bilaga A) som innebär att det finns noll eller flera orderrader för varje enskild order. Det är detta som presenteras i DataGriden. Om användaren står med markören på en rad i DataGriden som presenterar ordrarna så visas automatiskt alla orderrader i DataGriden för orderrader. För att detta skall fungera så har vi specificerat i Datasetet att det finns en relation mellan dessa tabeller, precis som i databasen. Detta innebär att ett Dataset kan lagra data precis som den ser ur i databasen. Användaren kan ändra och lägga till data direkt i en DataGrid men vi har valt att inte tillåta det utan endast inmatning av värden i textboxar. Dessa textboxar är också kopplade till Datasetet och en ändring i en av dessa textboxar uppdaterar Datasetet vilket sedan avspeglas i DataGridarna. Att vi valt att inte tillåta redigering direkt i DataGridarna beror på att referensapplikationen inte heller tillåter det. I referensapplikationen finns en del funktionalitet som vi har valt att inte ha med i vår applikation. Bl.a. så kan användaren i referensapplikationen utföra tester på produkter för att se om de uppfyller vissa krav och detta görs mot tabeller som vi inte har med. Vi har valt att inte ha med den funktionaliteten i vår applikation därför att testerna är affärsspecifika och inverkar inte på utvärderingen av VS Dotnet.

10.3.1 Datatransport mellan lagren

I referensapplikationen skickas datan mellan lagren i form av XML-strängar för att underlätta dataåtkomst för externa program. Alternativet skulle vara att skicka datan i form av Recordset genom alla lagren för att förenkla programmeringen. Men då förloras det oberoende som XML erbjuder. Alltså så har Midroc Electro valt en lite besvärligare implementation för att kunna behålla ett visst oberoende. I Dotnet är detta problem löst genom de nya Dataseten. Eftersom att ett Dataset bygger på XML-standarder så finns alltid möjligheten för externa program att komma åt datan och databasen via affärslagret så länge de programmen känner till ett Datasets XML-struktur. Ett Dataset kan laddas med data från en DataAdapter eller med data direkt från en XML-sträng med rätt struktur. Vilket innebär att ett externt program kan gå in via affärslagret och uppdatera databasen via ett Dataset eller få data från ett Dataset i form av en XML-sträng. Om t.ex. ett robotstyrningsprogram som inte har ett gränssnitt mot databasen behöver koordinatdata kan den gå in via affärslagret och begära data i form av XML från ett Dataset. Som sagt kan ett Dataset laddas med data från en XML-sträng men XML-strängar kan också skapas direkt från ett Dataset för att sedan distribueras till externa program.



Figur 10-3: Skiktmodell alternativ lösning i VS Dotnet

Oberoendet mellan lagren (se Figur 10-3) innebär t.ex. att GUI:t inte skall eller behöver känna till hur datan är lagrad, om det är i en databas eller i en ren textfil. Det optimala vore om affärslagret kunde skapa ett Dataset som alltid ser likadant ut oavsett hur datan i databasen är lagrad. Tex. om tabellstrukturen ändras så skall inte GUI:t påverkas. Då behöves ingen omkompilering av GUI:t vid ändringar i databasen. I nuläget har vi i vår applikation inte det oberoendet utan en kompilering är nödvändig vid t.ex. databasändringar. Vi har valt att ha en ”hård referens” d.v.s att vi har en referens till den DLL-fil som inhyser dataåtkomstlagret i vårt GUI-lager. Detta innebär att vi har en referens till ett Dataset i GUI:t från dataåtkomstlagret som vi då kan koppla till våra grafiska komponenter i designläge i VS Dotnet. Detta hade inte varit möjligt att göra om vi inte hade haft denna hårda referens utan hade då varit tvungna att koppla våra komponenter under runtime. Att implementera lagren utan hårda referenser låg inte inom tidsramen för detta arbete.

10.3.2 Dataåtkomstlagret (DataAccess)

Dataåtkomstlagrets (DA) uppgift i applikationen är att sköta kommunikationen med databasen och utföra uppdateringar på densamma. För att göra detta möjligt så har vi använt oss av ADO Dotnet som tillåter oss att anropa lagrade procedurer som vi skapat direkt i databasen. I DA finns en funktion *daGetAllOrders(string filter)* som har till uppgift att utföra ett SelectCommand mot databasen med en filterparameter och sedan returnera ett Dataset tillbaka till affärslagret. Denna filterparameter kontrollerar om alla ordrar i databasen skall returneras eller bara de ordrar som har status AKTIV (ej avslutade). Detta för att oftast finns inget intresse av de ordrar som är avslutade (KLAR). Möjligheten att få se alla ordrar finns dock, men med risk att väldigt stora datamängder skickas mellan lagren. Proceduren *daUpdateOrders(Dataset changed_dataset)* som tar ett Dataset som inparameter har till uppgift att uppdatera databasen utifrån de ändringar som finns representerade i Datasetet. Detta sker genom anrop till DataAdaptrarnas funktion Update. Update väljer själv vilket Commandobjekt som skall utföras genom att jämföra data i Datasetet med data i databasen. Här finns möjligheter att fånga upp de undantag (exceptions) som eventuellt sker. Här bör inte felmeddelande visas i form av messageboxes då DA troligen ligger på en server utan ett gränssnitt mot en användare.

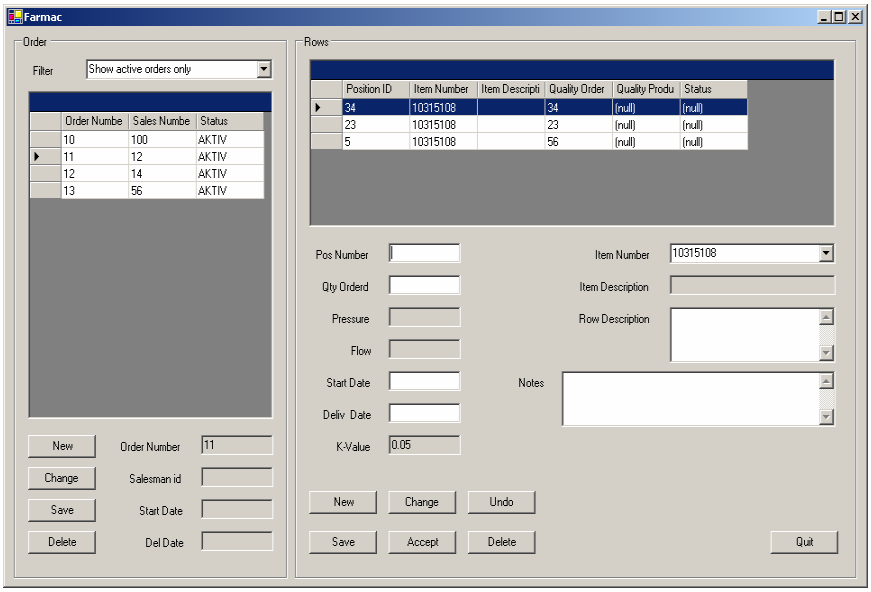
10.3.3 Affärslogiklagret (BusinessRules)

I affärslagret ligger normalt sett affärsspecifika regler t.ex. logiska operationer, matematiska beräkningar och kommunikation med externa program samt regler för hur fel eller undantag som genereras av systemet skall hanteras. I vårt affärslager har vi inga regler för programmet p.g.a. av det inte finns några regler att operera på. Som nämnts tidigare beror detta på att vi har valt att inte ha med affärsspecifika regler då dessa inte inverkar på resultatet av utvärderingen. Den enda uppgift affärslagret har är att vidarebefordra data till de andra lagren. Inkommande Dataset från GUI skickas till dataåtkomstlagret för att uppdatera databasen. Och inkommande Dataset från dataåtkomstlagret skickas till GUI:t för presentation.

10.3.4 Presentationlagret (GUI)

Presentationslagret är den del av programmet som används för att presentera data för en användare. Här finns möjligheten att lägga till, ta bort eller uppdatera data genom ett GUI. Efter att ändringar har skett från användarens sida så skickas Datasetet med uppdaterad data till affärslagret. Presentationen för användaren sker genom DataGrids som är en Windowskomponent med rutnät för presentation av data från ett Dataset samt genom

textboxar. Det är enkelt att koppla de grafiska komponenterna till ett Dataset då VS Dotnet medger denna möjlighet i det grafiska designläget.



Figur 10-4: Skärmbild applikationen i VS Dotnet

10.4 Problem och lösningar

Cachad data / synkronisering (concurrency)

Nya ADO Dotnets möjlighet att hantera cachad data tillsammans med Dataset objekt erbjuder betydande prestandahöjningar men introducerar samtidigt en ny typ av problem, nämligen synkroniseringsfel (concurrency errors). Synkroniseringsfel inträffar när DataAdaptern upptäcker att en den del av databasen som skall uppdateras inte stämmer överens med den bild av databasen som innehas av Datasetet. Detta är ett problem som uppstår när flera klienter jobbar mot samma databas. Varje klientapplikation som startas mot databasen skaffar sej omedelbart en bild av databasen och cachar den i internminnet i form av ett Dataset. Eftersom Datasetet innehåller mer än bara data såsom databasens relationer, regler och begränsningar så kan många operationer utföras mot Datasetet utan en massa onödig trafik mot databasservern. Datasetet kan modiferas på flera ställen innan det är dags för DataAdaptern som sammanfogar Datasetet och databasen att uppdatera databasen med de nya ändringarna. Synkroniseringsfel är något som vi träffade på vid flera tillfällen när vi implementerade vårt program. Det finns flera olika sätt att komma runt problemet med synkroniseringsfel, ett är att se till att det aldrig inträffar d.v.s. låta Datasetets data alltid skriva över det som står i databasen oavsett om där finns data som är uppdaterad efter att

Datasetet fylldes. Detta är en ganska brutal lösning som inte löser synkroniseringsproblemet utan mest undviker att problemet uppstår. Ett bättre sätt att ta hand om problemet är att använda DataAdapterns inbyggda undantagshanterare (exeption handler). Med hjälp av undantagshanteraren så kan systemet låta användaren avgöra på vilket sätt synkroniseringsproblemet skall lösas. Vanligtvis löses problemet på ett av följande sätt:

1. När en klientapplikation upptäcker (ett undantag inträffar) att den rad i databasen som skall uppdateras innehåller data som inte överensstämmer med den data som finns i Datasetet så hämtas all data i den aktuella dataraden upp till klientapplikationen. En dialogruta visas för användaren innehållande både den data som skulle uppdateras samt den data som lästes upp från databasen. Ingenting har hittills skrivits till databasen i den aktuella dataraden utan det är upp till användaren att bestämma vilken data som skall sparas i databasen.
2. Klientapplikationen hämtar upp den rad i databasen som skall uppdateras och jämför själv att all data stämmer innan updatekommandot ges till DataAdaptern för att undvika att ett undantag inträffar.
3. Ytterligare ett sätt att undvika synkroniseringsfel är att låta DataAdaptern fylla Datasetet med jämna mellanrum för att hålla det uppdaterat. Hur lång tid det skall vara mellan uppdateringarna beror helt på applikationens natur och arbetsbelastning.

Hanteringen av synkroniseringsfel kommer med största sannolikhet bli ett vanligt problem vid hantering av cachad data i framtiden då fler och fler klienter skall arbeta mot samma databas.

Tabell- och kolumnnamn genom lagren

För att förhindra att tabellnamnen från databasen ”lyser igenom” i lagerarkitekturen d.v.s att tabell- och kolumnnamn är hårdkodade i lagren, har Midroc Electro valt att definiera tabell- och kolumnnamn som konstanter i en extern fil som kompileras med i programmet. Om ett namnbyte på en tabell i databasen blir nödvändig så krävs dock en omkompilering av lagren. I Dotnet används Dataset som medger åtkomst till tabell eller tabellkolumn genom namnindexering eller dess nummerindex ungefär som åtkomst till ett visst element i en array. Vid användande av namnindexering så förloras ett visst oberoende genom att omkompilering av lagren blir nödvändig, i gengäld blir koden mer lättläst. Genom att indexera tabellerna med nummer vid åtkomst så blir omkompilering vid namnbyte i databasen inte nödvändig, till priset av mer svårläst kod. Lösningen i Dotnet för att slippa en omkompilering är att indexera

tabellerna med nummer. Med nummerindexering så krävs ingen ändring i programmet förutsatt att strukturen i databasen inte ändras utan endast namnen på tabeller eller kolumner.

För- och nackdelar med Dataset

En av fördelarna med att använda Dataset-objektet för att representera data från en datakälla är att läsning och sökning efter data går extremt fort eftersom Datasetet ligger laddat i klientens ramminne. En annan fördel är att Datasetet är XML-baserat och lämpar sig väl för att skickas över Internet. En nackdel med Dataseten är att de oftast fylls med väldigt mycket data för att slippa onödig kommunikation mot databasservern. Om Dataseten blir stora innebär det att mycket ramminne tas i anspråk hos klienten samt att fyllningen av stora Dataset kan bli tidsödande. För övrigt misstänkte vi att det fanns någon slags bug i nuvarande version av VS Dotnet som gjorde att fyllning och eventuell ihopslagning (merge) av ett Dataset till ett annat kunde ta extremt lång tid (upp till en halv minut). Denna bug fick vi sedan verifierad av Hussein Abuthuraya på Microsoft Developer Support [12].

GUI- komponenter

Ett av de problem vi träffade på var att DataGridarna som presenterar data ur Datasetet i presentationslagret var oerhört svårt att hantera med avseende på utseende och formatering. Grundinställt i DataGriden är att databasens tabellnamn visas i kolumnrubrikerna men detta är något som oftast inte är önskvärt utan programmeraren skall själv kunna välja namn som är mer lättförståliga. Vad som skulle kunna förväntas är att det finns någon typ av egenskapsruta som medger ändring av namnen på rubrikerna, men denna funktion är inte implementerad. Lösningen är att lägga till en s.k. *TableStyle* till DataGriden och som sin tur behöver en *GridColumnStyle* där kulumnnamnen kan ändras. Ett likartat problem uppstår vid borttagning av vissa kolumner för visning i DataGriden. Vi har hittat två lösningar på detta:

1. Att i en *ColumnStyle* sätta kolumnbredden på den kolumn som skall tas bort till noll.
2. Att i DataSetet ta bort kolumnmappningen för den kolumn som skall tas bort.

Ingen av dessa lösningar är vad som kan förväntas av en modern utvecklingsmiljö. De problem som vi nämnt ovan gäller endast vid utveckling Windowsapplikationer och inte webapplikationer. Webvarianten av DataGriden har stöd för att i designläge ändra dessa egenskaper. Varför Microsoft valt att inte ha detta stöd för Windowsapplikationer har vi inget svar på men av reaktionerna som vi har sett och läst på Internet så borde en uppdatering snart vara tillgänglig. Ett annat problem med DataGridarna var att de inte avspeglade datan i det kopplade Datasetet på ett tillfredsställande sätt. Vi trodde detta var ett programmeringsfel från

vår sida men visade sig vara en bug [12] i DataGrid-komponenten som uppstår när den används för att visa ett förälder-barn förhållande i ett Dataset. I samma bug finns också ett problem med minnesläckage vilket borde vara en ännu större anledning till att släppa en uppdatering av DataGriden.

11 Analys

Vi har valt att begränsa vår studie främst till databashanteringen ADO, detta p.g.a. av att VS Dotnet är allt för omfattande för att utvärdera under den tid som är avsatt för detta arbete. Det är främst skillnaderna mellan gamla ADO och nya ADO Dotnet som gör att applikationen kan implementeras på ett annorlunda sätt. Vi har valt att använda Visual Basic Dotnet som programspråk av den anledningen att det är detta språk Midroc Electro har använt för sina tidigare projekt och i vilket de kunde ge oss störst support. Vår handledare på Midroc Electro Leif Sounvieri har stått för expertis och har fungerat som bollplank för våra idéer samt tillhandahållit material. Efter att ha implementerat en alternativ lösning till Midroc referensapplikation med hjälp av VS Dotnet så kan nu de intressanta parametrarna från kapitel 3.4 diskuteras.

Mängden kod som manuellt måste produceras

Mängden manuellt producerad kod är klart mindre i och med användandet av ADO Dotnet. Största skillnaden ligger i att den manuella hanteringen av XML uteblir helt i ADO Dotnet samt att den kodgenerator som Midroc Electro tidigare använt nu helt kan ersättas av guider i VS Dotnet. Generellt sett så har införandet av Dataset bidragit till att den manuellt producerade kodmängden hålls nere. Konstruktionen av det grafiska gränssnittet i VS Dotnet skiljer sig inte nämnvärt från sin föregångare med avseende på kodmängd. Mycket av konstruktionsarbetet kan utföras i designläge såsom kopplingar mellan olika komponenter och Dataset. Enskilda komponenters uppförande och egenskaper under körning är dock föremål för manuellt skriven kod. Den GUI-komponent som var svarade för mest manuellt skriven kod var utan tvivel DataGriden då denna saknade vissa egenskaper som följaktligen fick skrivas in för hand, tex. så finns det inget lätt sätt att ändra kolumnrubriker eller att dölja kolumner. Kopplingar till datakällor kan göras via det grafiska designläget och definitioner av SQL-satser och lagrade procedurer likaså. I sin enklaste form utan flera skikt innebär att det går att skapa en koppling till en databas, skapa kommandon och presentera datan i ett GUI

med bara ett par rader manuellt skriven kod. Naturligtvis blir det mer skriven kod när en flerskiktad arkitektur används.

Kodens komplexitet

Med komplexitet avses hur svårt det är att sätta sig in i hanteringen av de nya objekten samt hur svår den genererade koden är att förstå och modifiera. Komplexiteten i den av VS Dotnet genererade koden anser vi är tämligen låg. Den manuellt producerad koden blir också mindre komplex p.g.a. av att den besvärliga XML-hanteringen som används av Midroc Electro i tidigare projekt inte längre är nödvändig. Koden för de mer avancerade komponenterna såsom DataAdapter och andra ADO Dotnet-komponenter är till stor del också lättläst och bra strukturerad. Om SQL-satserna ligger i koden kan den bli svår att överskåda med långa satser som är svåra att följa. Detta undviks dock vid användandet av lagrade procedurer där SQL-satserna inte blir synliga i koden utan ligger i databasen. Hur komplex en kod upplevs beror naturligtvis mycket på betraktarens kunskaper och erfarenheter, vilket innebär att vår bedömning kan skilja sig från bedömningar gjorda av andra.

Möjligheter till debuggning

Möjligheterna till normal debuggning i VS Dotnet skiljer sig inte nämnvärt från sin föregångare. Vid första anblicken saknade vi möjligheten till editering av koden under debuggning vilket VS 6.0 har stöd för. Det visade sig att denna möjlighet finns i VS Dotnet men är inte påslagen som standard för Visual Basic utan måste anges via menyvalet: *tools->options*. För C# projekt är dock detta valt som standard. Den stora nyheten i VS Dotnet är möjligheten till Remote Debugging på ett enkelt sätt. Remote Debugging ger möjlighet att debugga program som körs på en annan dator, tex. att ASP-sidor som körs på en webserver. Remote Debugging finns även VS 6.0 men är svår att använda. VS Dotnet har stöd för s.k. Cross Language Debugging vilket innebär att debuggern klarar av anropskedjor genom programmoduler skrivna i olika språk.

XML-stöd

Grundidén med Dotnet är att sammanlänka datorer och människor i distribuerade system och då behövs ett Internet- och brandväggsvänligt sätt att skicka data. XML lämpar sig mycket väl för denna uppgift och följaktligen har Microsoft valt att använda denna teknik i Dotnet. XML finns överallt i Dotnet, ADO Dotnets Dataset, konfigurationsfiler och mycket annat i Dotnet bygger på XML. Dotnet Framework innehåller två abstrakta klasser för XML-hantering.

XmlReader som tillhandahåller metoder och egenskaper för att ta emot och läsa XML. Och XmlWriter som tillhandahåller metoder för att producera XML-dokument. XmlReader och XmlWriter utgör ett alternativ till de klassiska DOM-objekten. ADO Dotnets Dataset bygger helt på XML men detta är dolt för programmeraren men kan plockas fram och användas på ett enkelt sätt. Ett Dataset kan fyllas direkt från ett XML-dokument samtidigt går det att extrahera ut datan från ett Dataset tillbaka till XML. Eftersom Dataseten baseras på XML så finns ingen behov längre att manuellt hantera XML i form av DOM-dokument utan hanteringen sköts implicit i bakgrunden. I C# finns möjlighet att skapa XML-dokumentation utifrån källkodsfiler som sedan kan presenteras i en webbläsare ungefär på samma sätt som Javas Javadoc. Av någon anledning finns inte detta stöd att generera XML-dokumentation i Visual Basic Dotnet vilket är synd då detta är ett effektivt sätt att dokumentera. Detta till trots har Dotnet ett utmärkt stöd för XML.

12 Slutsatser

För att besvara frågan om Midroc Electro skall övergå direkt till Dotnet för utveckling av programvara är svaret nej. Dotnet är ett utomordentligt verktyg men som tyvärr dras med en del barnsjukdomar och programfel. För att fullt ut kunna ta tillvara de fördelar som Dotnet erbjuder så krävs inte bara att system utvecklas i detta verktyg utan också att kunderna är villiga att uppgradera eller investera i system som är kompatibla med Dotnet. Vår rekommendation till Midroc Electro är att använda VS Dotnet och VS 6.0 parallellt under en tid. Vår uppfattning är att Midroc Electro bör fortsätta att utveckla större program i VS 6.0 och prova att utveckla mindre system i VS Dotnet i syfte att vänja sig vid verktyget, för att sedan helt övergå till Dotnet när en ny och mer beprövad version finns tillgänglig på marknaden. Enligt våra erfarenheter efter att ha gjort detta arbete kan vi säga att Dotnet starkt kan bidra till att effektivisera utveckling av flerskiktade system. Eftersom Dotnet och dess databasstöd är för stort att utvärdera helt under den begränsade tid vi har haft tillgänglig så finns en del kvar att studera. Framförallt bör transaktionshantering i Dotnet studeras eftersom detta är en väsentlig del av databashantering.

Referenser

Böcker

[1] *Riordan, M. Rebecca. (2002), Microsoft ADO Dotnet Steg för steg, Sunbyberg: Pagina förlags AB, ISBN: 91-636-0719-0*

[2] *Young, J. Michael (2000), Step by step XML, Redmond, USA: Microsoft Press, ISBN: 0-7356-1020-7*

[3] *Platt, S. David, (2001), Introducing Microsoft Dotnet, Redmond, USA: Microsoft Press, ISBN: 0-7356-1377-X*

[4] *Gray, Joe. Dalvi, Dinar. Joshi, Bipin. Normén, Fredrik. Norton, Francis. Olsen, Andy. Palermo IV, J Michael. Singh, Darshan. Slater, John. Williams, Kevin. (2001), XML for Dotnet Developers, Birmingham: Wrox Press, ISBN: 1-861005-31-8*

[5] *Nilsson, Jimmy (2002), Dotnet Enterprise Design with Visual Basic Dotnet and SQL Server 2000, Indiana, USA: Sams Publishing, ISBN: 0-672-32233-1*

Elektroniska

[6] *Cole, Thomas, Visual Basic Web Magazine, www.vdwm.com -> Jump from ADO to ADO Dotnet*

[7] *Matt Publishing, dnjonline, www.dnjonline.com -> Understanding the Dotnet Framework*

[8] *Matt Publishing, dnjonline, www.dnjonline.com -> Close up on Dotnet*

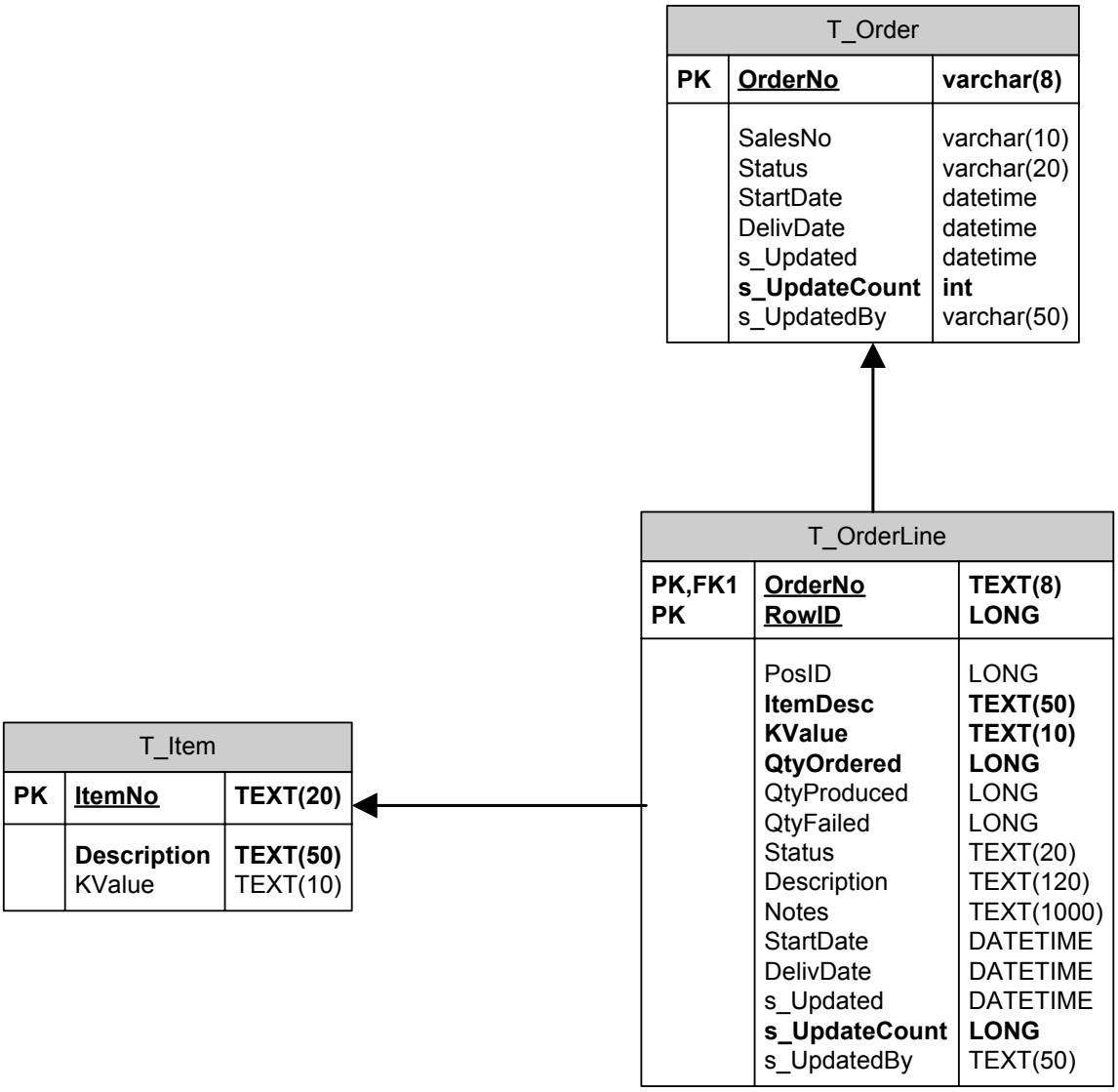
[9] *Microsoft Sverige, www.microsoft.com/sverige/thisis/terms -> Terminologi*

[10] *MSDN, Microsoft, www.msdn.microsoft.com*

[11] *Strawmyer, Mark, CodeGuru, www.codeguru.com/columns/DotNet -> Database Independent Data Access*

[12] *Abuthuraya, Hussein, Microsoft Developer Support, www.dotnet247.com/247reference/msgs/10/51621.aspx*

A Databasmodell referensapplikation



B Exempel på verktyg i utvecklingsmiljöerna

Programvara	Beskrivning
Visual Basic	Utvecklingssystem
Visual C++	Utvecklingssystem
Visual J++	Utvecklingssystem
Visual InterDev	Utvecklingssystem för Webbapplikationer
MSDN Library	Dokumentation
Visual SourceSafe	Versionhanteringssystem
Visual Database Tools	Verktyg för Databas modellering
Windows NT Option Pack	Transaction Server och IIS
Application Lifecycle Support	Komponent, design och databas hantering
Team Development Support	Stöd för att arbeta i grupp
BackOffice Server	SQL server, mm.
Visual FoxPro	Utvecklingssystem för databaser

Tabell C-1: Verktyg i VS 6.0

Programvara	Beskrivning
Visual Basic Dotnet	Utvecklingssystem
Visual C++ Dotnet	Utvecklingssystem
Visual C#	Utvecklingssystem
Mobile Web Forms	Skapa webbapplikationer för mobila enheter
Class Library Templates	Återanvändbara klassmallar.
Windows Control Library	Användarkontroller för Windows-formulär
Windows Services	Skapa Windowstjänster.
RAD-verktyg för servrar	Serveradministration händelseloggar etc.
Fjärrdebugging	Debuggning och webbtjänster över nätet
Crystal Reports	Rapportgenerator
Visual Database Tools	Verktyg för databas hantering.
Visual SourceSafe 6.0c	För versionshantering och teamarbete
Application Center Test	Testar funktionalitet och prestanda hos program
Visual Studio Analyzer	Hittar flaskhalsar i distribuerade COM.

Tabell C-2: Verktyg i VS Dotnet