Report Designer - Ett verktyg för att generera databasbaserade rapporter

John Edwards

2 juni 2007
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Thomas Johansson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Sammanfattning

Denna rapport beskriver hur Report Designer skapades. Report Designer är ett verktyg som kan användas för att på ett effektivt och användarvänligt sätt designa och generera rapporter utifrån en databas. Rapporten beskriver förutom det resulterande programmet den planering och arbetsprocess som utfallit i Report Designer. Författaren sammanför dessutom slutsatser och fallgropar som åskådliggjort sig under arbetets gång. Rapportverktyget är utvecklat i C#, ett objektorienterat språk som stödjer sig på .NET arkitekturen. Följaktligen ger föreliggande rapport, förutom en inblick i skapandet av ett rapportverktyg, en insyn i hur program kan implementeras i .NET. I rapporten finns också inkluderad djupgående analys av befintliga metoder ämnade åt att skydda programvara.

Report Designer - A tool aimed at generating database-based reports

Abstract

This report describes the making of Report Designer. Report Designer is a tool devised to design and generate database-based reports in an efficient and user-friendly manor. Beyond the resulting software this report describes the planning and work process that resulted in Report Designer. The author of the present paper summarises the conclusions and problems that became apparent during the work process. Report Designer was written in C#, an object-oriented language based on the .NET architecture. Thus this report provides insight on how software can be developed with .NET. This report also includes a deep analysis on existing methods aimed to secure software.

Innehåll

1 Inledning **1**

2 Problembeskrivning **3**
2.1 Problemspecifikation . . . 3
2.2 Riktlinjer . . . 4
2.3 Ändamål . . . 4
2.4 Metoder . . . 4
2.5 Relaterat arbete . . . 5

3 Tillvägagångssätt för att skydda källkod **7**
3.1 Skyddsmekanismer . . . 8
3.2 Intermediär kod . . . 8
3.2.1 MSIL . . . 9
3.3 Obfuskatorer . . . 9
3.4 Kryptering . . . 11
3.4.1 Bulkkryptering . . . 11
3.4.2 Partiell kryptering . . . 11
3.5 Hårdvarulösningar . . . 12
3.6 Sammanfattning och slutsatser . . . 12
3.6.1 Rekommenderad lösning för Report Designer . . . 13

4 Arbetets gång **15**
4.1 Förberedelse . . . 15
4.2 Specifikation . . . 16
4.3 Implementation . . . 16

5 Resultat **17**
5.1 Översikt . . . 17
5.2 Programtillstånd . . . 20
5.2.1 Starttillstånd . . . 20
5.2.2 PDF-inställningar . . . 21
5.2.3 Konfigurera databaskoppling . . . 22

5.2.4 Återskapning av rapport 23
5.2.5 Visning av tabell 23
5.2.6 Konfiguration av DSD 23
5.2.7 Import av DSD 27
5.2.8 Design av RD 27
5.2.9 Design av fråga 32
5.2.10 Design av graf 32
5.2.11 Generation av enstaka rapporter 34
5.2.12 Generation av rapporter 35
5.3 Tekniska aspekter 36
5.3.1 Klasser 36
5.4 Språkstöd 37

6 Slutsatser **39**
6.1 Begränsningar 39
6.2 Framtida arbete 39

7 Tack **41**

References **43**

Figurer

5.1 Bilden föreställer Report Designer. 19
5.2 Programmets starttillstånd. 20
5.3 PDF-inställningar som kan göras i programmet. 21
5.4 Konfiguration av databasen. 22
5.5 Gränssnitt för återskapning av rapport. 24
5.6 Gränssnitt för visning av tabell. 25
5.7 Konfiguration av DSD. 26
5.8 Import av DSD. 28
5.9 Designläge för RD. 29
5.10 Designläge för fråga. 31
5.11 Designläge för graf. 33
5.12 Gränssnitt för skapande av enstaka rapporter. 34
5.13 Gränssnitt för skapande av rapporter. 35
5.14 UML-diagram över de viktigaste klasserna samt några relaterade .NET-klasser och andra komponenter. 36
5.15 Bild av Language Editor. 38

Kapitel 1

Inledning

Denna rapport redogör för skapandet av prototypen Report Designer, ett rapportgenereringsverktyg. Programmet har skrivits på uppdrag av Wennström IT & Media. Prototypen är tänkt att ligga till grund för en färdig produkt som företaget planerar använda i ett större sammanhang. Wennström IT & Media är ett enmannaföretag som drivs av Andreas Wennström.

Kapitel 2 innefattar en problem beskrivning. Här beskrivs projektets mål samt de metoder som används under projektets gång. Olika metoder för att skydda programvara och källkod diskuteras i kapitel 3. I kapitel 4 redogörs för hur arbetet har gått. Här redogörs bland annat för de faser som projektet innefattat. Kapitel 5 redovisar arbetets resultat, det vill säga det program som implementerats. Slutligen presenteras slutsatser i kapitel 6.

Kapitel 2

Problembeskrivning

Syftet med projektet var att tillhandahålla ett verktyg som låter användaren designa rapportdefinitioner. Dessa rapportdefinitioner skall vara kopplade till en databas på ett sådant sätt att det är möjligt att generera rapporter utifrån en rapportdefinition. De genererade rapporternas innehåll varierar således med databasens innehåll samt parametrar som ges vid generationstillfället.

Sektion 1 specificerar projektets mål i detalj. I sektion 2 behandlas de riktlinjer som följts under arbetets gång. Sektion 3 redogör för projektets ändamål. De metoder som används under projektets gång redovisas i sektion 4. Slutligen diskuteras likartade arbeten i sektion 5.

2.1 Problemspecifikation

Antag att man har en databas som innehåller information relaterad till ett antal klienter. Exempel på typiska klienter är företag, personer och länder. Vidare vill man skapa flera rapporter baserade på dessa klienter. Om rapporternas upplägg är lika kan det vara önskvärt att skriva en metarapport (här kallad rapportdefinition) och sedan generera rapporter utifrån denna metarapport; en för varje klient. Att tillhandahålla ett program som låter användaren designa dessa rapportdefinitioner och generera rapporter baserade på dem, beroende på val av klient, är syftet med detta projekt.

Exempelvis skulle det kunna handla om en databas som innehåller information om anställda i ett företag. Varje anställd kopplas ihop med personuppgifter, arbetade timmar, lön, klagomål och diverse kommentarer från ledningen. Programmet skulle här kunna användas för att designa en rapportdefinition och sedan skapa en rapport per anställd; baserad på dennes sammankopplade data.

Följande är en sammanfattning av produktkriterierna som innefattades i den ursprungliga specifikationen:

- Programmet skall skrivas i C#.
- Alla rapportdefinitioner och inställningar skall lagras i en SQL-databas.
- Rapportdefinitionernas utseende skall kunna definieras på ett användarvänligt sätt i en grafisk editor.
- Inställningar för varje generationstillfälle skall sparas i en databastabell; utifrån detta kan samma mängd rapporter skapas vid senare tillfälle.

- Användaren skall kunna generera en enstaka rapport baserad på en klient.
- Användaren skall kunna generera flera rapporter baserade på en mängd klienter. Denna mängd skall kunna framtas genom att ställa en SQL-fråga till databasen.
- Användaren skall kunna schemalägga skapandes av rapporter. Exempelvis så att en viss mängd rapporter skapas varje månad.
- Rapporter skall automatiskt kunna lagras på en webbserver.
- Rapporter skall automatiskt kunna skrivas ut tillsammans med ett kuvert för postbefordran.
- Rapporter skall kunna skapas i HTML- och PDF-format.
- Enkätsvar, grafer och tabeller skall kunna ingå i rapportdefinitionerna.
- Stöd för flera språk skall finnas.
- Man skall kunna välja att endast generera rapporter för ett särskilt tidsintervall.

Då projektet är tidsbegränsat är det inte nödvändigt att alla dessa kriterier uppfylls.

2.2 Riktlinjer

Det är viktigt att notera att det är en prototyp som skall skapas, inte en slutgiltig produkt. Därför har fokus lagts på att programmet skall ha många funktioner som inte alltid är finslipade; hellre än att ha få finslipade funktioner.

Även om prototypen i första hand är tänkt att fungera som en komponent i ett specifikt webbprojekt skall den kunna fungera självständigt; samt att olika delar av programmet skall kunna användas i andra applikationer. Därför är det viktigt att programmet är modulärt. Modulariteten är också viktigt då flera färdiga komponenter kommer att användas i programmet. Det skall snabbt gå att byta ut dessa om så skulle önskas.

Då programmet är tänkt att användas av personer som inte besitter djupare datakunskap har användarvänlighet hög prioritet. Det skall också vara smidigt att snabbt växla mellan olika delar av programmet. Dialogrutor skall därför undvikas i så stor utsträckning som möjligt.

Vad gäller språkstöd är det viktigt att det är enkelt att lägga till stöd för nya språk. Modifikation av källkoden skall ej krävas.

2.3 Ändamål

Programmet skall främst användas som en komponent i ett specifikt webbprojekt. Eventuellt skall det också kunna användas i andra sammanhang.

2.4 Metoder

Programmet har utvecklats i Visual Studio 2005 [8]; källkoden är skriven i C#. Utvecklingen har skett mot en MySQL-databas som tillhandahållits av Wennström IT & Media. För kommunikation med MySQL-databasen har MySQL Connector använts [6]. Vissa färdiga komponenter har använts i programmet; dessa beskrivs i senare kapitel.

2.5 Relaterat arbete

Flera applikationer har utvecklats i syfte att generera rapporter utifrån MySQL databaser. Några av dessa är PHP Report Maker [4], Documalis Free Reporter [5] och Smart Report Maker [7]. Det finns dock många fler.

Kapitel 3

Tillvägagångssätt för att skydda källkod

All kommersiell mjukvara är potentiella mål för piratkopiering. Programvara som inte alls är skyddad kan enkelt spridas vidare från en legitim köpare till någon annan. Problemet har egentligen funnits så länge som kommersiell mjukvara har funnits, men det har fått en annan magnitud i och med Internet.

Ett vanligt tillvägagångssätt för att skydda programvara är att man kräver att användaren uppger en programnyckel. Då användaren uppger en nyckel jämförs denna med den korrekta. Exakt hur den korrekta nyckeln skapas kan variera. Om nyckeln är korrekt tillåts användaren använda mjukvaran, om nyckeln inte är korrekt tillåts användaren inte använda mjukvaran. Detta realiseras ofta som ett villkorat hopp i maskinkoden. En nackdel med detta tillvägagångssätt är att det räcker med att göra en ändring i detta villkorade hopp för att eliminera säkerhetsspärren [10]. För att hitta de säkerhetskritiska sektionerna i ett program kan debuggers och dekompilatorer användas. Dekompilatorer är program som omvandlar maskinkod till mera läsbar kod så att den blir lättare att analysera. Några exempel på populära dekompilatorer är w32dasm och IDAPro. Debuggers är program som låter användaren följa exekveringen av programmet; något som underlättar analysen. Några exempel på populära debuggers är TRW2000 och SoftICE [12].

När det gäller mjukvara skriven i .NET är det ännu enklare att ändra i programmet. Anledningen till detta är att .NET program inte kompileras till maskinkod utan till intermediär kod [18]. Den intermediära koden är väldigt lättläst och kan enkelt konverteras tillbaks till källkod [11].

Detta kapitel syftar till att redogöra för några av de metoder som finns för att skydda mjukvara. Vidare rekommenderas en lämpig teknik att användas tillsammans med Report Designer. I sektion 1 ges en kortfattad överblick. Sektion 2 klargör begreppet intermediär kod samt går in djupare på MSIL-kod. I sektion 3 presenteras obfuskatorer; här redogörs också för ett test av en obfuskator. Sektion 4 sammanfattar olika metoder som relaterar till kryptering av koden. I sektion 5 beskrivs hårdvarulösningar. Sektion 6 sammanfattar och jämför kapitel 3, 4 och 5.

3.1 Skyddsmekanismer

Det har gjorts många försök att skapa lösningar som gör det omöjligt för användare att komma runt säkerhetsspärrar. Här sammanfattas några av dessa.

Obfuskatorer Obfuskatorer är program som omvandlar kod till en annan ekvivalent kod som är svårare att omvandla till tillbaks till lättläst kod. Ekvivalent ska här förstås som att programmen gör ”till synes” samma sak; de behöver inte beräkna samma sak eller utföra beräkningar på samma sätt. Med kod kan här menas intermediär kod, maskinkod och eventuellt källkod.

Extern exekvering (Server-Side Execution) Detta begrepp handlar om att exekvera delar av programmet på en server istället för lokalt. På detta sätt får användaren aldrig tillgång till programmet och kan därför aldrig omvandla det till källkod eller eliminera säkerhetsmekanismer. Problemet med denna lösning är att den kräver att användaren ska ha en (snabb) Internetuppkoppling. All kommunikation gör dessutom applikationen mycket långsammare än en ekvivalent lokal variant. Extern exekvering används bland annat i visa typer av onlinespel där alla spelare interagerar med samma virtuella värld. Här sköts många av beräkningarna som relaterar till världen på servern medan några sköts av klienterna.

Hårdvarulösningar Det finns flera förslag till hårdvarulösningar som gör det omöjligt att via mjukvara omvandla program till källkod eller komma förbi säkerhetsspärrar [10][13]. Hårdvarulösningar kräver dock i praktiken att folk samarbetar och kommer överens om en gemensam, generisk lösning. Det blir kostsamt att använda olika processorer för olika applikationer.

Kryptering Det är möjligt att kryptera hela eller delar av program och sedan dekryptera dessa delar under exekveringen [14]. Dessa lösningar innebär en del extra arbete då kod måste dekrypteras (och i vissa fall krypteras). Dessutom återskapar vissa av dessa lösningar hela programmet i minnet; i vilket fall en kopia kan skapas i klartext genom att avläsa minnet.

3.2 Intermediär kod

Problemet med maskinkod är att den endast kan exekveras på en typ av processor. För att lösa detta problem har det uppkommit flera kompilatorer som kompilerar källkoden till intermediär kod istället för maskinkod. Intermediär kod är ett slags mellansteg mellan källkod och maskinkod. Olika typer av intermediära språk har funnits länge; men dessa språk har implicit konverterats vidare i kompilatorn till någon form av maskinkod. Processorer kan inte tolka maskinkod direkt. Man kan endera använda en interpretator som gör om den intermediära koden till maskinkod under själva exekveringen. Eller så kan man använda en JIT-kompilator (just-in-time) som gör om all intermediär kod till maskinkod och sedan exekverar den. JIT-kompilatorer är ofta att föredra då de låter hela programmet exekvera fritt utan interpretation. Således går det mycket snabbare [15].

Javas bytecode och .NETs MSIL är två exempel på intermediära språk. Då Report Designer är skrivet i .NET är MSIL-kod att föredra för vidare analys.

3.2.1 MSIL

Allt som skrivs i .NET kompileras till så kalla MSIL-kod (Microsoft Intermediate Language). När användaren kör programmet används CLR (Common Language Runtime) för att JIT-kompilera koden till den aktuella arkitekturen så att den kan exekveras.

Här följer ett exempel på hur C#-kod ser ut efter att ha konverterats till MSIL-kod.

C#:

```
public void Helloworld()
        {
            Console.WriteLine("Hello World!");
        }
```

MSIL:

```
.method public hidebysig instance void  Helloworld() cil managed
{
  // Code size       13 (0xd)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  ldstr      "Hello World!"
  IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000b:  nop
  IL_000c:  ret
} // end of method HelloWorld::Helloworld
```

Som man kan se är det inte alldeles för svårt att gissa sig till den ursprungliga koden om man känner MSIL-koden. Dessutom kan man konvertera MSIL-koden tillbaks till C#-kod. Detta kan exempelvis göras med gratisprogrammet Reflector. Om man använder Reflector för att konvertera MSIL-koden ovan till C# får man exakt den överstående C# koden.

3.3 Obfuskatorer

Obfuskatorer är program som omvandlar kod till en annan ekvivalent kod som är svårare att omvandla tillbaka till källkod. Det finns både obfuskatorer för maskinkod och intermediär kod [11] [19]. Här behandlas dock obfuskatorer i kontexten av intermediär kod.

Begreppet ”Obfuskator Transformation” kan användas för att klargöra vilka modifikationer på koden som obfuskerar den och vilka som inte gör det [11]. $P \overset{T}{\mapsto} P'$ är en obfuskator transformation om följande gäller:

- T tranformerar programmet P till programmet P'.
- P och P' har samma upplevda beteende; förutom när det gäller ickedeterminism och programfel. Med ”upplevda beteende” menas här att vi inte märker av någon funktionell skillnad när vi använder programmet. P' får ha sidoeffekter så som att skapa filer och skicka meddelanden över Internet.
- P' är mer obskyr än P.

Här följer några obfuskation transformationer [11]:

Modifikation av klasstrukturen Införsel av nya redundanta klasser; samt dela upp klasser eller sammanslå klasser som inte har någonting gemensamt.

Modifikation av arrayer Sammanslagning av arrayer; samt plocka isär arrayer, öka arrayens dimension eller minska arrayens dimension.

Byta namn på objekt Namnbyte av klasser, variabler, funktioner, namnrymder med mera.

Införa skräpkod Införsel av redundant kod; exempelvis kan man göra följande transformation:

Ursprunglig kod:

```
if x = 2 then c()
```

Transformerad kod:

```
int z = 0;
if x = z^2 + 2-1*z then if z = 0 then c() else return false
```

Interpretation Omskrivning av delar av programmet till ett annat språk som interpreteras av programmet självt.

Modifikation av strängar Omvandling av strängar till andra objekt som måste återställas innan programmet kan tolka dem som strängar. Dessa transformationer är väldigt användbara eftersom det ofta går att söka efter en viss sträng för att hitta en önskvärd del av programmet. (Exempelvis kan man söka efter ”Expiration Date” (eller liknande) i vissa program för att hitta jämförelsen där programmet avgör om man inte får använda programmet längre eller ej.)

Test har gjorts av obfuskatorn Dotfuskator som följer med Microsoft Visual Studio .NET 2005. Följande funktion användes för att testa programmet:

```
public static int Fibonacci(int n)
        {
            int previous = -1;
            int result = 1;
            for (int i = 0; i <= n; ++i)
            {
                int sum = result + previous;
                previous = result;
                result = sum;
            }
            return result;
        }
```

Koden kompilerades och Dotfuskator applicerades på MSIL-koden. Så här tolkas den obfuskerade koden av Reflector:

```
public static int a(int A_0)
{
    int num = -1;
```

```
    int num2 = 1;
    for (int i = 0; i <= A_0; i++)
    {
        int num4 = num2 + num;
        num = num2;
        num2 = num4;
    }
    return num2;
}
```

Dotfuskator skapade också en del andra klasser som inte fyllde någon funktion. Tyvärr är det inte så svårt att se vad den obfuskerade funktionen gör om man lägger ner lite tid.

3.4 Kryptering

Det finns metoder för att kryptera både intermediär kod och maskinkod. Då krypteringsalgoritmerna behandlar både intermediär kod och maskinkod som ren data existerar det inga konceptuella skillnader mellan hur dessa medier behandlas [14].

3.4.1 Bulkkryptering

Om det existerar en enskild funktion som dekrypterar hela programmet kallas tillvägagångssättet för bulkkryptering. I detta fall är allt i programmet utan en inledande rutin för dekryptering krypterat. När programmet exekveras använder den inledande rutinen en nyckel för dekryptera det återstående programmet och exekvera detta. Denna nyckel kan ligga gömd i den inledande rutinen eller hämtas från någon annan stans (exempelvis genom att användaren skriver in den). Bulkkryptering är i någon mån analogt med JIT-kompilering i kompileringssammanhang [14].

Även om bulkkryptering är en metod som är enkel att implementera och inte kräver mycket extra arbete är den inte särskilt användbar. Detta beror på att programmet kommer att existera i klartext i primärminnet. Således kan användaren lagra en icke krypterad version av programmet utifrån minnet. Denna icke krypterade version kan sedan analyseras eller ändras [14].

3.4.2 Partiell kryptering

Med partiell kryptering dekrypteras bit för bit av programmet när det behövs. Det finns olika grader av finkornighet för denna metod; från större program delar till att enskilda funktioner dekrypteras. Partiell kryptering är i någon mån analogt med interpretering i kompileringssammanhang [14].

Fördelen med partiell kryptering jämfört med bulk kryptering är att man aldrig exponerar hela programmet i minnet. En användare kan alltså inte skapa en klartext kopia av hela programmet på en gång. Nackdelen är att det krävs mer extra arbete då dekrypteringsfunktionen måste anropas flera gånger [14].

3.5 Hårdvarulösningar

Det har föreslagits flera hårdvarulösningar för att lösa problem med piratkopiering och reverse engineering. Två exempel är:

Dongles Dongles är hårdvara som skall vara kopplade till datorn när ett program exekveras (exempelvis via en USB-port). Då programmet exekverar undersöker det om donglen är ansluten till datorn och tillåter endast användaren att använda programmet om så är fallet. Dongles är alltså produkt-specifika och tillhandahålls tillsammans med produkten. Problemet med dongles är att det är relativt enkelt att modifiera programmet så att det inte längre kräver att dongeln är ansluten till datorn när programmet exekveras. [10].

Smartcards Smartcards är kort som används för autentisering i olika sammanhang. Det kan exempelvis röra sig om bankautomater, betal-TV och kommunaltrafik. Dock brukar de inte användas i kombination med datorprogram. [16].

Ett mer omfattande förslag ämnat åt att skydda programvara med hårdvara är med så kallade säkra processorer (Tamper-Resistant processors, secure processors) [17]. Många förslag till exakt hur dessa skall designas finns men många av riktlinjerna är de samma. De ska skydda programvaran mot att användare läser koden (genom att dekompilera den), modifierar koden samt kopierar den. Detta realiseras genom att programmet är krypterat, både på hårddisken och i primärminnet.

I en sådan arkitektur litar man bara på den säkra processorn. Processorn har en publik och en privat nyckel. Den privata nyckeln är inbränd i processorn och avslöjas aldrig för någon. All skyddad programvara är krypterad med nycklar som bestäms av upphovsmännen. Dessa nycklar krypteras dock med den säkra processorns publika nyckel så att bara den säkra processorn kan komma åt programmets innehåll.

När nu processorn tar emot cachelinor från primärminnet är dessa som tidigare nämnt krypterade med upphovsmännens nycklar. Den säkra processorn kan dock dekryptera dessa cachelinor eftersom den kan dekryptera programmets egna nycklar med hjälp av sin privata nyckel. (Upphovsmännens nycklar var ju krypterade med den säkra processorns publika nyckel.) Den säkra processorn kan nu exekvera den dekrypterade cachelinan som vanligt. När resultat ska skrivas till primärminnet krypteras det igen. Någon form av integritetskontroll utförs också när cachelinan är dekrypterad; denna kan exempelvis innefatta hash- eller MAC-träd.

Hur exakt man använder den säkra processorns publika nyckel för att kryptera det skyddade programmets nycklar nämns dock inte av Shi et al.[17].

3.6 Sammanfattning och slutsatser

I denna översikt har vi klargjort begreppet intermediär kod samt presenterat ett antal alternativ för att skydda dels intermediär kod, dels maskinkod.

Som framgår av Dathathri [12] gäller att så länge som användaren har tillgång till hela programmet så finns det alltid ett sätt att ta bort alla säkerhetsspärrar så vida man inte tillämpar en hårdvarulösning. Dock finns det inget som hindrar att man förlänger tiden det tar att bryta dessa spärrar. Detta kan göras med obfuskatorer, kryptering och diverse kombinationer av dessa [10].

Vad beträffar en hårdvarulösning verkar det inte finnas några teoretiska hinder. Dock är detta mer av en global lösning som givetvis inte tillämpas på Report Designer.

3.6.1 Rekommenderad lösning för Report Designer

Följande frågor har ställts för att hitta en passande lösning:

1. Vilka kommer att använda programmet? Vilken är målgruppen?
2. Hur många kommer att använda programmet?
3. Är säkerheten viktig i andra aspekter än piratkopiering?
4. Hur mycket får säkerheten kosta?

Följade antaganden har gjorts angående Report Designer; dessa antaganden är baserade på program specifikationen och övrig kontakt med uppdragsgivaren:

1. Programmet kommer främst att användas av företag.
2. Det är svårt att avgöra hur populärt programmet kommer att bli; men då det troligtvis kommer att i första hand användas på företagsnivå och inte individnivå så kommer det att användas av betydligt färre än ett program som ämnas för privatpersoner av samma popularitet.
3. Programmet genererar rapporter utifrån en databas. Programmet kommunicerar med databaser via vanliga SQL-frågor och kan således tillfoga lika mycket skada på databasen som SQL-frågor skickade från godtyckliga program. Därmed ser jag inte någon anledning att lägga större vikt på säkerhet än vad beträffar piratkopiering.
4. Säkerhetens kostnad skall minimeras.

Baserat på detta verkar Report Designer löper små risker att ta alvarlig skada av piratkopiering. Detta följer av att:

- Företag kan inte tillgodogöra sig piratkopierad mjukvara lika säkert som privatpersoner. Detta då anställda på företag är mer benägna att anmäla företagets lagbrott än privatpersoner är att anmäla sig själva. Vidare har företag generellt sätt mer pengar än privatpersoner; därför tar de lite lika stor ekonomisk skada av att köpa mjukvara än privatpersoner.
- Då programmet främst tros komma att användas av företag finns det inget incentiv för privatpersoner att sprida det till andra privatpersoner.

Därmed rekommenderas att en vanlig obfuskator används för att skydda programmet. Det är ett billigt alternativ som gör mycket för att fördröja ett eventuellt attackförsök. Dotfuscator testades på Report Designer utan någon märkbar användarmässig förändring.

Kapitel 4

Arbetets gång

Arbetet som resulterade i programmet Report Designer har utförts på heltid under vårterminen 2007. Arbetsprocessen kan delas in i tre faser; förberedelse, specifikation och implementation. Dessa tre faser redovisas i kronologisk ordning under sektion 1, 2 och 3.

4.1 Förberedelse

Den förberedande fasen har innefattat följande sysselsättningar:

Förståelse av problemet Detta innebär diskussion med uppdragsgivaren samt tolkning av företagets program specifikation.

Införskaffning av programvara Den programvara som införskaffats redovisas i 2.4.

Grundläggande beslut samt val av komponenter För att skapa en approximativ bild av hur programmet skulle fungera implementerades testversioner av vissa kritiska delar av programmet. Dessa kritiska delar inkluderar rapportdesignläget. Det var viktigt att förstå exakt hur rapportdefinitioner skulle designas. Vilket format rapportdefinitionerna är designade i är avgörande för hur de skall konverteras till HTML- och PDF-format.

Slutprodukten skall kunna skapa PDF-filer med grafer i. Det stod tidigt klart att det varken fanns tid att skriva en komponent som konverterar data till PDF-format eller ett grafbibliotek. Därför har ett grafbibliotek samt en PDF omvandlare använts i programmet. Dessa komponenter beskrivs mera utförligt i ett senare kapitel.

Det beslutades att .NET komponenten WebBrowser skulle användas vid design av rapportdefinitioner. Denna komponent implementerar en komplett webbläsare som kan användas i designläge. I designläge kan man modifiera HTML-objekt i webbläsaren samt skriva in text. Att använda en webbläsare har fördelen att rapportdefinitionerna designas i ett välkänt format. Därmed kan en färdig komponent användas för att omvandla HTML-koden till PDF-format. Att generera rapporter i HTML-format blir trivialt. Webbläsaren är dock inte problemfri. Komponenten beskrivs mera utförligt i ett senare kapitel.

Efter denna fas fanns en förståelse för vad syftet med programmet var samt en ungefärlig bild av hur det skulle implementeras. Det stod klart vilka komponenter som

skulle användas i programmet samt hur dessa skulle fungera tillsammans. Konkret skrevs en del kod som senare skulle komma att användas i den fullständiga versionen. Denna fas upptog ungefär en fjärdedel av den totala arbetstiden.

4.2 Specifikation

När det fanns en approximativ bild av hur den slutgiltiga prototypen skulle te sig skrevs en utförlig specifikation av hur programmet skulle fungera. I denna specifikation var det grafiska användargränssnittet i fokus. Mallar för hur fönster och komponenter skulle se ut skapades i Visual Studio. Förutom det grafiska gjordes beslut om hur och var olika data skulle lagras. Arbetsprocessen såg ut som följer:

1. Specifikationen skrivs/ändras.
2. Uppdragsgivaren läser specifikationen.
3. Ändringar diskuteras.
4. Om ändringar skall göras; repetera från 1.

Efter denna fas fanns en detaljerad specifikation som beskrev vad som skulle implementeras. Denna fas upptog ungefär en fjärdedel av den totala arbetstiden.

4.3 Implementation

När det fanns en detaljerad specifikation implementerades den. Drygt 10 000 rader kod skrevs uppdelade på två exekverbara program samt ett klassbibliotek. Efter denna fas fanns en färdig prototyp; denna diskuteras i detalj i senare kapitel. Denna fas upptog ungefär hälften av den totala arbetstiden.

Kapitel 5

Resultat

Projektet har resulterat i programmet Report Designer. Sektion 1 ger en översikt. I sektion två redovisas programmets olika tillstånd. Tekniska aspekter diskuteras slutligen i sektion 3.

5.1 Översikt

Report Designer är skrivet i C# och kan exekveras på Windows-klienter med .NET Framework 2.0 (eller högre) och Acrobat Reader [2] installerat. Programmet tillhandahåller möjligheten att generera rapporter baserade på rapportdefinitioner samt en databas. Det är viktigt att klargöra följande termer innan programmet kan redovisas:

Klient När en rapport skall genereras väljs en klient som rapporten skall baseras på. Varje klient har ett ID-nummer som matchas mot alla övriga ID-märkta data i databasen. På så sätt kan Report Designer generera rapporter utifrån endast de data som är relevant för en specifik klient. Databasens klienter lagras i en klienttabell; klienter kan exempelvis vara företag, privatpersoner, musikalster eller bilmärken.

Databaskoppling En databaskoppling är ett objekt som innehåller all information som behövs för att Report Designer skall kunna ansluta sig till en specifik databas. Exempelvis innefattas information om server, användarnamn, lösenord och port nummer. Alla databaskopplingar som skapas av Report Designer lagras lokalt i en XML-fil.

Databas Struktur Definition (DSD) En DSD är ett objekt som beskriver hur Report Designer skall tolka informationen i en databas. Dessa objekt lagras i den aktuella databasen. I en DSD tilldelas varje tabell en tabelltyp och varje fält (i varje tabell) en fälttyp. Denna tilldelning görs av användaren. Följande tabelltyper kan väljas:

Client Innebär att tabellen innehåller data om klienter.

Ignored Innebär att tabellen skall ignoreras och ej visas i listor.

Questionnaire Innebär att tabellen definierar enkäter.

QuestionnaireQuestions Innebär att tabellen innehåller enkätfrågor.

QuestionnaireAnswers Innebär att tabellen innehåller enkätsvar.

Other / Unknown Innebär att tabellen är av en annan typ än överstående.

Vilka fälttyper som kan väljas beror på vilken tabelltyp som är vald för fältets tabell. Om tabelltypen exempelvis är ”Client” kan fälttyper som indikerar klienternas namn, adress och liknande väljas. DSD:er är nödvändiga då Report Designer inte kan veta vad data i databasen syftar på.

Rapportdefinition (RD) RD:er kallas de dokument som kan designas i Report Designer. De är metarapporter som innehåller kopplingar till databasen. Alla RD:er måste associeras med en DSD och lagras i databasen.

Massrapportdefinitioner (BRD) En BRD är en SQL-fråga som skall användas till att extrahera klienter från databasen. BRD:er lagras i databasen.

Loggade rapporter Varje gång användaren genererar en eller flera rapporter lagras en loggad rapport i databasen. En loggad rapport innehåller information som tillåter Report Designer att återskapa samma mängd rapporter som genererades vid det ursprungliga tillfället. Om informationen i databasen har ändrats sedan det ursprungliga tillfället kommer dessa förändringar att reflekteras i de rapporter som återskapas av en loggad rapport.

Programtillstånd Report Designer är alltid i ett av flera programtillstånd, även kallade tillstånd. Programtillstånden beskrivs i sektion 2. Varje programtillstånd har ett huvudfönster, en knapprad, ett hjälpfönster samt en rubrik. De flesta programtillstånd svarar också mot en viss typ av nod i strukturträdet. Termerna som beskriver användargränssnittet beskrivs nedan.

Report Designers användargränssnitt 5.1 består av sju delar, dessa är märkta A-G:

Huvudmenyn (A) Huvudmenyn innehåller två menyflikar; Arkiv och Språk. Under Arkiv kan endast alternativet Avsluta väljas; detta alternativ avslutar programmet. Under språk finns ett alternativ för varje språk som finns installerat i Report Designer. Dessa alternativ aktiverar respektive språk.

Strukturträdet (B) Strukturträdet tillhandahåller en strukturerad översikt över de aktiva komponenterna i programmet. Genom att klicka på olika noder i trädet kan man aktivera olika programtillstånd. Exempelvis kan man klicka på en databaskoppling för att ansluta sig till respektive databas.

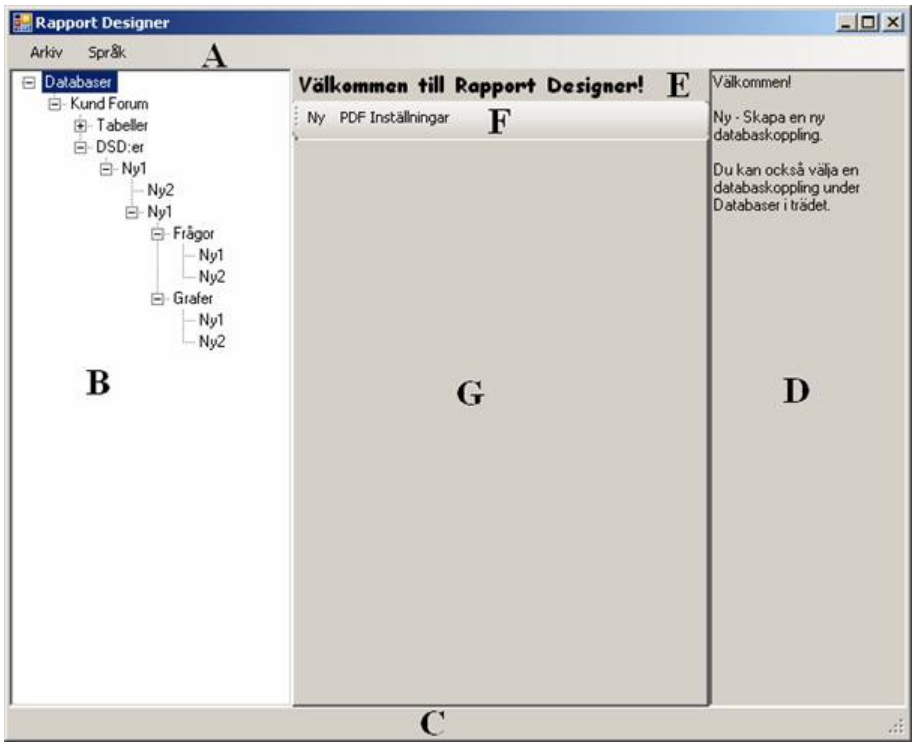
Statusraden (C) Statusraden visar för närvarande ingen information.

Hjälpfönstret (D) Hjälpfönstret beskriver det aktuella programläget och redogör för användarens möjligheter.

Rubriken (E) Varje programtillstånd har en rubrik, dess innehåll kan variera. Om det aktuella programtillståndet är associerat med ett dokument inkluderas dokumentets namn i rubriken.

Knapprad (F) På knappraden visas knappar som ger användaren möjligheter som är specifika för det aktuella programtillståndet.

Huvudfönster (G) Huvudfönstrets innehåll varierar med programmets tillstånd. Då användaren designar en rapportdefinition sker detta exempelvis i huvudfönstret.



Figur 5.1: Bilden föreställer Report Designer.

Figur 5.2: Programmets starttillstånd.

5.2 Programtillstånd

Denna sektion redogör för de olika programtillstånd som Report Designer kan befinna sig i. Varje programtillstånd beskrivs tillsammans med en lista över användarens valmöjligheter. I alla tillstånd kan användaren byta språk och avluta programmet. Användaren kan också i alla programtillstånd markera en nod i strukturträdet för att aktivera ett annat tillstånd. Eftersom att dessa möjligheter är generiska beskrivs de inte i varje undersektion. Triviala programtillstånd har utelämnats i detta kapitel.

5.2.1 Starttillstånd

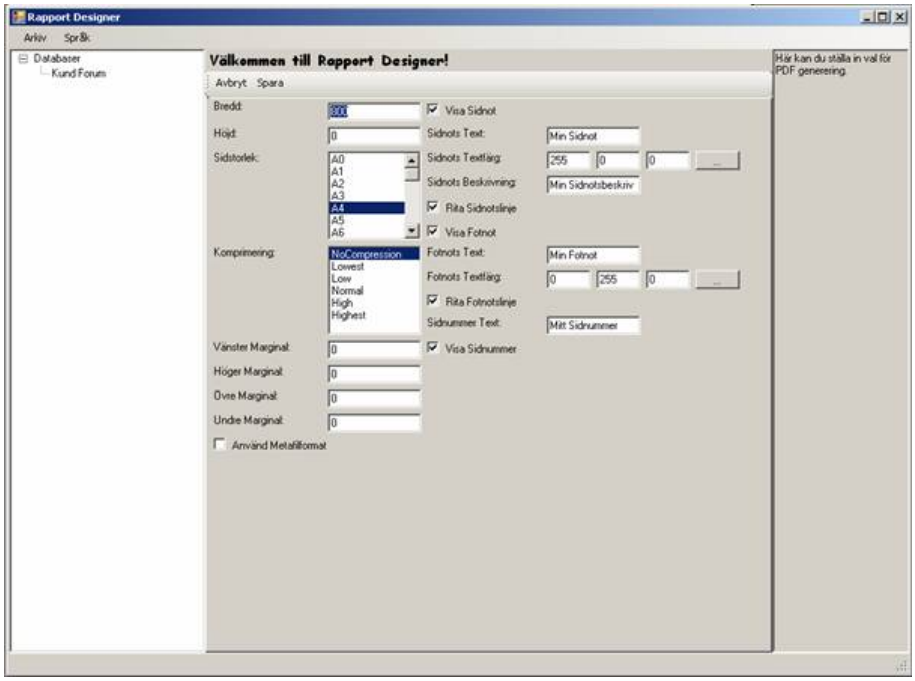
Fig. 5.2 visar hur programmet ser ut när det startas. I strukturträdet listas de databaskopplingar som finns sparade lokalt.

I detta tillstånd har användaren följande valmöjligheter:

Val av databaskoppling Användaren kan välja en databaskoppling genom att markera den i strukturträdet. Detta aktiverar programtillstånd 5.2.3.

Skapa en ny databaskoppling Genom att klicka på ”Ny” i knappraden kan användaren skapa en ny databaskoppling.

Aktivera tillstånd för PDF-inställningar Genom att klicka på ”PDF-Inställningar” kan användaren aktivera programtillstånd 5.2.2.



Figur 5.3: PDF-inställningar som kan göras i programmet.

5.2.2 PDF-inställningar

När detta tillstånd aktiveras läses lagrade PDF-inställningar in från en lokal XML-fil. Dessa inställningar presenteras i huvudfönstret och kan modifieras av användaren. Inställningarna används då användaren genererar en PDF rapport.

I detta tillstånd har användaren följande valmöjligheter:

Spara PDF-Inställningar Lagrar eventuella förändringar av PDF-inställningarna i en lokal XML-fil. Detta görs genom att klicka på ”Spara”.

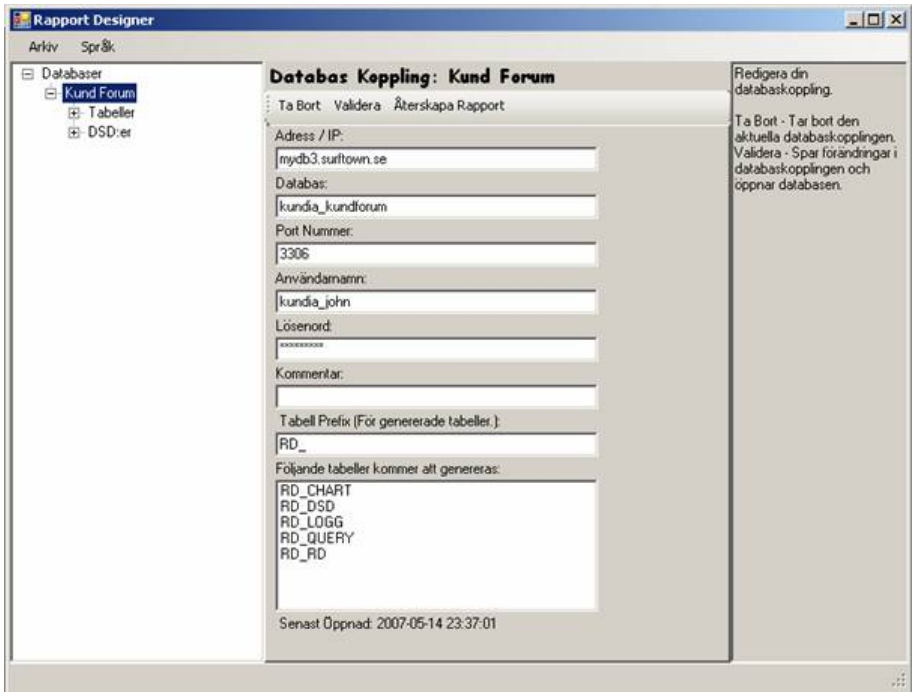
Avbryt Användaren kan återgå till 5.2.1 genom att klicka på ”Avbryt”.

5.2.3 Konfigurera databaskoppling

Då detta tillstånd aktiveras läses den aktuella databaskopplingen in från en lokal XML-fil. Dessa inställningar presenteras i huvudfönstret och kan modifieras av användaren. Report Designer gör skillnad mellan validerade och icke validerade databaskopplingar. Endast om den aktuella databaskopplingen är validerad ansluter programmet till den valda databaskopplingens associerade databas. Om programmet ansluter till en databas listas databasens tabeller och DSD:er i strukturträdet.

I detta tillstånd har användaren följande valmöjligheter:

Ta bort databaskopplingen Den aktuella databaskopplingen kan tas bort genom att klicka på ”Ta Bort”; detta val aktiverar tillstånd 5.2.1.



Figur 5.4: Konfiguration av databasen.

Aktivera tillstånd för återskapning av rapport Genom att klicka på ”Återskapa Rapport” kan användaren aktivera tillstånd 5.2.4.

Val av tabell Givet att den aktuella databaskopplingen är validerad kan användaren aktivera tillstånd 5.2.5 genom att markera en tabell i strukturträdet.

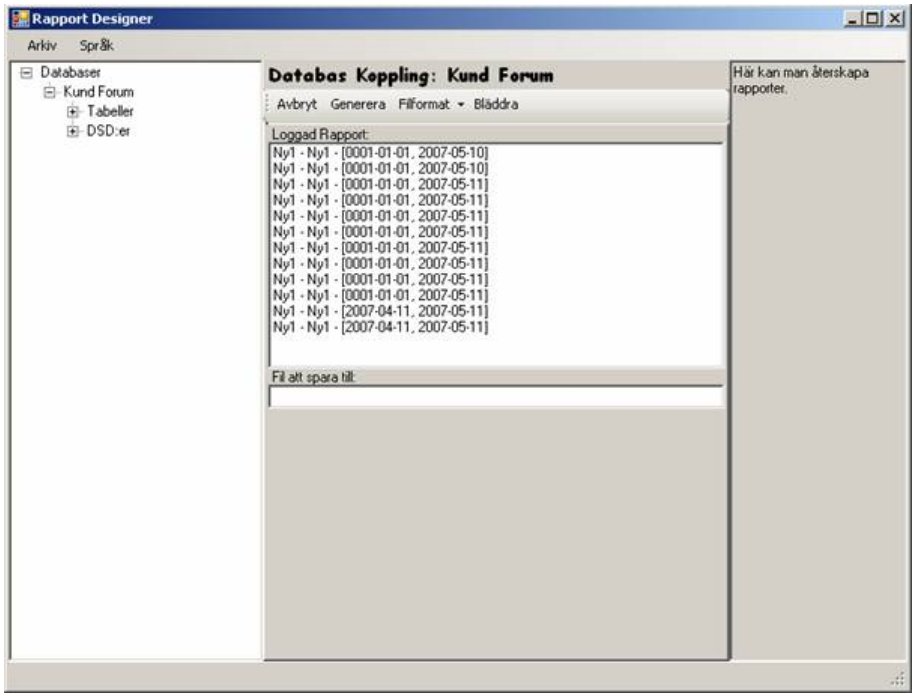
Namnbyte Användaren kan genom att återmarkera den aktuella databaskopplingen byta namn på den.

Vad av DSD Givet att den aktuella databaskopplingen är validerad kan användaren aktivera tillstånd 5.2.6 genom att markera en DSD i strukturträdet.

Validera databaskoppling Om användaren klickar på ”Validera” försöker Report Designer validera den aktuella databaskopplingen. För att databaskopplingen skall kunna valideras måste följande kriterier satisfieras:

1. Det skall gå att ansluta till databasen.
2. Användarnamnet och lösenordet måste accepteras av databasen.
3. Om någon av de tabeller som listas längst ner i fig. 5.4 finns i databasen måste de vara kompatibla med Report Designer; vad detta innebär beskrivs inte djupare här.

Om dessa kriterier uppfylls ansluter sig Report Designer till databasen och skapar (eventuellt) nya tabeller som skall användas av programmet.



Figur 5.5: Gränssnitt för återskapning av rapport.

5.2.4 Återskapning av rapport

I detta tillstånd kan användaren återskapa loggade rapporter. De loggade rapporter som finns i den aktuella databasen listas i huvudfönstret. Det filformat som rapporterna kommer att återskapas i kan väljas i menyn ”Filformat” på knappraden.

I detta tillstånd har användaren följande valmöjligheter:

Avbryt Användaren kan återgå till 5.2.1 genom att klicka på ”Avbryt”.

Val av filnamn Genom att klicka på ”Bläddra” kan användaren välja vad de genererade rapporterna skall heta samt var i filsystemet de ska placeras. Om användaren gör detta val visas en standardiserad ”Öppna”-dialog. Det valda filnamnet placeras i textrutan i huvudfönstrets nedre del.

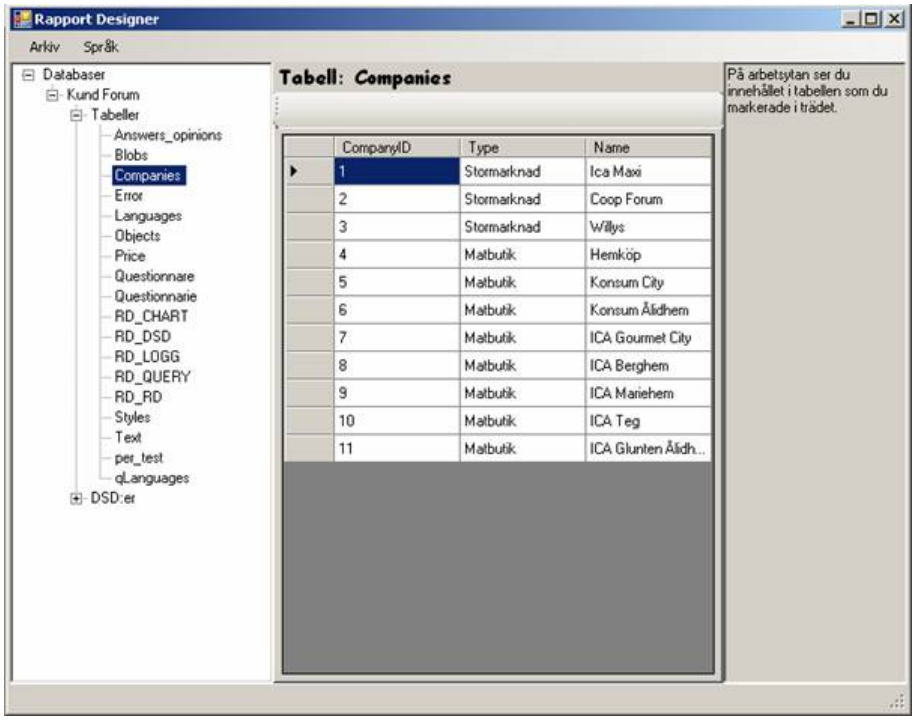
Återskapa rapport(er) Om användaren klickar på ”Generera” återskapas den loggade rapport som är markerad i huvudfönstret.

5.2.5 Visning av tabell

Den markerade tabellens innehåll presenteras i huvudfönstret.

5.2.6 Konfiguration av DSD

Då detta tillstånd blir aktivt presenteras databasens tabeller tillsammans med den aktuella DSD:ns respektive tabelltyper i huvudfönstret. Då användaren markerar en tabell



Figur 5.6: Gränssnitt för visning av tabell.

visas dess fält med respektive fälttyp, detta exemplifieras av fig. 5.7. De fem första posterna i det markerade fältet presenteras längst ner i huvudfönstret, detta syns dock inte i fig. 5.7. Användaren kan byta fält- och tabelltyper genom att använda rullistorna.

I detta tillstånd har användaren följande valmöjligheter:

Skapa ny DSD Om användaren markerar ”DSD:er” i strukturträdet aktiveras ett trivialt programtillstånd som ej redovisas i detta kapitel. Härifrån kan användaren klicka på ”Ny” för att skapa en ny DSD.

Aktivera tillstånd för import av DSD Om användaren markerar ”DSD:er” i strukturträdet och klickar på ”Importera” aktiveras programtillstånd 5.2.7.

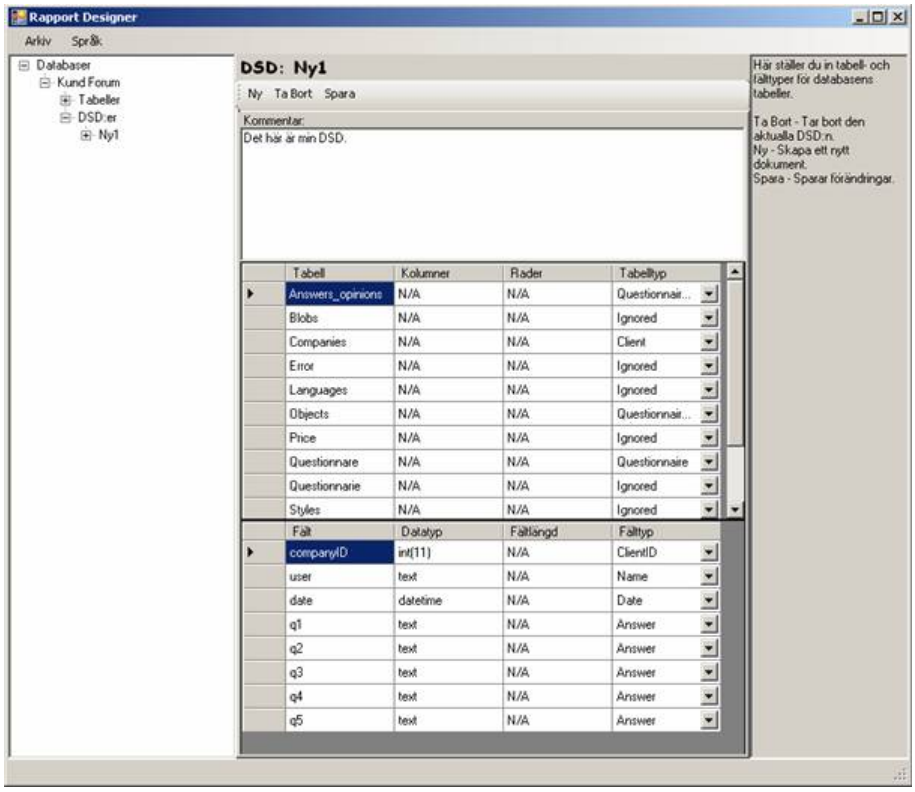
Skapa ny RD Om användaren klickar på ”Ny” skapas en ny RD som associeras med den aktuella DSD:n.

Borttagning av DSD Användaren kan ta bort den aktuella DSD:n genom att klicka på ”Ta Bort”; detta val aktiverar ett trivialt tillstånd.

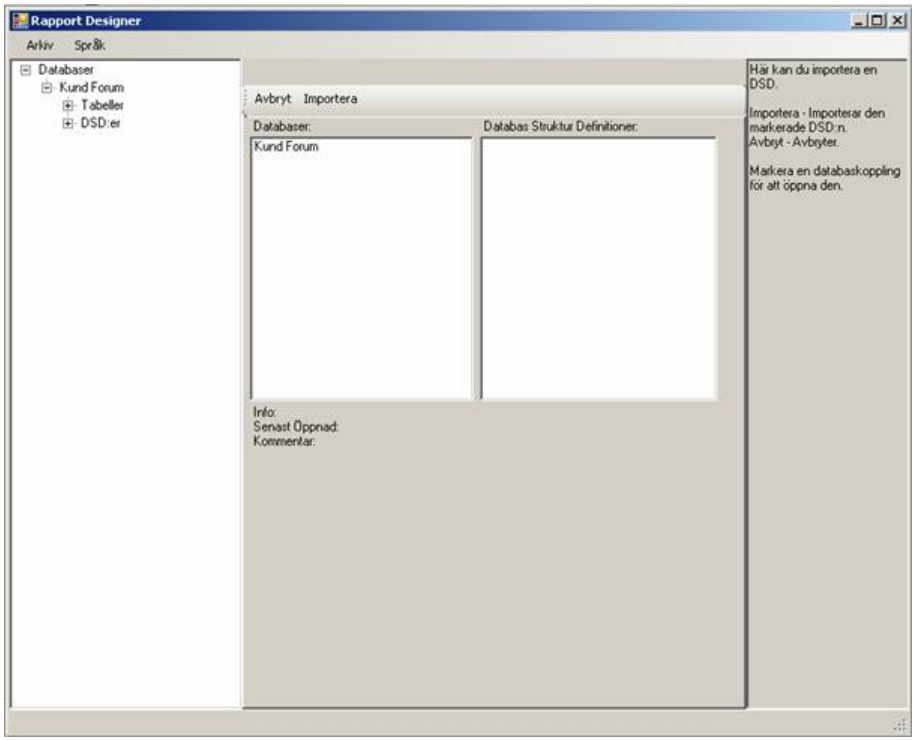
Spara DSD Om användaren klickar på ”Spara” sparas de förändringar som gjorts på DSD:n.

Namnbyte Användaren kan byta namn på den aktuella DSD:n genom att återmarkera den i strukturträdet.

Val av RD Användaren kan välja att redigera en befintlig RD genom att markera den i strukturträdet; detta val aktiverar tillstånd 5.2.8.



Figur 5.7: Konfiguration av DSD.



Figur 5.8: Import av DSD.

5.2.7 Import av DSD

I detta tillstånd kan användaren importera en DSD från en annan databas. Huvudfönstrets vänstra lista listar de lokalt lagrade databaskopplingarna. Markeras en av dessa ansluter Report Designer till den associerade databasen och listar dess DSD:er i huvudfönstrets högra lista.

I detta tillstånd har användaren följande valmöjligheter:

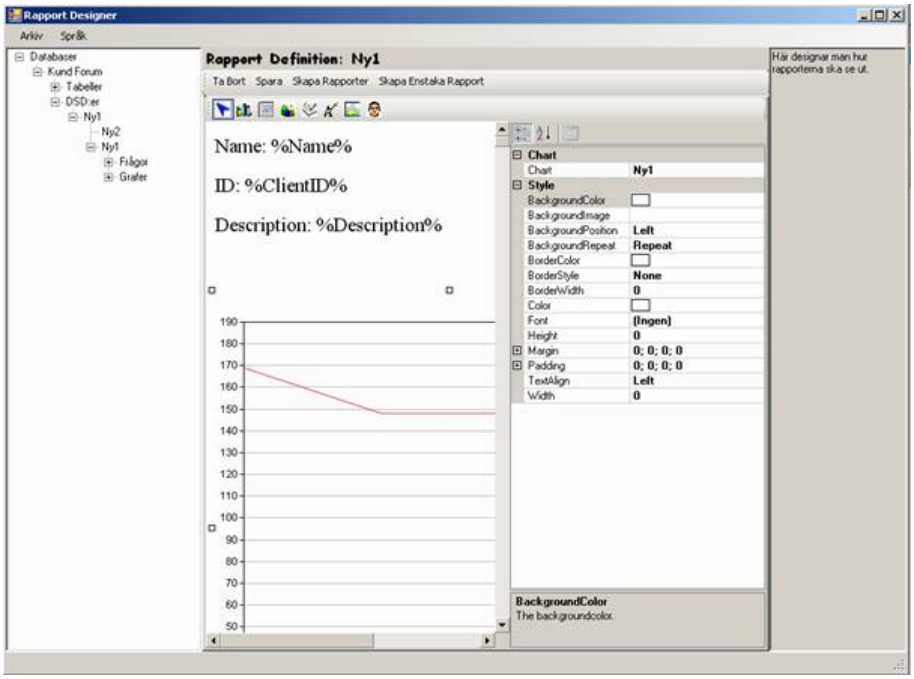
Avbryt Genom att klicka på ”Avbryt” kan användaren aktivera programtillståndet som var aktivt innan detta.

Importera Om användaren klickar på ”Importera” importeras den markerade DSD:n till den aktuella databasen. Endast de tabeller och fält som är applicerbara på den aktuella databasen importeras.

5.2.8 Design av RD

När detta tillstånd blir aktivt presenteras den aktuella RD:n i huvudfönstrets webbläsare. Huvudfönstret är uppdelat i tre delar:

Webbläsaren Webbläsaren innehåller den aktuella rapportdefinitionen. I denna komponent kan användaren modifiera innehållet genom att mata in text, kopiera och klistra in objekt och ta bort objekt. Skriver användaren namnet på en klientfälttyp



Figur 5.9: Designläge för RD.

inom två ”%”-tecken kommer denna text att bytas ut mot den valda klientens data för respektive fält vid rapportgenerationstillfället. Webbläsarens innehåll skiljer sig från det innehåll som kommer att inkluderas i genererade rapporter. Innehållet i webbläsaren är baserat på alla klienter; innehållet i genererade rapporter är endast baserat på en klient.

Egenskapsfönstret Om användaren markerar ett objekt i webbläsaren visas dess egenskaper i egenskapsfönstret till höger i huvudfönstret. I egenskapfönstret visas dels generiska egenskaper för HTML-objektet, dels specifika egenskaper. Exempel på generiska egenskaper är höjd, bredd, färg och bakgrundsbild. Specifika egenskaper är ofta kopplingar till andra objekt i databasen. Alla objektegenskaper kan modifieras av användaren.

Verktygsfönstret Verktygsfönstret är den knapprad som syns ovanför webbläsaren i 5.9. Genom att markera en knapp i verktygsfönstret och klicka på webbläsaren kan användaren infoga nya objekt i rapportdefinitionen. Följande objekt kan infogas:

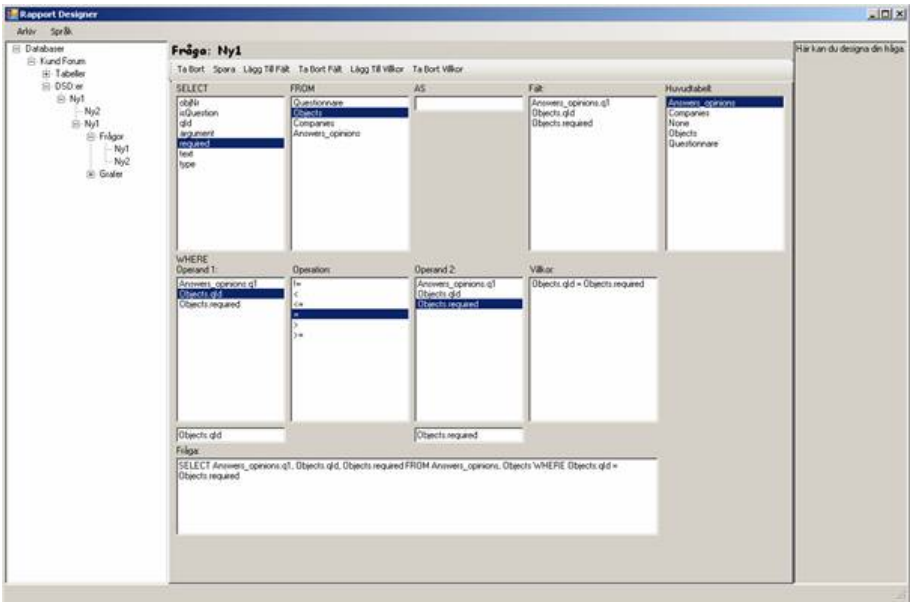
Stapeldiagram Detta objekt är sammankopplat med ett grafobjekt av typen stapeldiagram i databasen.

Linjediagram Detta objekt är sammankopplat med ett grafobjekt av typen linjediagram i databasen.

Cirkeldiagram Detta objekt är sammankopplat med ett grafobjekt av typen cirkeldiagram i databasen.

Tabell Detta objekt är sammankopplat med ett frågeobjekt i databasen.

Textruta



Figur 5.10: Designläge för fråga.

Bild

Enkätsvar

Detta objekt är sammankopplat med en enkät som finns lagrad i databasen.

I detta tillstånd har användaren följande valmöjligheter:

Ta bort Genom att klicka på ”Ta Bort” kan användaren ta bort den aktuella RD:n; detta val aktiverar tillstånd 5.2.6.

Spara Användaren kan spara den aktuella RD:n genom att klicka på ”Spara”.

Aktivera tillstånd för generering av rapporter Användaren kan aktivera tillstånd 5.2.12 genom att klicka på ”Skapa Rapporter”.

Aktivera tillstånd för generering av enstaka rapporter Användaren kan aktivera tillstånd 5.2.11 genom att klicka på ”Skapa Enstaka Rapporter”.

Namnbyte Användaren kan byta namn på rapportdefinitionen genom att återmarkera den i strukturträdet.

Val av fråga Användaren kan aktivera tillstånd 5.2.9 genom att klicka på en fråga i strukturträdet.

Vad av graf Användaren kan aktivera tillstånd 5.2.10 genom att klicka på en graf i strukturträdet.

5.2.9 Design av fråga

Detta tillstånd låter användaren designa SQL-frågor på ett användarvänligt sätt. Då tillståndet aktiveras presenters den aktuella frågan i huvudfönstret. Användaren kan sedan modifiera frågan. Genom att markera ett fält och sedan klicka på ”Lägg Till Fält” kan användaren infoga ett fält i frågan. På ett likartat sätt kan även enkla villkor infogas. Det går dock inte att konstruera alla möjliga SQL-frågor på detta sätt; därför kan användaren själv modifiera SQL-frågan i textrutan längst ner i huvudfönstret. Varje fråga måste associeras med en huvudtabell. Huvudtabellen informerar Report Designer om vilken tabell som innehåller de fält som är relevanta vid generationstillfället.

I detta tillstånd har användaren följande valmöjligheter:

Skapa en ny fråga Om användaren markerar ”Frågor” i strukturträdet aktiveras ett trivialt programtillstånd som ej redovisas i detta kapitel. Härifrån kan användaren klicka på ”Ny” för att skapa en ny fråga.

Ta bort Användaren kan ta bort den aktuella frågan genom att klicka på ”Ta Bort”.

Spara Användaren kan spara de modifikationer som gjorts på frågan genom att klicka på ”Spara”.

Namnbyte Användaren kan byta namn på frågan genom att återmarkera den i strukturträdet.

5.2.10 Design av graf

När detta tillstånd blir aktivt presenteras den markerade grafen i huvudfönstret. Grafen kan modifieras av användaren. En graf innehåller ett antal datakällor. Datakällor är statistiska funktioner av mängder. Dessa mängder är resultat av SQL-frågor. För att lägga till en datakälla klickar användaren på ”Lägg Till Fält” och väljer en fråga. Sedan kan den statistiska funktionen väljas. Det finns fyra funktioner:

Count Räknar antalet element i mängden.

Sum Summerar elementen i mängden. Denna funktion kräver att elementen är av numerisk typ.

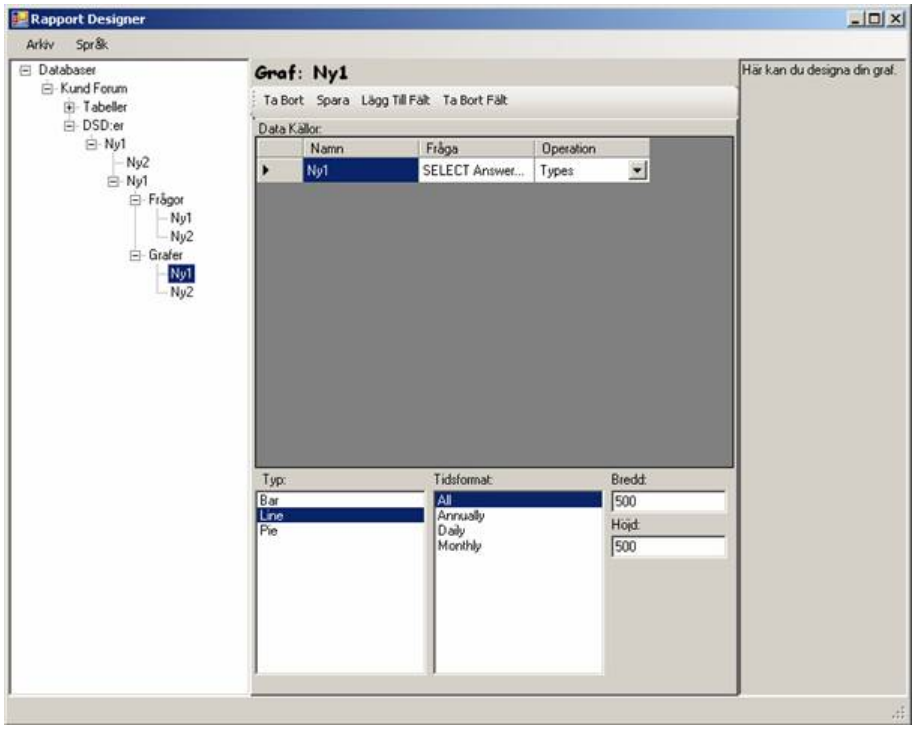
Mean Beräknar medelvärdet av mängden. Denna funktion kräver att elementen är av numerisk typ.

Types Räknar antalet element av samma värde, för alla värden. Detta innebär att denna funktion returnerar en mängd tal istället för ett tal.

En datakälla resulterar alltså i ett eller flera tal. Förutom datakällor inkluderar grafer en graftyp samt en tidsperiod. Graftypen bestämmer om grafen är ett stapeldiagram, linjediagram eller cirkeldiagram. Om en tidsperiod annan än ”All” väljs kommer datakällorna delas upp i respektive tidsperioder.

I detta tillstånd har användaren följande valmöjligheter:

Skapa en ny graf Om användaren markerar ”Grafer” i strukturträdet aktiveras ett trivialt programtillstånd som ej redovisas i detta kapitel. Härifrån kan användaren klicka på ”Ny” för att skapa en ny graf.



Figur 5.11: Designläge för graf.

Ta bort Användaren kan ta bort den aktuella grafen genom att klicka på ”Ta Bort”.

Spara Användaren kan spara de modifikationer som gjorts på grafen genom att klicka på ”Spara”.

Namnbyte Användaren kan byta namn på grafen genom att återmarkera den i strukturträdet.

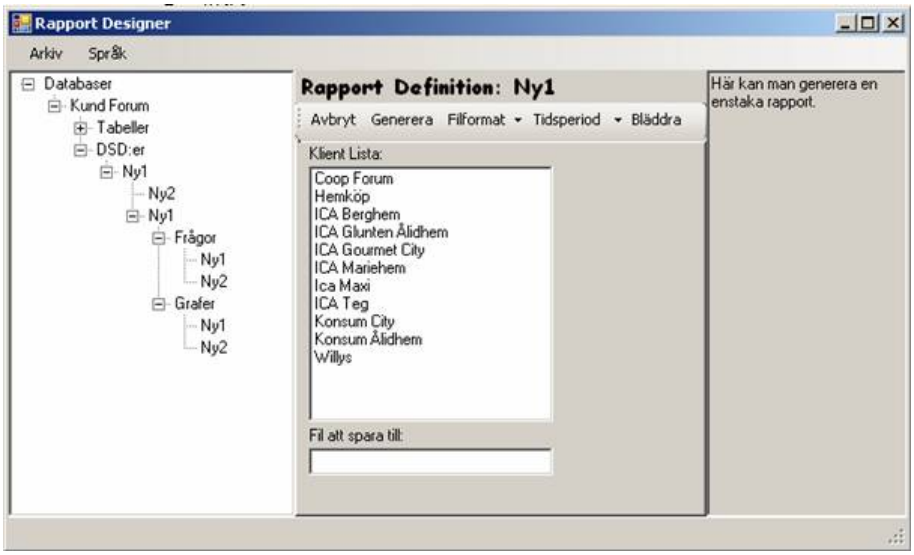
5.2.11 Generation av enstaka rapporter

Då detta tillstånd aktiveras listas de klienter som finns i databasen i huvudfönstret. Klienterna hämtas från tabellen med tabelltyp ”Client”. Användaren kan i knappraden välja filformat, tidsperiod samt filnamn. Om en tidsperiod anges kommer programmet endast att ta med de databasposter som hamnar inom den valda tidsperioden i skapandet av rapporten. En posts tid bestäms av dess fält med fälttyp ”Date”. Om en post saknar ett sådant fält antas posten hamna innanför det valda tidsintervallet.

I detta tillstånd har användaren följande valmöjligheter:

Avbryt Genom att klicka på ”Avbryt” kan användaren återaktivera tillstånd 5.2.8.

Skapa rapport Genom att klicka på ”Generera” kan användaren skapa en rapport med de aktuella parametrarna. Om ingen klient är markerad i huvudfönstret baseras rapporten på data från alla klienter. När rapporten har lagrats i en lokal fil öppnas den i ett nytt fönster.



Figur 5.12: Gränssnitt för skapande av enstaka rapporter.

5.2.12 Generation av rapporter

Då detta tillstånd aktiveras listas databasens BRD:er i huvudfönstret. En BRD är en fråga med en huvudtabell av tabelltyp ”Client”. Val av parametrar kan göras på samma sätt som i 5.2.11.

I detta tillstånd har användaren följande valmöjligheter:

Avbryt Genom att klicka på ”Avbryt” kan användaren återaktivera tillstånd 5.2.8.

Skapa rapport Klickar användaren på ”Generera” skapas en rapport per klient som returneras av den valda BRD:n.

5.3 Tekniska aspekter

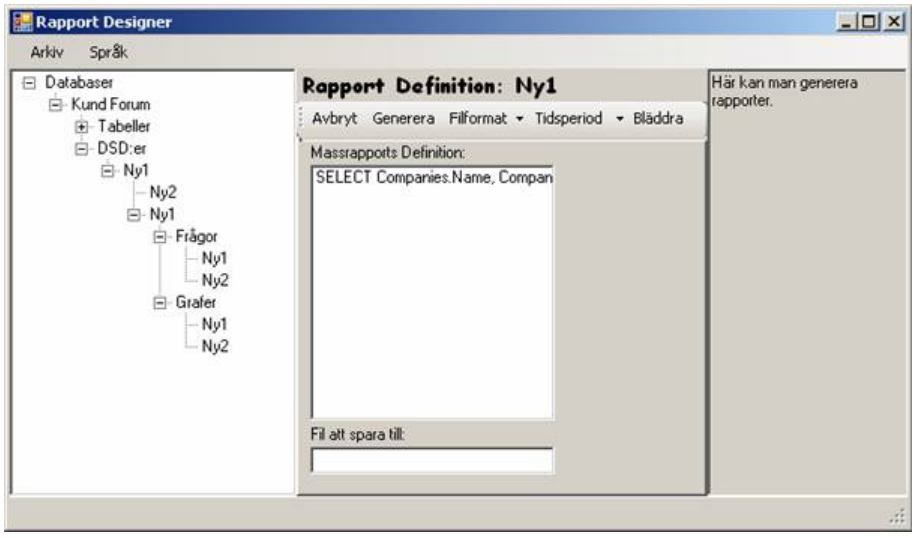
5.3.1 Klasser

Några viktiga klasser

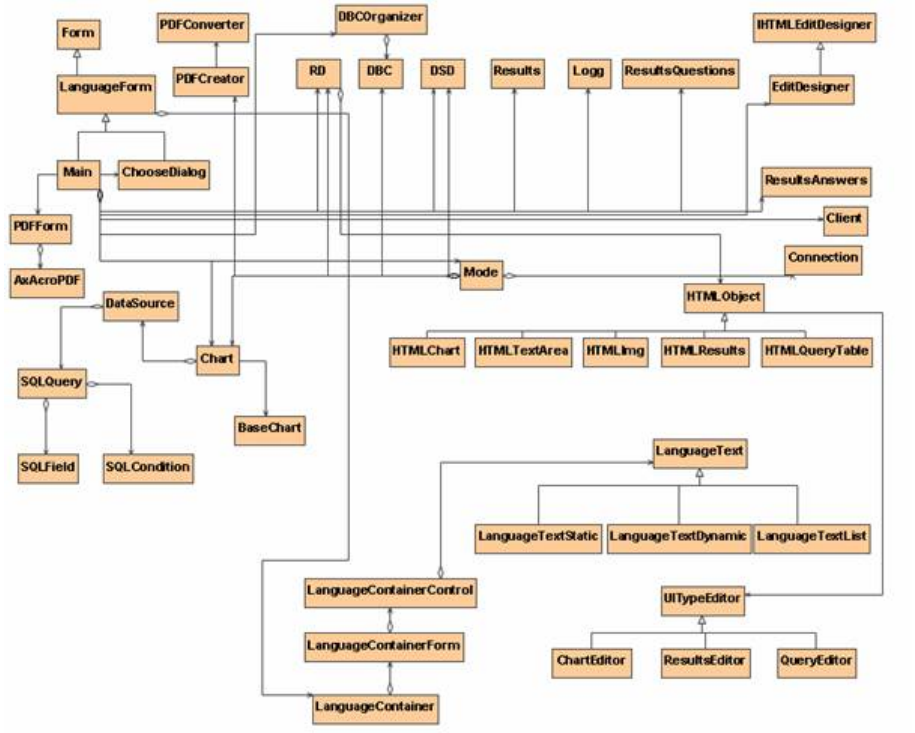
Main Huvudfönstret är en instans av klassen Main. Main är en subklass av LanguageForm som i sin tur är en subklass av den standardiserade .NET-komponenten Form. Form implementerar ett Windows-fönster.

Som impliceras av 5.14 är Main kärnan i programmet. All interaktion med användaren sköts av Main. Report Designer skapar bara en instans av Main som via en statisk metod görs tillgänglig för alla andra klasser i programmet.

WebBrowser och Editdesigner .NET-komponenten WebBrowser har använts vid design av rapportdefinitioner. Komponenten är egentligen menad att fungera som webbläsare men det är även möjligt att använda den som HTML-editor via det underliggande



Figur 5.13: Gränssnitt för skapande av rapporter.



Figur 5.14: UML-diagram över de viktigaste klasserna samt några relaterade .NET-klasser och andra komponenter.

COM-objektet. Fördelen med att använda WebBrowser komponenten är att den tillhandahåller en komplett HTML-designer. Nackdelen är att en del kontroll över komponentens beteende samt utseende förloras. Vidare är arkitekturen som omger WebBrowser komponenten olämplig och svårarbetad. Exempelvis så är det omständligt att skriva in ny HTML-kod i komponenten; istället för att direkt skriva in den nya koden så lägger WebBrowser en händelse i applikationens händelsekö, således ändras HTML-innehållet först när programmet bearbetar kön.

För att få kontroll över de (intressanta) händelser som genereras av WebBrowser komponenten måste en IHTMLEditDesigner implementeras och registreras hos WebBrowser komponenten. Klassen EditDesigner har implementerats i detta syfte.

PropertyGrid PropertyGrid är en .NET-komponent som har använts för att lista egenskaper hos markerade HTML-objekt i rapportdefinitionsdesignläget (, kallat egenskapsfönstret i sektion 3). För att få PropertyGrid att visa egenskaper anropar man en av dess metoder med ett objekt som argument. När detta är gjort listar PropertyGrid objektets egenskaper samt låter användaren modifiera dessa. Problemet med komponenten är att man inte har någon kontroll över vilka egenskaper som visas. För att lösa detta implementerades ett antal wrapper-klasser (HTMLObject, HTMLChart, HTMLQueryTable, HTMLImg och HTMLResults) som endast exponerar de önskade egenskaperna för PropertyGrid.

Ett annat problem med PropertyGrid är att man inte har någon kontroll över vilket sätt komponenten låter användaren modifiera egenskaperna på. Exempelvis kan man inte på ett enkelt sätt få den att öppna en fil-dialog då användaren ska modifiera en sträng. För att lösa detta implementerades ett antal derivat från UITypeEditor (ChartEditor, ResultsEditor och QueryEditor) som definierar funktioner som anropas då användaren modifierar vissa värden.

Connection Connection används av Main för att kommunicera med MySQL-databaser. Klassen implementerar ett skal till den relevanta funktionaliteten hos MySQL Connector [6].

Klassen DBC representerar en databaskoppling. En instans av denna klass ges till Connection då den instansieras. På detta sätt håller klassen reda på vilken databas den skall koppla upp sig mot.

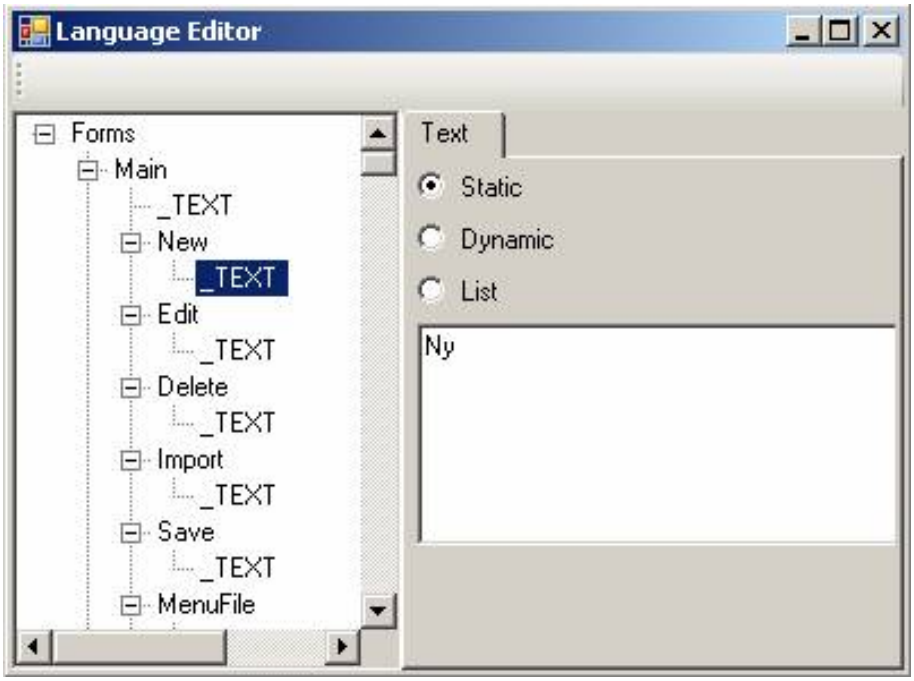
PDFConverter PDFConverter är en del av den färdiga komponenten HTML to PDF Converter [9] som har använts i projektet. Klassen används för att konvertera HTML filer till PDF filer.

AxAcroPDF AxAcroPDF är en klass som ingår i Adobe Acrobat SDK [1]. Klassen används för att presentera de PDF dokument som genereras av programmet.

BaseChart BaseChart är en klass som ingår i den färdiga komponenten Chart Director [3] som har använts för att skapa de grafer som kan inkluderas i rapportdefinitioner.

5.4 Språkstöd

Report Designer stödjer byte av språk under körning. Detta realiseras genom att alla fönster i programmet ärver av LanguageForm. Fönster som ärver av LanguageForm kan an-



Figur 5.15: Bild av Language Editor.

ropa metoden SetLanguage (String language) som väljer language som språk i det aktuella fönstret. Detta förutsätter att det finns en XML-fil som heter lang_<language>.xml i samma katalog som Report Designer.

XML-filen innehåller textsträngar associerade med namn. LanguageForm letar igenom fönstret efter komponenter med värden på sina TAG fält; alla fönsterkomponenter i .NET har ett TAG fält. Där ett TAG fält är lika med ett namn i XML-filen byts komponentens text ut mot textsträngen i XML-filen. Det finns också stöd för att hämta godtyckliga textsträngar ur XML-filen.

XML-filerna med språkdefinitioner skrivs i programmet LanguageEditor som skrevs parallellt med Report Designer. LanguageEditor och Report Designer använder samma klasser för att representera språken - LanguageContainer, LanguageContainerForm, LanguageContainerControl, LanguageText, LanguageTextStatic, LanguageTextDynamic och LanguageTextList; därför har dessa klasser placerats i ett gemensamt klassbibliotek.

Kapitel 6

Slutsatser

Den ursprungliga planeringen som skrevs under projektets första vecka har inte fungerat. Trots detta uppfyller Report Designer de essentiella krav som ställdes initialt. Programmet möjliggör design av rapportdefinitioner samt generering av rapporter; dessutom finns för tillfället stöd för både svenska och engelska.

6.1 Begränsningar

Kravet på att kunna skriva ut rapporter tillsammans med kuvert för postbefordran har inte uppfyllts. Detta beror främst på att kravet inte har haft hög prioritet. I den detaljerade specifikationen som skrevs halvvägs genom projektet fanns kravet inte med.

Ett annat krav som ej är uppfyllt är schemalagd generation av rapporter; det vill säga möjligheten att generera en bestämd mängd rapporter med jämna tidsintervall. Precis som med kravet på utskrifter har detta krav haft låg prioritet och inkluderades ej i den detaljerade specifikationen.

6.2 Framtida arbete

Det finns många förbättringar som skulle kunna göras. Möjligheten att inkludera enkätsvar i rapportdefinitioner är idag väldigt specialiserad och förutsätter i praktiken att enkätdatabasen är designad med Report Designer i tanke. En mer generell modell skulle här vara att föredra.

Designläget för SQL-frågor skulle kunna utökas till att omfatta hela språket. Ett alternativ till detta skulle vara att använda en färdig komponent till detta ändamål. I dagsläget kan bara väldigt elementära frågor ställas utan att man modifierar SQL-koden för hand; denna förbättring skulle därför öka användarvänligheten avsevärt.

Flera förbättringar skulle kunna göras vad gäller rapportdefinitionsdesignläget. Under rapportdefinitionen i strukturträdet skulle HTML-objekten kunna listas, om en av dessa markerades skulle respektive objekt markeras i webbläsaren. För närvarande tillåts bara en liten del av HTML i rapportdefinitionerna, fler objekttyper skulle kunna läggas in. I bästa fall skulle hela HTML-språket stödjas med alla dess objekt och attribut.

Kapitel 7

Tack

Ett stort tack till min uppdragsgivare Andreas Wennström som givit mig denna synnerligen lärorika möjlighet.

Jag vill också tacka Thomas Johansson, min handledare vid Umeå Universitet.

Sist men inte minst vill jag tacka min flickvän, Malin, som varit ett oerhört stöd genom hela projektet.

References

[1] Adobe development center, 2007. Nerladdad 15-maj-2007 från `http://www.adobe.com/devnet/acrobat/`.

[2] Adobe reader, 2007. Nerladdad 15-maj-2007 från `http://www.adobe.com/se/products/acrobat/readstep2.html`.

[3] Chartdirector, 2007. Nerladdad 15-maj-2007 från `http://www.advsofteng.com/`.

[4] Create dynamic web reports in minutes!, 2007. Nerladdad 15-maj-2007 från `http://www.hkvstore.com/phpreportmaker/`.

[5] Documalis freeware, 2007. Nerladdad 15-maj-2007 från `http://www.documalis.com/free/`.

[6] Download connector .net/5.1, 2007. Nerladdad 15-maj-2007 från `http://dev.mysql.com/downloads/connector/net/5.1.html`.

[7] Smart report maker, 2007. Nerladdad 15-maj-2007 från `http://mysqlreports.com/`.

[8] Visual studio 2005 development center, 2007. Nerladdad 15-maj-2007 från `http://msdn2.microsoft.com/sv-se/vstudio/default.aspx`.

[9] Winnovative pdf tools suite, 2007. Nerladdad 15-maj-2007 från `http://www.dotnet-reporting.com/PdfTools.aspx`.

[10] H. Chang and M.J. Atallah. Protecting software code by guards. *Proceedings of the ACM Workshop on Security and Privacy in Digital Rights Management*, pages 160–175, 2000.

[11] Christian Collberg, Clark Thomborson, and Douglas Low. Breaking abstractions and unstructuring data structures. In *International Conference on Computer Languages*, pages 28–38, 1998.

[12] A. Dathathri. Protecting Software against Crackers.

[13] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[14] Dries Schellekens Jan Cappaert, Nessim Kisserli and Bart Preneel. Self-encrypting code to protect against analysis and tampering. In *First Benelux Workshop on Information and System Security*, 2006.

[15] Xiaoping Jia. *Object Oriented Software Development Using Java, 2nd edition.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[16] O. Kommerling and M.G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. *USENIX Workshop on Smartcard Technology*, pages 9–20, 1999.

[17] W. Shi, H.H.S. Lee, C. Lu, and M. Ghosh. Towards the issues in architectural support for protection of software execution. *ACM SIGARCH Computer Architecture News*, 33(1):6–15, 2005.

[18] Nikhil Viswanathan. Can microsoft .net deliver "trustworthy computing"? *Information Security Reading Room*, 2002.

[19] G. Wroblewski. General method of program code obfuscation, 2002.