Fakulteten för ekonomi, kommunikation och IT
Datavetenskap

David Berg, Malin Öborn

# Utbyggnad av Diamond

-

# ett generellt dokumenthanteringssystem

# Extension of Diamond

-

# a General Document Management System

Datavetenskap
C-uppsats

Datum/Termin: 12-01-11
Handledare: Karl-Johan Ginnemo
Examinator: Donald F. Ross
Löpnummer: C2012:1

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

# Förord

Vi skulle vilja tacka vår handledare, Karl Johan Grinnemo, som har varit ett stort stöd under projektets gång. Vi vill även tacka Sogeti Karlstad för att vi fick möjligheten att göra vårt examensarbete tillsammans med er. Att få sitta på plats hos er under arbetets gång var mycket givande och lärorikt. Slutligen, ett stort tack till personalen på Gruvöns bruk som tog emot oss för ett studiebesök. Vi kände oss välkomna hos er och fick svar på de frågor och funderingar vi hade.

Karlstad den 11 januari 2012

David Berg och Malin Öborn

## Sammanfattning

Att hantera och distribuera dokument är något som är centralt för många organisationer. Diamond är ett exempel på ett system som används för att förenkla sådana aktiviteter. Vårt examensarbete har inneburit att utöka Diamonds funktionalitet så att det klarar av att hantera fler filtyper. Resultatet har blivit ett system som kan hantera alla filtyper, men där vissa filtyper inte går att distribuera till samtliga destinationer. Detta är en ganska naturlig begränsning; det går till exempel inte att distribuera ett dokument som utgörs av en videofil till en skrivare. För att visa hur man trots allt kan minska denna begränsning så gjorde vi en prototyp-lösning för utskrift av PDF-filer. Denna lösning anser vi att man i framtiden skulle kunna utveckla så att ännu fler filtyper kan hanteras av Diamond.

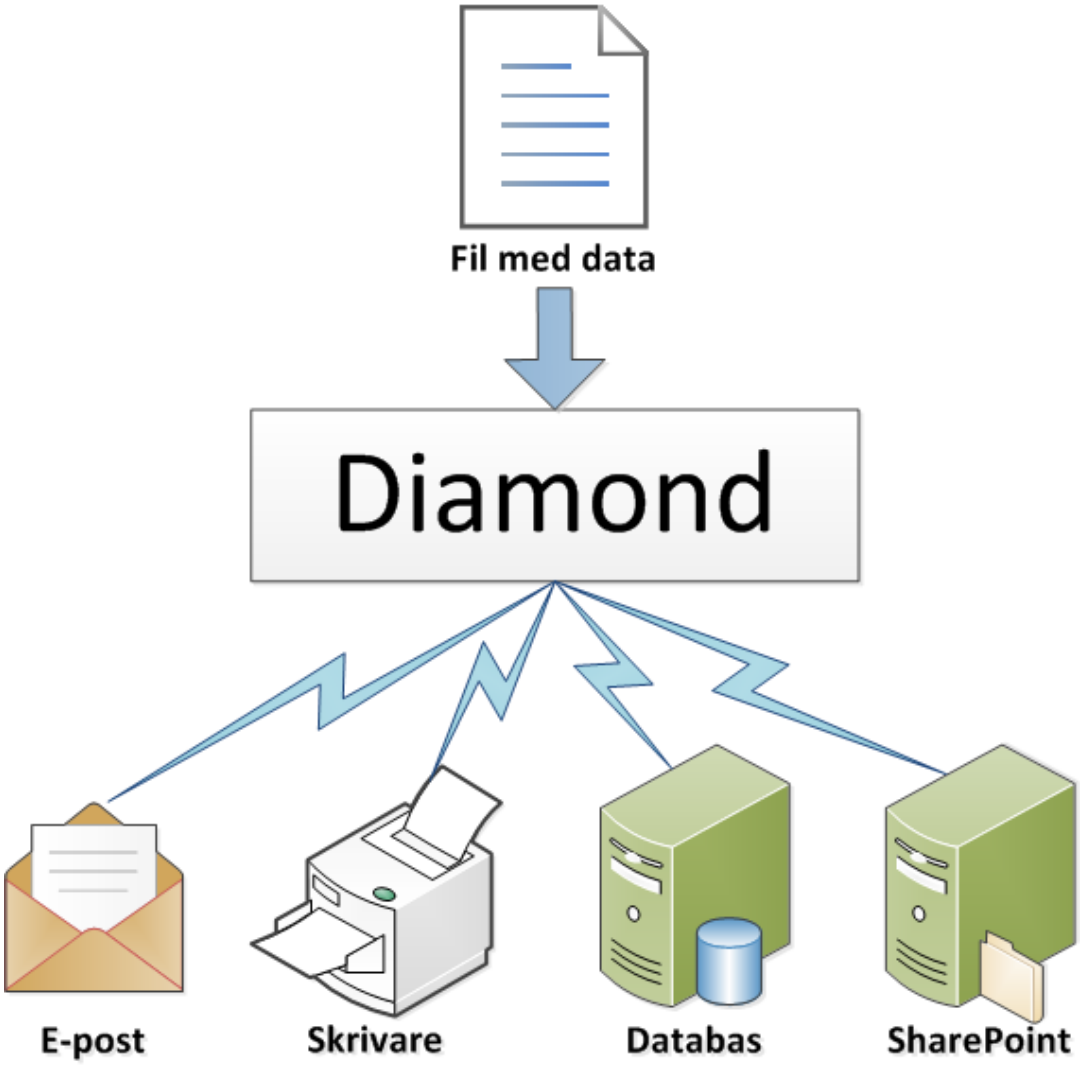
## Abstract

Managing and distributing documents is something that is central to many organizations. Diamond is an example of a system that is used to facilitate such activities. Our dissertation is meant to extend Diamond's functionality to make it handle more file types. The result is a system that can handle all file types, but certain file types cannot be distributed to all destinations. This is a quite natural restriction; as an example, it is not possible to distribute a document that consists of a video file to a printer. To show how this limitation can be reduced, we made a prototype solution for printing PDF files. We believe that this solution can be further developed in the future, so that even more file types can be handled by Diamond.

# Innehållsförteckning

# 1 Introduktion

Något som är centralt i många verksamheter är behovet av att distribuera dokument och rapporter på ett effektivt sätt. Förr sköttes denna process manuellt. I dagsläget är det vanligt att man istället automatiserar processen med hjälp av IT-system. Ett exempel på ett sådant system är Diamond.




**Figur 1-1: Översiktlig bild av Diamond.**

Diamond (akronym av Document Information And Management On Demand) är en så kallad Output Management-lösning, det vill säga ett system som underlättar distribution av dokument (Figur 1-1). Systemet är utvecklat av Sogeti i Karlstad och är en sammankoppling av olika Microsoft-komponenter: en BizTalk-applikation, en Windows Communication Foundation-tjänst, en Reporting Services-tjänst, en SQL-databas och en SharePoint-tjänst. XML är det filformat som utgör kärnan i systemet. Mer utförliga beskrivningar av de olika komponenterna följer i avsnitt 2.2**.**

## 1.1 Diamond i praktiken – Gruvöns bruk

Billerud är en koncern som levererar pappersmassa och förpackningsmaterial. En av koncernens produktionsanläggningar finns belägen på Gruvön, Grums. Denna anläggning använder i dagsläget Diamond för att distribuera dokument och rapporter. Man har tidigare använt sig av StreamServe (http://www.streamserve.com/), ett annat system för distribution av dokument.

Vid samtal med personal på Gruvöns IT-avdelning[1] framkom det att man innan Gruvön började använda sig av automatiserad dokumentdistribution, skrev ut dokument på papper och delade ut dessa manuellt. För att kunna generera dokument utifrån förbestämda mallar så använde man sig av en form av kopiator, Rank Xerox 960. Denna kopiator innehöll en trumma med ett antal standardmallar för exempelvis fakturor, fraktsedlar och dylikt. Varje sådan mall hade en unik kod, som användes för att avgöra enligt vilken mall dokumentet skulle genereras. När man skulle generera dokumentet fick man först, med hjälp av ett affärssystem, skriva ut de data som dokumentet skulle innehålla. Den resulterande pappersutskriften lades sedan i ovan nämnda kopiator, varpå man angav koden för den önskade mallen och startade kopieringen. Resultatet blev ett dokument som var formaterat enligt den valda mallen.

Gruvön genererade sina dokument på detta sätt fram till 1988, då man gick över till en moderniserad variant av samma lösning. Lösningen innebar att man lät en PC skicka lagrad data, samt den kod som motsvarade önskad mall, till en laserskrivare. Laserskrivaren skrev sedan ut ett dokument med inskickad data ifylld i den valda mallen. Från denna lösning gick man så småningom över till att använda StreamServe, vilket år 2009 ersattes med Diamond. Konsekvenserna av bytet från StreamServe till Diamond var framför allt minskade driftskostnader.

Med hjälp av Diamond är det möjligt att skapa formaterade dokument efter olika mallar, och distribuera dessa. På Gruvöns bruk används denna funktion bland annat för att generera dokument med riktlinjer åt de chaufförer som levererar material till bruket. Det är logistikavdelningen som ansvarar för uppgifterna kring när inplanerade leveranser ska anlända. Denna avdelning har även ansvar för att, med hjälp av receptionen, dirigera de ankomna lastbilarna till rätt område inom bruket. Uppgiften sköts med hjälp av Diamond: Logistikavdelningen skapar ovan nämnda dokument och distribuerar detta till receptionen. Receptionen kan då skriva ut dokumentet och lämna över det till respektive lastbilschaufför vid ankomst.

En annan uppgift som Diamond underlättar är distribution av ett dokument till ett flertal olika destinationer och mottagare. Det är med Diamond möjligt att organisera destinationer och mottagare i namngivna grupper. Sådana grupper kallas inom Diamond för distributionskanaler. Vid distribution av ett dokument anger man namnet på den distributionskanal man önskar

[1] Samtalet ägde rum vid ett studiebesök på Gruvöns bruk den 26/9 2011

använda, och dokumentet distribueras då till alla de destinationer och mottagare som ingår i kanalen.

På Gruvöns bruk finns det exempel på användningsfall där användaren främst utnyttjar Diamond för möjligheten att generera formaterade dokument. När dokumenten skapats så skickar användaren dessa till en skrivare på sitt kontor. De utskrivna dokumenten distribueras sedan manuellt.

Diamond erbjuder lagring av dokument på SharePoint-servrar. En fördel med detta är att man har möjlighet att ge externa intressenter tillgång till dokumenten på servrarna. Intressenterna har då alltid tillgång till de mest aktuella versionerna av dokumenten. Att ge utomstående tillgång till dokument på detta vis var inte möjligt med StreamServe. Diamond använder sig av en databas, i vilken data om alla dokument som passerar systemet lagras. Detta ger spårbarhet samt en möjlighet att distribuera ett dokument flera gånger.

## 1.2 Problembeskrivning och avgränsningar

Något som saknats i Diamond har varit möjligheten att distribuera dokument och rapporter som genererats av ett externt system. Systemet kunde enbart hantera XML-filer med data, utifrån vilka systemet genererade en rapport som sedan distribuerades till vald(a) destination(er). Målet har varit att Diamond — förutom att generera och distribuera egna rapporter — även ska kunna hantera distribution av rapporter som har skapats av externa system eller användare. Önskemål om denna funktion har kommit från Sogetis kunder, och faktum är att Sogeti själva har sett ett behov av denna funktion. Vårt examensarbete har bestått i att bygga ut Diamond med denna funktion.

Vi var från början inställda på att projektet skulle innefatta en BizTalk-lösning eftersom detta stod i projektspecifikationen, men så blev inte fallet. Det finns två anledningar till detta: dels rymdes det inte inom projektets tidsram, och dels kom vi, efter diskussion med vår handledare på Sogeti, fram till att det var överflödigt. Något som ytterligare stärkte detta beslut var en diskussion vi hade med en anställd på Sogeti, som administrerar och utvecklar Diamond. Det framkom under diskussionen att BizTalk bör ses som en utbytbar komponent, som vid önskemål ska kunna ersättas med andra liknande verktyg.

En annan komponent som ingår i Diamond som vi inte har arbetat med är Reporting Services (RS). RS har som uppgift att generera de rapporter som ska distribueras, och då vi endast har hanterat rapporter genererade av andra program så har vi inte behövt använda den funktionen. Då vi anser att en viss kunskap om BizTalk och RS behövs för att få en helhetsbild av Diamond, så har vi trots detta valt att inkludera beskrivningar av dessa komponenter i avsnitt 2.

## 1.3 Bidrag

I vårt examensarbete har vi utökat Diamond så att systemet inte längre saknar den önskade funktionalitet som nämndes under föregående rubrik. Diamond klarar i dagsläget av att

distribuera vilka filtyper som helst, och därmed har systemets användningsområde utökats. Systemet är fortfarande begränsat vid distribution till skrivare, eftersom denna funktion ännu inte kan hantera alla filtyper. Vi har tagit fram två konceptuella lösningar som visar hur man, genom översättning mellan filformat, kan modifiera Diamond för att minska denna begränsning. Vi har därmed lagt en grund för fortsatt utveckling av systemet. Diamonds nya funktion är implementerad med hänsyn till programkodens ursprungliga design. Fokus har varit att integrera den nya funktionen med Diamonds tidigare funktioner: att producera en lösning som är väl integrerad och enkel att bygga ut vid behov.

## 1.4 Rapportens disposition

Resten av rapporten är organiserad som följer. Avsnitt 2, ”Teori”, ger en översiktlig beskrivning av Diamonds arkitektur. Målet är att ge läsaren en grundläggande förståelse för Diamond, och därmed göra så att resterande avsnitt blir mer begripliga. Avsnitt 3, ”Lösningsförslag”, beskriver och utvärderar de förslag på lösningar som vi har studerat i vårt examensarbete. Avsnitt 4, ”Implementation”, ger en detaljerad beskrivning av valt lösningsförslag. Avsnitt 5, ”Diskussioner och erfarenheter”, ger våra reflektioner kring det utförda projektet. Rapporten avslutas med avsnitt 6, ”Sammanfattning och slutsatser”, vilken ger en sammanfattad beskrivning av projektet och dess resultat. Detta avsnitt tar även upp idéer och förslag till eventuell vidareutveckling av Diamond.

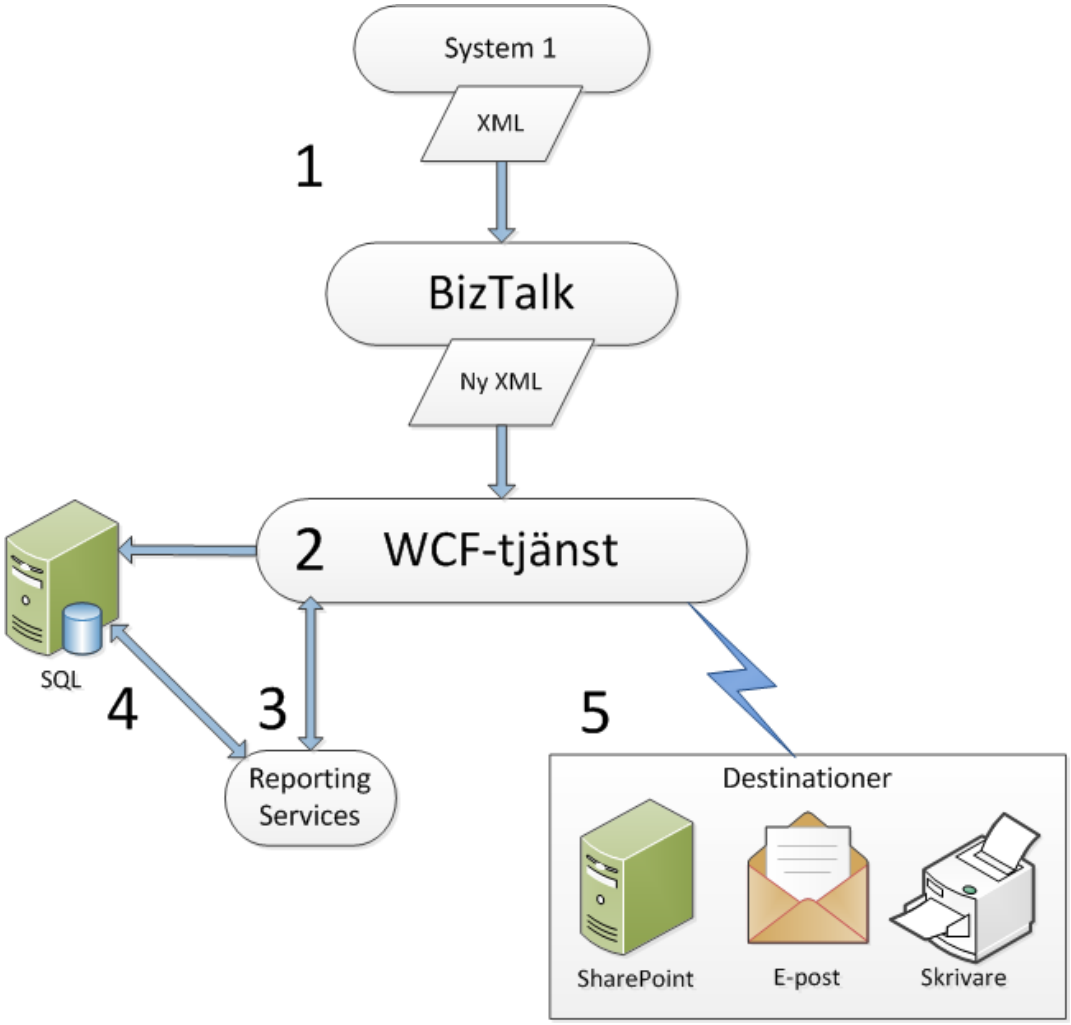
# 2 Teori

I följande avsnitt beskrivs översiktligt de tekniker och komponenter som man behöver känna till för att kunna tillgodogöra sig det övriga innehållet i uppsatsen. Avsnittet inleds med en beskrivning av det logiska flödet för en dokumentfil genom Diamond (med dokumentfil avses här en fil i formatet XML, som innehåller underlaget till en rapport). Därnäst följer beskrivningar av de komponenter som ingår i Diamond. Fokus ligger på den funktionalitet som är central för hur Diamond fungerar, övriga detaljer har medvetet utelämnas.

## 2.1 Diamond - översikt

Som nämndes i föregående avsnitt, utgörs Diamond av en sammankoppling av ett antal Microsoft-applikationer och -tjänster, vilka samarbetar för att uppnå önskad funktionalitet. I det följande avsnittet ges en översiktlig beskrivning av de olika Microsoft-komponenternas roller. På denna beskrivning följer ett praktiskt exempel på hur tjänsterna samverkar.

BizTalks roll i systemet är att ta emot den inkommande dokumentfilen, och skicka den vidare till WCF-tjänsten. Det är med andra ord via BizTalk som användare kommunicerar med Diamond. WCF-tjänstens roll är att samordna kommunikationen mellan ett antal andra tjänster i systemet. Den första tjänsten som anropas är en SQL-databas som har ansvar för att lagra den data som finns i den till WCF-tjänsten inskickade filen. Den andra tjänsten är Reporting Services, vars roll är att utgående från det data som lagrats i SQL-databasen generera en rapport. Det är slutligen WCF-tjänsten som skickar rapporten till korrekta distributionsdestinationer.




**Figur 2-1: En fils flöde genom Diamond.**

För att förklara de olika delarnas roll i Diamond följer här en beskrivning av hur en fil innehållande rapportdata hanteras av systemet: hur den skickas in, görs om till en rapport och skickas ut till valda destinationer (Figur 2-1).

1. En fil kommer in i BizTalk. BizTalk genererar en ny XML-fil, enligt ett fördefinierat format som Diamonds WCF-tjänst kan hantera. Den genererade XML-filen skickas sedan från BizTalk till WCF-tjänsten.

2. WCF-tjänsten tar emot XML-filen och delar upp den i två delar: ett huvud och en kropp. Kroppen innehåller rapportunderlaget och huvudet innehåller information om var rapporten ska distribueras, samt aktuell rapportmall.

3. WCF-tjänsten sparar all data i XML-filen i en SQL-databas. Sedan anropar WCF-tjänsten RS, med indata som talar om vilken rapportmall som ska användas, samt var i databasen RS kan hitta den data rapporten ska fyllas med.

4. RS genererar rapporten enligt angiven mall och fyller den med de data som finns sparade i databasen. Rapporten genereras i det format som destinationen/destinationerna kräver. Rapporten returneras till WCF-tjänsten.

5. WCF-tjänsten distribuerar rapporten till den/de destination(er) som angivits i XML-huvudet.

## 2.2 Diamond – komponenter

### 2.2.1 XML

XML är en förkortning av eXtensible Markup Language (1) och är det format som Diamonds BizTalk-implementation använder för att kommunicera med systemets WCF-tjänst. XML är ett textbaserat format som används för att representera strukturerad information. En fördel med XML är att det är ett standardiserat och plattformsoberoende format med stor spridning.

```
<?xml version=”1.0" encoding=”ISO-8859-1"?>

<produkter>
   <produkt>
      <namn>ExempelProdukt</namn>
      <pris>19,90</pris>
   </produkt>
   <produkt namn=”ExempelProdukt2" pris=”24,90"/>
</produkter>
```

**Figur 2-2: Exempel på XML-fil.**

Informationen i en XML-fil struktureras och presenteras med hjälp av taggar (Figur 2-2). Dessa taggar, och de data som de innehåller, kallas med ett samlingsbegrepp för element. En fördel med att strukturera informationen på detta sätt är att man enkelt kan komma åt önskad information. Denna funktionalitet kommer till nytta i Diamond, till exempel när WCF-tjänsten ska dela upp den inkomna XML-filen i ett huvud och en kropp: all information som ska finnas i huvudet ligger i ett visst element och all information som hör till kroppen ligger i ett annat element.

Ett element kan antingen deklareras mellan en start- och en sluttagg eller i en enda tagg. Det första `Produkt`-elementet i Figur 2-2 är deklarerat på det förstnämnda sättet medan det andra elementet är deklarerat på det sistnämnda sättet. Ett element har vanligtvis ett eller flera attribut (jämför exempelvis taggarna `<namn>` och `<pris>` som är deklarerade i första `Produkt`-elementet i Figur 2-2). Lägg märke till skillnaden i hur de båda `Produkt`-elementens attribut är deklarerade. Första elementet har sina attribut deklarerade mellan start- och sluttaggen medan

andra elementet har sina attribut deklarerade inuti en enda tagg. Det är även möjligt att deklarera vissa attribut i starttaggen och vissa mellan start- och sluttaggen.

Element kan deklareras inuti andra element, det vill säga mellan dessa elements start- och sluttaggar. Man säger då att det förstnämnda elementet är nästlat. Detta är fallet med båda `Produkt`-elementen i Figur 2-2, då de är deklarerade mellan `Produkter`-elementets start- och sluttagg.

```
<?xml version=”1.0" encoding=”ISO-8859-1=?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
   <xsd:element namn=”Produkt”>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element namn=”produktNamn” type=”xsd:string/>
            <xsd:element pris=”pris” type=”xsd:int/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
```

**Figur 2-3: Exempel på XML-schema.**

För att säkerställa att en XML-fil är formaterad enligt en förutbestämd struktur och innehåller viss förutbestämd information kan ett XML-schema användas (Figur 2-3). Ett XML-schema är en XML-fil som innehåller information om önskad struktur och önskat innehåll hos en eller flera andra XML-filer (2). Schemat används sedan för att validera dessa XML-filer. Diamond använder sig av scheman i sin BizTalk-implementation, närmare bestämt för att formatera inkommande filer till en XML-fil med fördefinierad struktur. Scheman använder en något annorlunda syntax jämfört med vanliga XML-filer, då de bland annat innehåller element som specificerar själva formatet hos de XML-filer som ska valideras. Exempel på detta är elementen `<xsd:complexType>` och `<xsd:sequence>`. De förstnämnda av dessa element specificerar att alla element som finns deklarerade inuti det tillåts vara nästlade. Det andra elementet specificerar, vilket namnet antyder, att de element som deklareras inuti aktuellt element måste ligga i samma ordning i den tillhörande XML-filen som i schemat. I annat fall anses inte XML-filen vara valid.

```
<ReportMessage>

   <Header>
      <ReportTemplateName>HyresFaktura</ReportTemplateName>
      <DistributionName>Hyresgäster</DistributionName>
      <Created>2011-10-18</Created>
   </Header>

   <ReportData>
      <Rubrik>Faktura</Rubrik>
      <Adress>Storgatan 2</Adress>
      <Meddelande>Du har 15 dagar på dig att betala</Meddelande>
      <Belopp>4000kr</Belopp>
   </ReportData>

</ReportMessage>
```

**Figur 2-4: Exempel på XML-fil som skickas in i Diamond.**

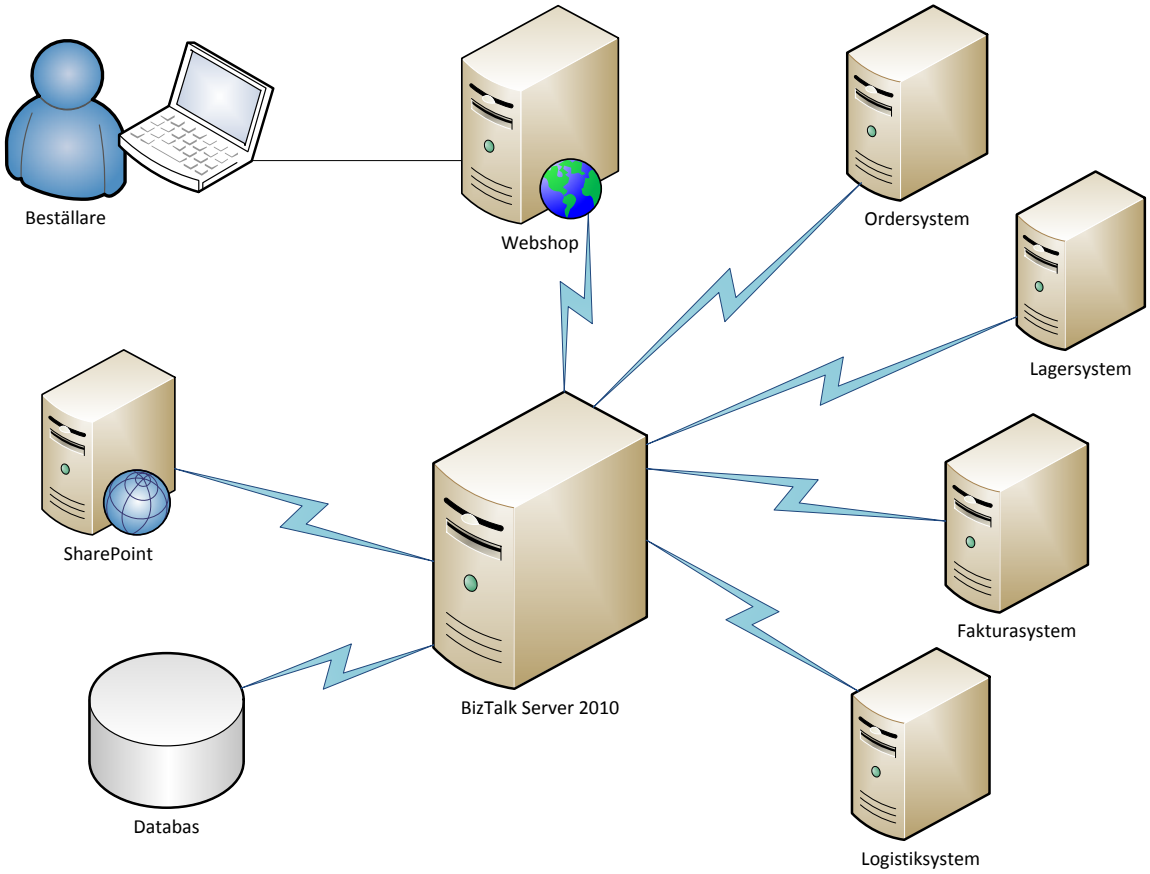
Bilden ovan (Figur 2-4) visar ett konkret exempel på hur en XML-fil som kommer in Diamond kan se ut. De data som hanteras av Diamond är definierade i elementet `ReportMessage`. `ReportMessage` innehåller två nästlade element: `Header` och `ReportData`.

`Header` innehåller information om vilken mall som ska användas vid genereringen av rapporten, var rapporten ska distribueras samt när rapporten skapades. I det här exemplet är det en hyresfaktura som ska genereras, och den ska gå ut till alla hyresgäster.

I `ReportData`-elementet ligger det innehåll som ska finnas i den rapport som Diamond genererar. Alla rapportmallar har olika data som ska, eller kan, finnas med i motsvarande rapport. I det aktuella exemplet består dessa data av en rubrik, en adress, ett meddelande och ett belopp. Med hjälp av denna data så kan RS (se 2.2.4) skapa en rapport i ett antal olika format.

### 2.2.2 BizTalk

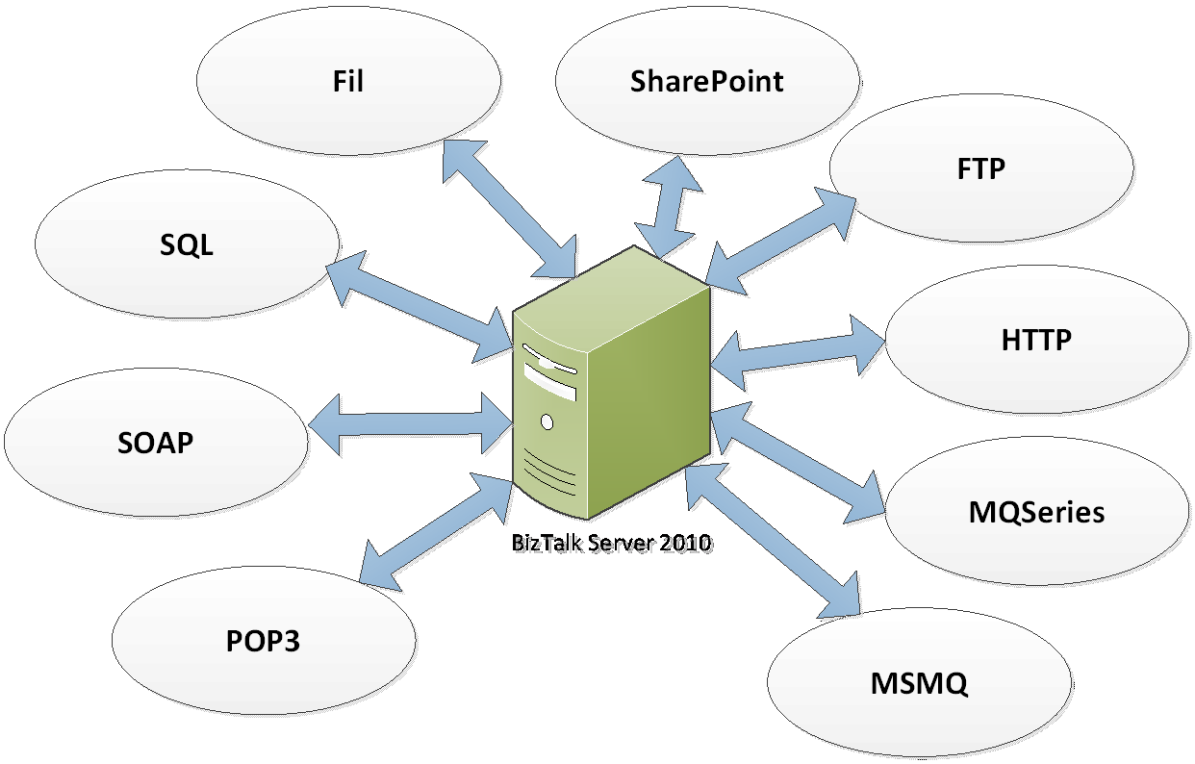
BizTalk är ett integrationsverktyg utvecklat av Microsoft (3). Med integrationsverktyg menas ett verktyg som används för att integrera olika system med varandra (Figur 2-5). Ett praktiskt exempel på en BizTalk-lösning är ett fakturasystem som behöver ta in data från ett ordersystem.




**Figur 2-5: Exempel på hur en lösning med BizTalk kan se ut.**

I Diamond används BizTalk för att ta emot inkommande filer och formatera om dem till en XML-fil. Formateringen sker utifrån ett XML-schema, som specificerar det format som WCF-tjänsten förväntar sig. Något förenklat kan man säga att BizTalk tar in filer (vilka enligt BizTalk-semantik kallas meddelanden) och översätter sedan dessa till det format som mottagaren eller mottagarna vill ha. BizTalk skickar sedan vidare de formaterade filerna till mottagaren eller mottagarna. BizTalk kan, via så kallade adapters, hantera filer, såväl inkommande som utgående, för ett flertal olika plattformar (Figur 2-6). Dessa adaptrar utgör ett gränssnitt mellan BizTalk och en serie vanligt förekommande plattformar (2). De filformat som BizTalk kan hantera som indata är XML och flatfiler[2]; internt använder sig BizTalk av XML.

[2] Med en flatfil avses en textfil i ASCII-format. (17)




**Figur 2-6: BizTalks olika adapters.**

### 2.2.3 WCF

WCF är en förkortning för Windows Communication Foundation, och är ett ramverk som används för att skapa distribuerade applikationer, exempelvis för användning inom en affärsverksamhet eller som en webbservice. WCF är designat för att stödja realiseringen av serviceorienterade arkitekturer (SOA) (4). SOA är ett paradigm som föreskriver hur distribuerade resurser, som eventuellt kan vara spridda på olika platser, kan organiseras och användas (5). Enligt OASIS[3] så kan SOA definieras på följande vis:

*"[…] det representerar en tydlig ansats att separera angelägenheter. Vad detta innebär är att logik som krävs för att lösa ett stort problem kan konstrueras, verkställas och hanteras bättre om den är nedbruten i en samling mindre, relaterade delar. Varje sådan del adresserar en angelägenhet eller en specifik del av problemet."* (5)

En serviceorienterad applikation är med andra ord en applikation som är uppdelad utifrån de specifika problem applikationen är tänkt att lösa. En sådan applikation består av flera mindre delar som var och en har ett specifikt ansvarsområde. Dessa delar kan vara utspridda på olika domäner.

De mindre delar som var och en har ett specifikt ansvarsområde kallas inom WCF för tjänster (services). Tjänster i WCF kan definieras enligt följande:

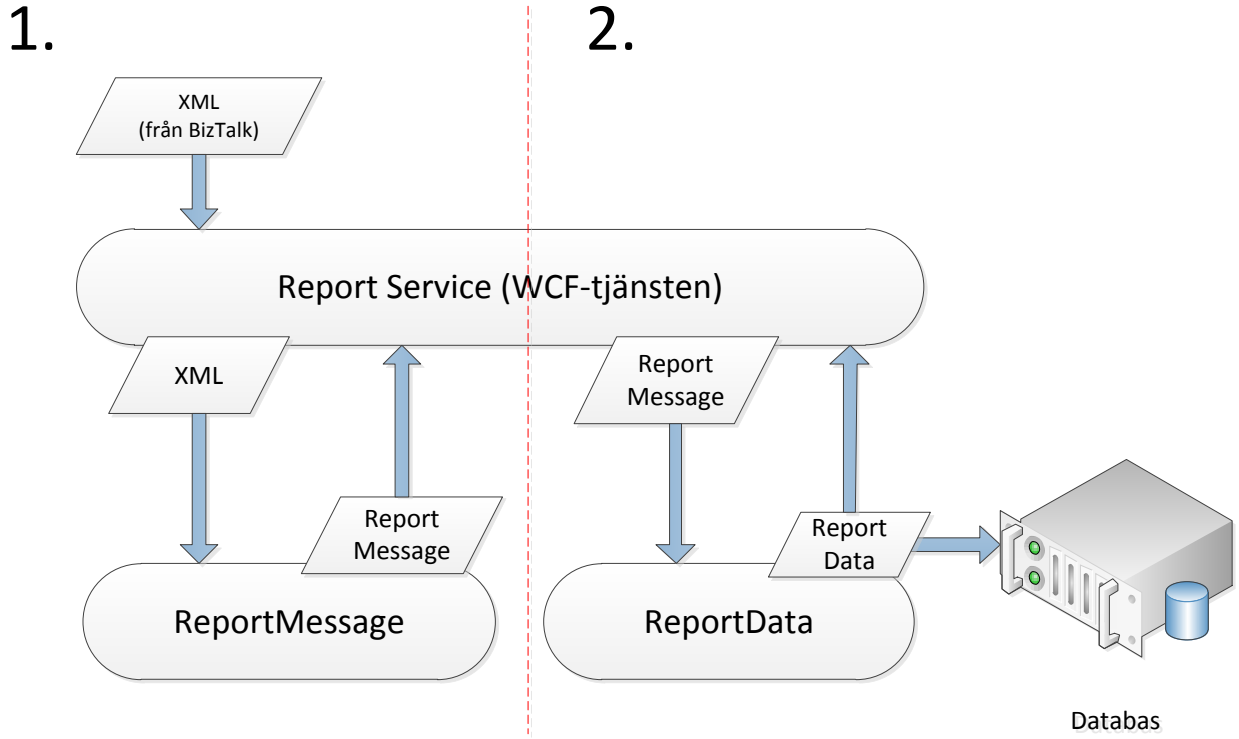
*"Tjänster är ansvariga för att exponera en specifik samling affärsfunktionalitet genom ett väldefinierat kontrakt, vilket beskriver en samling konkreta operationer och meddelanden som tjänsten stödjer."* (4)

En tjänst är med andra ord inkapslad funktionalitet som skyddas genom ett tydligt specificerat kontrakt (representerat av exempelvis en klass) vilket anger hur tjänsten kan anropas och användas. På detta sätt exporteras tjänstens funktionalitet samtidigt som dess implementation är skyddad från omvärlden. Tjänstens kontrakt bör vara hård- och mjukvaruoberoende.

I Diamond ingår en WCF-tjänst som kallas Report Service (notera att denna inte bör förväxlas med Reporting Services). Denna tjänst ansvarar för kommunikationen mellan diverse andra komponenter i systemet, närmare bestämt RS, en SQL-databas samt ett antal destinationer. Tjänsten läser in de data som ska göras om till en rapport samt kopplar ihop dem med relaterad metadata som RS samt destinationerna kräver. Utdata från tjänsten är en formaterad rapport av formatet PDF eller TIFF, vilken distribueras till förutbestämda destinationer.

[3] Organization for the Advancement of Structured Information Standards, en organisation som bland annat arbetar med att driva utvecklingen av öppna standarder inom informationssamhället. (5)

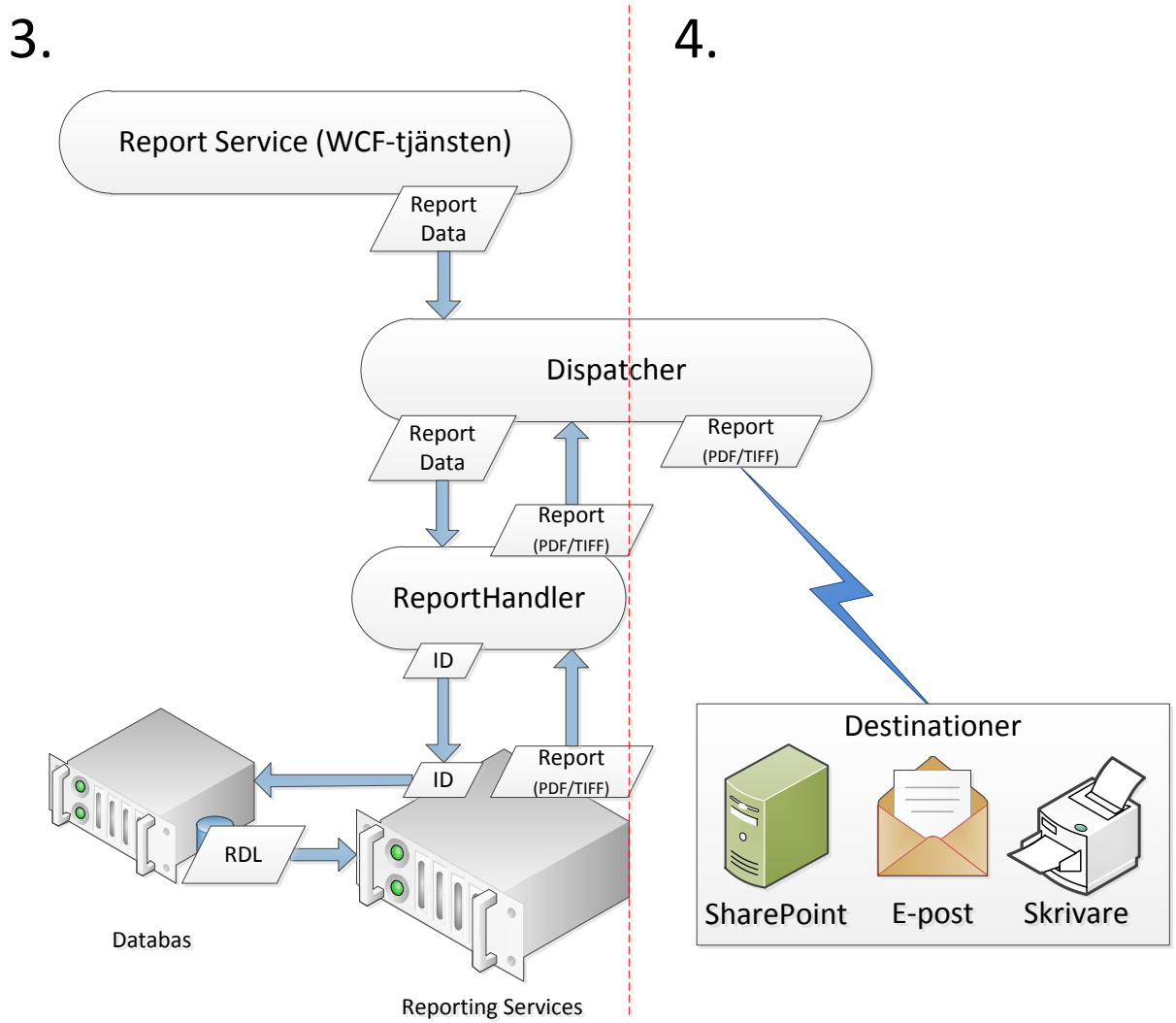
Här följer en översiktlig beskrivning av WCF-tjänsten, Report Service, i Diamond. Vi har valt att dela upp följande beskrivning i fyra steg. De två första stegen representeras av Figur 2-7, och den två sista representeras av Figur 2-8. Tjänsten startas, som tidigare nämnts, av att en XML-fil skickas till tjänsten från BizTalk




**Figur 2-7: WCF-tjänsten, översikt (del 1)**

1. Report Service skickar vidare XML-filen till klassen `ReportMessage`. Filens innehåll delas upp i två delar, och av dessa delar skapas sedan ett `ReportMessage` vilket returneras till Report Service.

2. Report Service skickar `ReportMessage` till klassen `ReportData`, vilken med hjälp av innehållet i `ReportMessage` skapar ett nytt objekt. Objektet innehåller de data som behövs för att skapa en rapport. Objektet returneras sedan till Report Service, och dess data lagras i databasen.



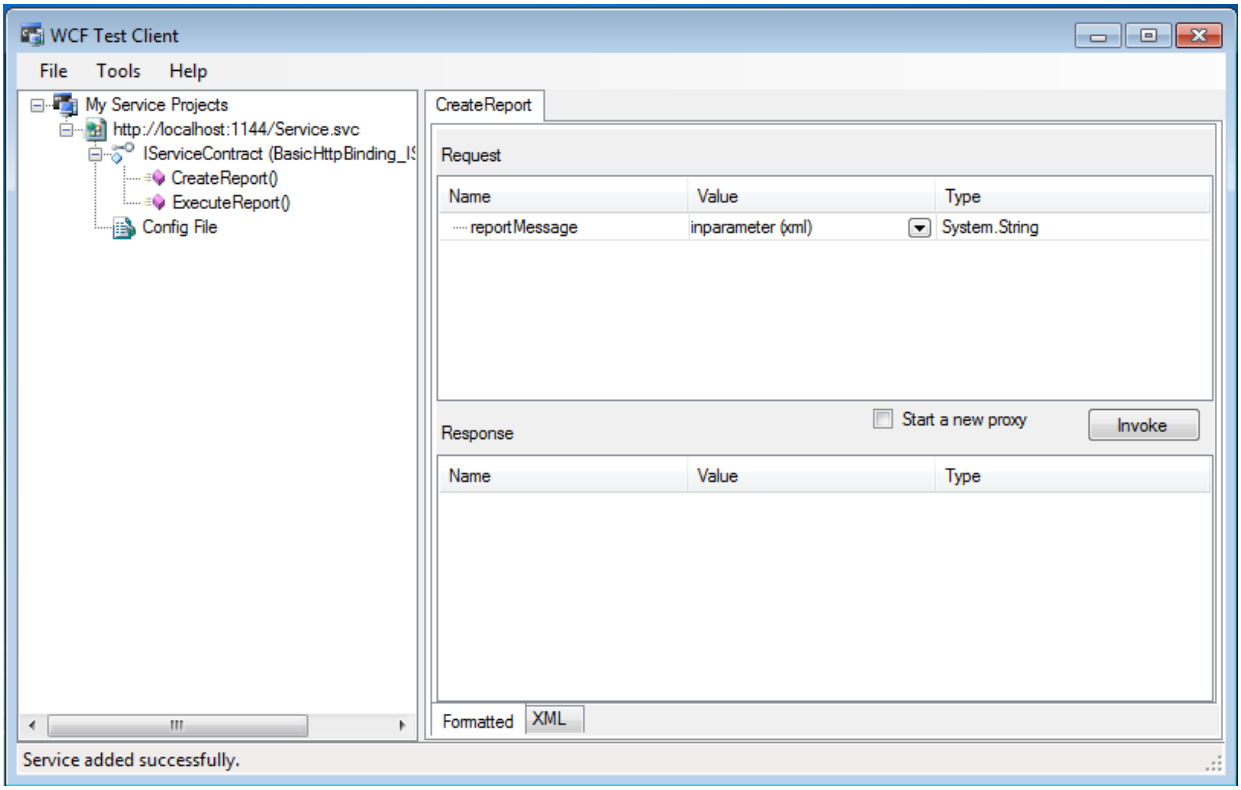

**Figur 2-8: WCF-tjänsten, översikt (del 2)**

3. Report Service skickar `ReportData`-objektet till `Dispatcher`, som vidarebefordrar det till klassen `ReportHandler`. `ReportHandler` hämtar ut en identitet (ID) ur `ReportData`-objektet och använder detta för att göra ett anrop till RS. RS använder i sin tur detta ID för att hämta de data som i steg 2 lagrades i databasen. Datat returneras som ett RDL-dokument (se avsnitt 2.2.4), som RS sedan använder för att generera en rapport i antingen formatet PDF eller TIFF. Denna rapport returneras till `ReportHandler`, som i sin tur returnerar det till `Dispatcher`.

4. `Dispatcher` skickar rapporten till de destinationer som finns angivna i `ReportData`–objektet.

### WCF Test Client

WCF Test Client (Figur 2-9) är ett verktyg som följer med utvecklingsmiljön Visual Studio 2010 och används för att testa WCF-tjänster (6). Verktyget består av ett grafiskt användargränssnitt som ger möjlighet att ansluta till en tjänst, preparera de parametrar tjänsten kräve,r samt anropa tjänsten med dessa parametrar. Verktyget tar sedan emot och visar det svar tjänsten skickar ut. Om exekveringen av tjänsten lyckades, visas en bekräftelse på detta. Om något gick fel under exekveringen, visas istället ett felmeddelande. Under vår implementation av Diamonds nya funktionalitet har vi använt oss av detta verktyg för att iterativt testa och felsöka implementationen.




**Figur 2-9: WCF Test Client.**

### 2.2.4 Reporting Services

Reporting Services (RS) (7) är ett serverbaserat mjukvarusystem som används för att generera rapporter i ett flertal olika format (8) (Figur 2-10). RS ingår som en del i Microsoft SQL Server (från och med Microsoft SQL Server 2005).

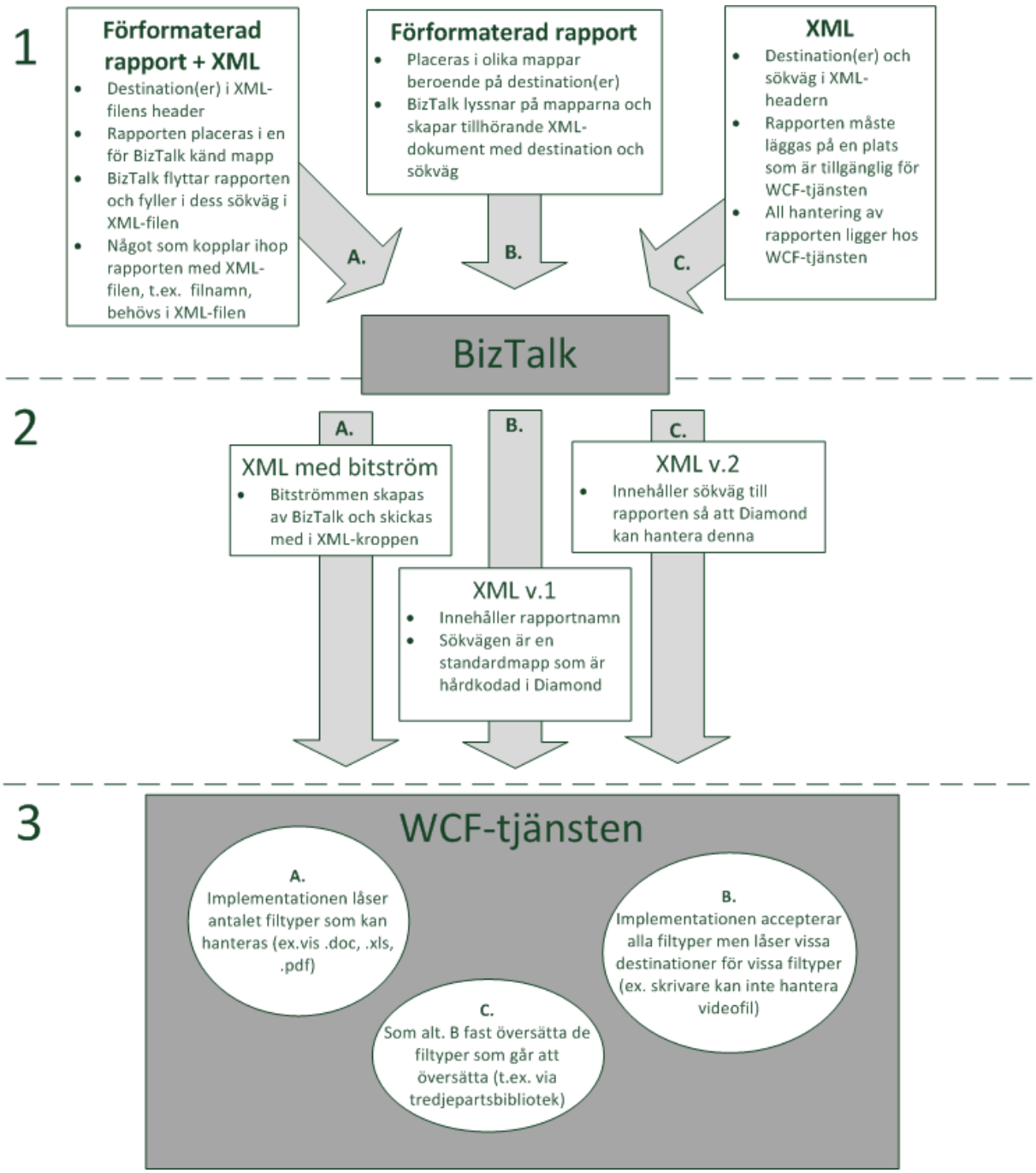

**Figur 2-10: Översiktlig bild av Reporting Services.**

RS tar via ett Webbinterface emot indata i form av ett RDL-dokument (Report Definition Language). RDL är ett RS-specifikt format baserat på XML (9). Det är med RS möjligt att generera rapporter i ett flertal olika format, bland annat PDF och TIFF.

I Diamond används RS för att, utifrån de data som skickats in till WCF-tjänsten, generera rapporter. RS använder sig för detta ändamål av rapportmallar, kallade Reporting Service Templates, och kan på så vis koppla specifika element i XML-/RDL-dokumentet till specifika fält i rapporten. Implementationen av WCF-tjänsten i Diamond gör det möjligt att, utifrån samma samling data, generera flera rapporter med olika utseende. Detta genom att specificera flera Reporting Service Templates och koppla dessa till de data man vill generera rapporter av.

# 3 Lösningsförslag

Följande avsnitt innehåller en övergripande beskrivning av de lösningsförslag vi har studerat, en utvärdering av dessa förslag, samt en utförlig motivering av valt förslag. I vår utvärdering utgick vi från det nuvarande filflödet genom Diamond. För att förenkla arbetet valde vi att dela upp flödet i tre delar och ta fram lösningsförslag för respektive del. Uppdelningen gjorde vi genom att utgå från de huvudsakliga aktiviteter som ingick i flödet: inskickandet av fil till systemets BizTalk-del, kommunikationen mellan BizTalk och WCF-tjänsten, samt WCF-tjänstens interna hantering av XML-filen. Samtliga av dessa aktiviteter behövde modifieras för att få Diamond att stödja godtyckliga dokumenttyper. Vi fann ett flertal möjliga delförslag. Dessa delförslag är, med ett undantag, oberoende av varandra.




**Figur 3-1: Översikt över lösningsförslag.**

Vi har valt att dela upp helhetsförslaget i tre delar som motsvarar de huvudaktiviteter vi fann i flödet. Delarna är numrerade från 1 till 3 (Figur 3-1). Del 1 avser definitionen av hur indatat till BizTalk ska se ut. Del 2 specificerar vilken data som ska finnas i den XML-fil som skickas från BizTalk till WCF-tjänsten. Del 3 beskriver hur WCF-tjänsten ska fungera internt med avseende på hantering av inkommande data samt distributionen densamma. Notera att det råder ett beroende mellan två av delförslagen i Figur 3-1; förslag 2-C (XML v.2) är beroende av att förslag 1-C (XML) väljs. Begreppet fil återkommer ett flertal gånger i Figur 3-1, och syftar i samtliga fall på den fil, av valfritt format, som Diamond ska kunna hantera efter vår modifikation

av systemet. Det kan exempelvis röra sig om en PDF- eller en flatfil. Denna typ av fil kommer fortsättningsvis att kallas för förformaterad rapport. Resonemanget bakom valet av begrepp är att filer av denna typ inte kommer att formateras av Diamond; istället kommer filerna att ha samma form och utseende när de skickas ut från Diamond som när de skickades in. Vi har för enkelhetens skull valt att kalla filerna för rapporter, även om det inte nödvändigtvis behöver röra sig om just rapporter. För att tydligt särskilja de förformaterade rapporterna från den andra typen av rapport, vilken genereras av RS, har vi valt att kalla den sistnämnda för RS-rapport. Notera att implementeringen av lösningsförslagen till del 1 och 2 faller utanför ramen för vårt projekt. Det finns två anledningar till att lösningsförslag ändå tagits fram till dessa delar. För det första trodde vi i början av examensarbetet att en implementering av en BizTalk-lösning skulle ingå i projektet. Detta var något som antytts i projektspecifikationen, men som senare visade sig vara överflödigt. För det andra är lösningsförslagen i del 3 beroende av valen av lösningar till del 1 och 2; med andra ord, vår implementation kommer vara beroende av innehållet i den XML-fil som WCF-tjänsten i Diamond tar emot som indata. Framtagandet av lösningsförslag till del 1 och 2 har varit viktigt för vår helhetssyn på projektet då det har fördjupat vår förståelse för hur Diamond fungerar. Lösningsförslagen i del 1 och 2 kan ses som specifikationer av den data vår version av WCF-tjänsten kräver från BizTalk-applikationen.

## 3.1 Indata till BizTalk

Följande delavsnitt presenterar de tre delförslag vi tagit fram för del 1. Förslagen avser den eller de fil(er) som skickas in i Diamonds BizTalk-lösning. Innan vår modifikation av systemet var det uteslutande XML-filer med rapportrelaterad data som skickades in i BizTalk. Den nya funktionaliteten, att låta Diamond hantera filer av varierande format, öppnar dock upp för ett antal nya möjligheter angående vad som ska skickas in i BizTalk.

### 3.1.1 Förslag 1-A, Förformaterad rapport + XML

Förslag 1-A går ut på att den inkommande rapporten placeras i en på förhand bestämd mapp som är känd av BizTalk. Den tillhörande XML-filen innehåller information om vilken distributionskanal (med andra ord, vilken destination eller samling av destinationer) rapporten ska distribueras till, samt det data som kopplar ihop XML-filen med rapporten, exempelvis filnamn. BizTalks uppgift blir enligt detta lösningsförslag att monitorera ovan nämnda mapp för att, när den förformaterade rapporten placeras där, flytta den till en annan lämplig mapp, samt preparera en ny XML-fil med det data som ska skickas vidare till WCF-tjänsten. Anledningen till att BizTalk flyttar den inkommande rapporten till en ny mapp, är att den mapp där rapporten först placeras bör hållas tom för att undvika konflikter gällande namn på inkommande rapporter. Den nya XML-fil som BizTalk preparerar åt WCF-tjänsten bör innehålla sökvägen till den inkommande rapportens nya placering.

Förslag 1-A har en tydlig styrka: man får full kontroll över rapporten så fort den kommer in i Diamonds BizTalk-del. Genom att låta BizTalk flytta rapporten kan vi försäkra oss om att den placeras på en plats till vilken WCF-tjänsten har åtkomsträttigheter. Detta är viktigt eftersom ansvaret för att öppna den förformaterade rapporten, samt läsa in dess innehåll, ligger hos WCF-tjänsten. BizTalk kan även säkerställa att rapporten har ett unikt namn genom att byta namn på

den innan den flyttas, vilket eliminerar risken för namnkonflikter mellan olika inkommande rapporter. En möjlig nackdel med förslag 1-A är att BizTalk ansvarar för ett flertal operationer, något som kan riskera att göra BizTalk-delen till en flaskhals. På grund av vår begränsade kunskap om BizTalk kan vi dock inte avgöra hur effektivt BizTalk sköter operationer av det slag som förslag 1-A föreslår. Vi kan därför inte avgöra med säkerhet om denna tänkbara nackdel är realistisk eller inte.

### 3.1.2 Förslag 1-B, Förformaterad rapport

Enligt förslag 1-B så kommer den förformaterade rapporten in i BizTalk genom att den placeras i en specifik mapp som representerar den distributionskanal som rapporten ska distribueras via. Förslaget medför med andra ord att varje distributionskanal som finns specificerad i Diamond kräver en egen mapp där inkommande rapporter kan placeras. BizTalk lyssnar av samtliga av dessa mappar. När en inkommande rapport placeras i en av mapparna, så prepararer BizTalk en utgående XML-fil med sökvägen till rapporten, samt information om aktuell distributionskanal. Det kan även vara önskvärt att låta BizTalk byta namn på rapporten innan dess sökväg sparas i den utgående XML-filen för att undvika namnkonflikter. Eftersom den inkommande rapportens placering ger information om var rapporten ska distribueras, så krävs ingen inkommande XML-fil som explicit anger detta, vilket var fallet i förslag 1-A.

En fördel med förslag 1-B är att det, till skillnad från förslag 1-A, endast kräver en inparameter till BizTalk-delen. Anledningen till detta är att den information WCF-tjänsten kräver för att kunna distribuera en förformaterad rapport, det vill säga rapportens sökväg samt distributionskanal, avgörs av den inkommande rapportens placering. Detta medför att BizTalk kan hämta in ovan nämnda information. En nackdel med lösningsförslaget är att man tvingas tillhandahålla en mapp per distributionskanal, något som kan medföra två problem. För det första kan en sådan lösning bli svår att överblicka i de fall ett stort antal distributionskanaler finns specificerade. För det andra kräver en sådan lösning att en ny mapp skapas manuellt varje gång en ny distributionskanal skapas, vilket innebär merarbete för den som administrerar systemet.

### 3.1.3 Förslag 1-C, XML

Förslag 1-C går ut på att den inkommande rapporten placeras i en valfri mapp som motsvarar sökvägen specificerad i XML-filen. Användaren av systemet ansvarar för att placera rapporten i denna mapp, samt lägga in rapportens sökväg i den XML-fil som användaren skickar in i systemet. Även information om distributionskanal behöver finnas med i XML-filen. Rapporten som ska distribueras kommer i detta fall inte skickas in i BizTalk; all hantering av rapporten sker i WCF-tjänsten. BizTalks uppgift blir att vidarebefordra den information som finns i den

inkommande XML-filen. Den XML-fil som BizTalk prepararer och skickar ut innehåller sökvägen till rapporten som ska distribueras samt information om distributionskanal. Förslaget ställer krav på att den inkommande rapporten placeras på en plats som är tillgänglig för WCF-tjänsten, då det är den som kommer hantera rapporten.

En tänkbar fördel med förslag 1-C är att BizTalk-implementationen blir förhållandevis enkel. Den enda bearbetning som kommer göras av den inkommande XML-filen är en omstrukturering

av dess innehåll, för att få den att överensstämma med det XML-schema WCF-tjänsten förväntar sig. Det potentiella problemet med förslag 1-A, att BizTalk riskerar att bli en flaskhals, elimineras därmed. En stor nackdel med förslag 1-C är att systemet inte får kontroll över rapporten som ska distribueras förrän långt fram i händelsekedjan. Skulle det vid det laget visa sig att den sökväg som finns specificerad i XML-filen är felaktig, så har mycket arbete utförts i onödan. Dessutom bidrar detta problem till att försvåra felsökning i systemet.

## 3.2 Utdata från BizTalk till WCF-tjänsten

Del 2 avser kommunikationen mellan BizTalk och Diamonds WCF-tjänst, och specificerar innehållet i den XML-fil som skickas från BizTalk till WCF-tjänsten. Eftersom WCF-tjänsten är starkt beroende av att den XML-fil som skickas in är formaterad på ett visst sätt, så finns det vissa begränsningar kring hur del 2 kan lösas. Vi har därför utgått från XML-filens ursprungliga formatering för att avgöra var vi bör lägga till de nya data som krävs. Det finns viss information som alltid måste finnas med i ovan nämnda XML-fil, exempelvis distributionskanal. Sådan information kommer inte nämnas i lösningsförslagen; beskrivningen av dessa begränsas till den information som skiljer förslagen åt.

### 3.2.1 Förslag 2-A, XML med bitström

Enligt förslag 2-A innehåller den XML-fil som skickas från BizTalk till WCF-tjänsten rapporten som ska distribueras, konverterad till en bitström. Konverteringen sköts av BizTalk. WCF-tjänsten kan sedan spara bitströmmen i databasen om detta är önskvärt. Innan distribution måste bitströmmen konverteras tillbaka till en rapport av rätt format, det vill säga samma format som rapporten hade när den skickades in i BizTalk.

Förslag 2-A är det enda förslag som medför att den förformaterade rapport som ska distribueras kan sparas i databasen. Detta är en fördel om man vid ett senare tillfälle vill distribuera rapporten igen, eftersom man då kan hämta rapporten ur databasen, konvertera den och distribuera den. Ett problem med lösningsförslaget är dock att WCF-tjänsten måste känna till rapportens ursprungliga format så att tjänsten kan konvertera bitströmmen till rätt filformat innan distribution. Detta problem kan dock lösas genom att introducera ett fält som specificerar filformatet i den XML-fil som BizTalk skickar ut. Ett allvarligare problem är att vi inte är säkra på om det är möjligt att implementera lösningsförslaget då det skulle kräva att BizTalk gör om rapporten till en bitström. En annan potentiell nackdel är att databasen kan komma att växa i storlek mycket snabbare än tidigare då den genererade bitströmmen riskerar att vara stor.

### 3.2.2 Förslag 2-B, XML v.1

Förslag 2-B innebär att den XML-fil som skickas ut från BizTalk innehåller namnet på den förformaterade rapport som ska distribueras. Sökvägen till rapporten är i detta förslag fördefinierad, och lagras exempelvis i den databas WCF-tjänsten använder sig av. Rapporten måste med andra ord finnas i den mapp sökvägen hänvisar till, och ha placerats där av antingen BizTalk eller Diamonds användare. Förslaget kräver vidare att rapportens namn är unikt för att undvika namnkonflikter.

Genom att definiera en standardmapp där den förformaterade rapporten placeras kan man enkelt försäkra sig om att WCF-tjänsten alltid har åtkomst till denna rapports placering. I teorin skulle detta kunna underlätta implementationen av vår modifikation av WCF-tjänsten. I praktiken blir denna skillnad dock marginell: validering av sökvägen krävs fortfarande eftersom rapporten kan saknas eller ha felaktigt namn i sökvägen. Standardmappen måste definieras på ett sådant sätt att den är enkel att ändra vid behov, exempelvis genom att placera den i databasen. Detta kräver dock en ombyggnad av databasen då den i sitt ursprungliga tillstånd inte har någon given tabell där liknande data kan sparas. En tänkbar nackdel med förslaget är att det inte är flexibelt med avseende på placering av rapporterna, då alla rapporter måste ligga i standardmappen. Skulle man vilja ha flera mappar att placera rapporterna i, för att exempelvis kunna sortera dem utifrån olika kriterier, så är förslag 2-B otillräckligt.

### 3.2.3 Förslag 2-C, XML v.2

Enligt förslag 2-C innehåller den XML-fil som skickas ut från BizTalk, liksom i förslag 2-B, namnet på den förformaterade rapport som ska distribueras. XML-filen innehåller enligt detta förslag dessutom sökvägen till rapporten. Denna sökväg (inklusive rapportens namn) måste vara garanterat unik för att undvika namnkonflikter.

Placeringen av rapporten är mer flexibel i förslag 2-C än i förslag 2-B. Enligt detta förslag kan rapporten placeras i vilken mapp som helst som WCF-tjänsten har åtkomst till. Det är upp till BizTalk eller användaren av Diamond att se till att rapporten placeras på en plats som WCF-tjänsten kan komma åt.

## 3.3 WCF-tjänsten internt

Del 3 avser WCF-tjänstens interna hantering av de data som genom en XML-fil skickas från BizTalk till WCF-tjänsten. Innan vår modifikation av Diamond utgjordes dessa data alltid av information utifrån vilken en RS-rapport skulle genereras. Dessa rapporter genereras antingen som en PDF eller en TIFF-fil, beroende på vilken eller vilka destination(er) rapporten ska distribueras till. Vår modifikation av Diamond medför att det är önskvärt att systemet kan hantera fler filformat än så. Vi kommer dessutom inte kunna använda den översättningsfunktion som RS erbjuder eftersom RS kräver tillgång till det data utifrån vilka en rapport ska genereras, något som inte är möjligt att realisera; istället kommer vi ha sökvägen till den förformaterade rapporten. I praktiken handlar lösningsförslagen i del 3 om att bestämma hur WCF-tjänsten ska hantera de olika tänkbara filformat som tillkommer i och med vår modifikation. Förslagen kommer beskrivas på en översiktlig nivå. Vi kommer därmed inte gå in på implementationsspecifika detaljer i beskrivningarna av förslagen.

### 3.3.1 Förslag 3-A, Lås antal filtyper

Förslag 3-A innebär att vi beslutar oss för ett antal filformat som WCF-tjänsten ska kunna hantera, exempelvis Word- och PDF-filer, och implementerar funktionalitet för att hantera dessa format. Detta medför att rapporter av andra format än de som definierats kommer inte kunna distribueras med hjälp av Diamond.

En fördel med förslag 3-A är att man får full kontroll över vilka filformat som är tillåtna i Diamond. Detta underlättar distributionen då man helt enkelt kan neka att hantera rapporter av sådana format som destinationerna inte klarar av att distribuera. En stor nackdel med förslaget är att Diamonds funktionalitet blir beroende av den destination som klarar av att distribuera minst antal filformat, då förslaget innebär att vi måste garantera att de rapporter som skickas in i Diamond kan distribueras till alla specificerade destinationer. Det skulle dessutom krävas mycket jobb för att få Diamond att acceptera ytterligare filtyper, då modifikationer skulle behöva göras såväl i databasen som i själva programkoden. Förslaget ställer dessutom krav på att användaren är medveten om, och tar ansvar för att vissa filtyper inte kan hanteras, genom att se till att rapporten som skickas in är av rätt format. Förslaget kräver slutligen att vissa filtyper översätts för att garantera att alla destinationer ska kunna hantera alla tillåtna filtyper, något som med stor förmodan medför att tredjepartsbibliotek måste användas.

### 3.3.2 Förslag 3-B, Lås destinationer för filtyper

Förslag 3-B medför att vilken filtyp som helst kan accepteras som indata. Förslaget begränsar istället vilka destinationer den förformaterade rapporten kan distribueras via, genom att undersöka dess filsuffix och blockera distribution till de destinationer som inte kan hantera motsvarande filtyp. Information om vilka filtyper en destination kan hantera kommer förslagsvis att lagras i databasen.

I och med förslag 3-B blir Diamond inte längre beroende av den destination som klarar minst antal filtyper. Vår utgångspunkt med detta förslag är att Diamond ska kunna acceptera alla filtyper som indata. Därför verkar det rimligt att se den destination som klarar minst antal filformat som ett undantagsfall snarare än det huvudsakliga fallet. Då de flesta destinationer klarar av att distribuera vilket filformat som helst så känns denna lösning mer rimlig än förslag 3-A. Lösningen är dessutom lätt att bygga ut: vill man exempelvis lägga till en ny destination, eller modifiera vilka filformat en existerande destination kan hantera, så behöver man i princip bara göra ändringar i databasen. En nackdel med förslaget är att det ställer krav på användaren då alla filformat inte kan distribueras via alla destinationer. En risk med detta är att Diamond går emot användarens förväntningar när denne exempelvis inte kan skicka ett Word-dokument till en skrivare, något man kan förvänta sig att ett system som Diamond ska klara av.

### 3.3.3 Förslag 3-C, Som B plus översättning

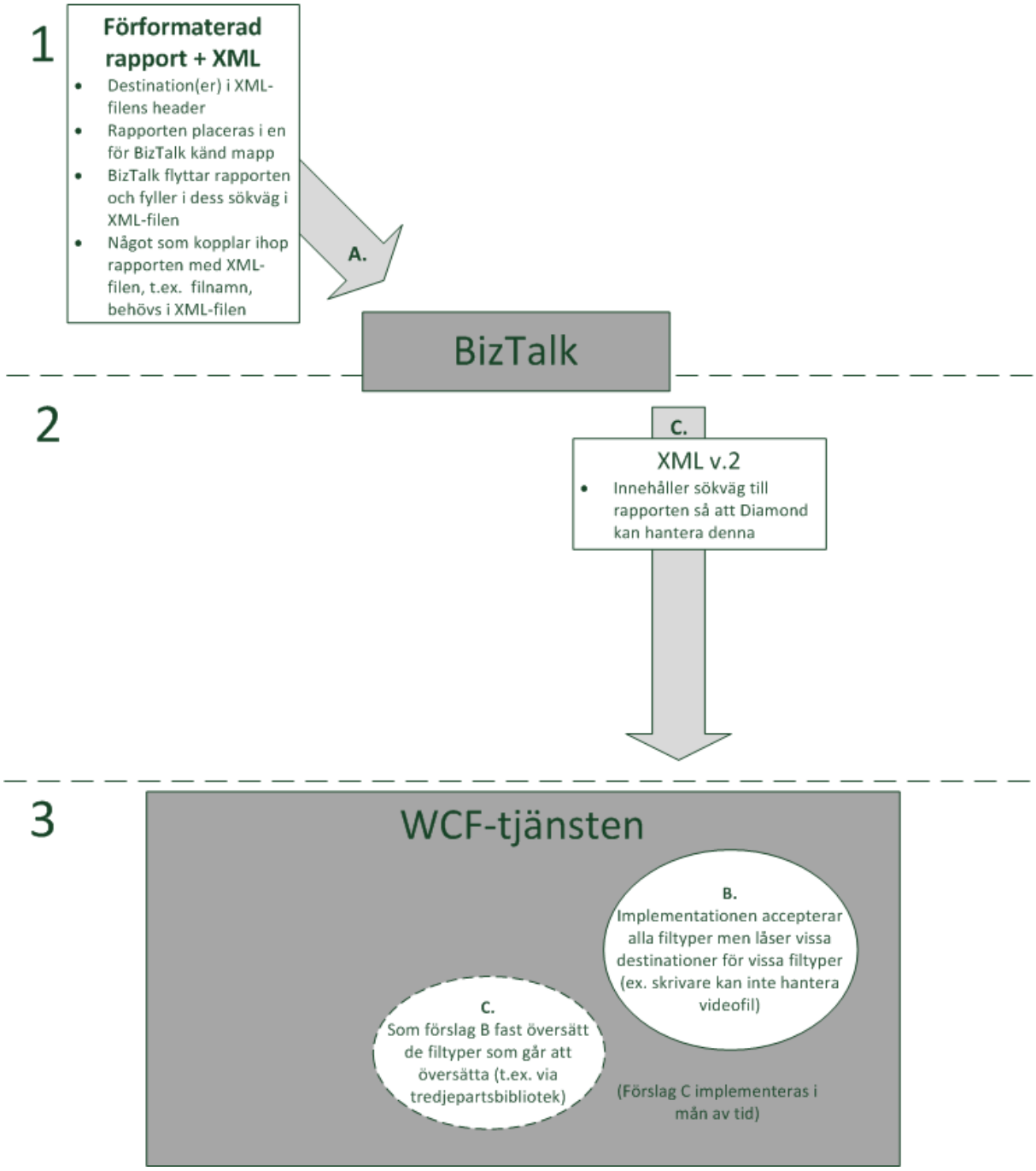
Förslag 3-C kan ses som en utbyggnad av förslag 3-B. Vad som tillkommer är att den förformaterade rapport som ska distribueras, översätts till ett för destinationen hanterbart filformat, i de fall ursprungsformatet inte går att hantera. Översättningen hanteras då av ett tredjepartsbibliotek eller någon extern tjänst. Förslaget medför att den begränsning av distributionsmöjligheter som förslag 3-B innebär försvinner.

Med förslag 3-C, så försvinner det användarrelaterade problemet i förslag 3-B. Användaren behöver i detta fall inte ta lika stort ansvar för filformatet och användarens förväntningar krockar inte med systemets funktionalitet. En nackdel med förslaget är dock att översättning krävs mellan ett stort antal filtyper, vilket med stor förmodan skulle kräva ett eller flera tredjepartsbibliotek.

Implementationen blir dessutom omfattande, då hänsyn måste tas till behovet av översättning mellan ett flertal olika filtyper.

## 3.4 Val av lösning

Som nämndes i introduktionen till detta avsnitt så kommer lösningar i del 1 och 2 inte implementeras. Vi har ändå valt att rekommendera lösningar till del 1 och 2, samt har anpassat implementationen av del 3 efter dessa rekommendationer.




**Figur 3-2: Valda dellösningar.**

Nedan följer en redogörelse över vilka dellösningar vi valt (Figur 3-2) och varför.

I del 1 har vi valt att rekommendera lösningsförslag 1-A, Förformaterad rapport + XML. Anledningen till detta är att lösningsförslag 1-A medför att Diamond får full kontroll över den förformaterade rapporten så fort den kommer in i systemet, samtidigt som vi slipper den potentiellt komplicerade mappstruktur som förslag 1-B medför. Validering av rapportens sökväg kommer av två anledningar inte krävas. För det första kommer den inkommande rapporten inte hämtas in i BizTalk om den inte placerats i den förbestämda mapp BizTalk lyssnar på, vilket automatiskt medför att rapportens sökväg kommer vara giltig när den kommer in i BizTalk. För det andra har BizTalk kontroll över att flytta rapporten till önskvärd mapp, vilket medför att även den nya sökvägen kommer vara giltig. WCF-tjänsten kan därmed lita på att den sökväg som angivits i den till tjänsten inkommande XML-filen är korrekt. Eventuella fel fångas upp så tidigt som möjligt, vilket gör systemet mer effektivt samt underlättar felsökning.

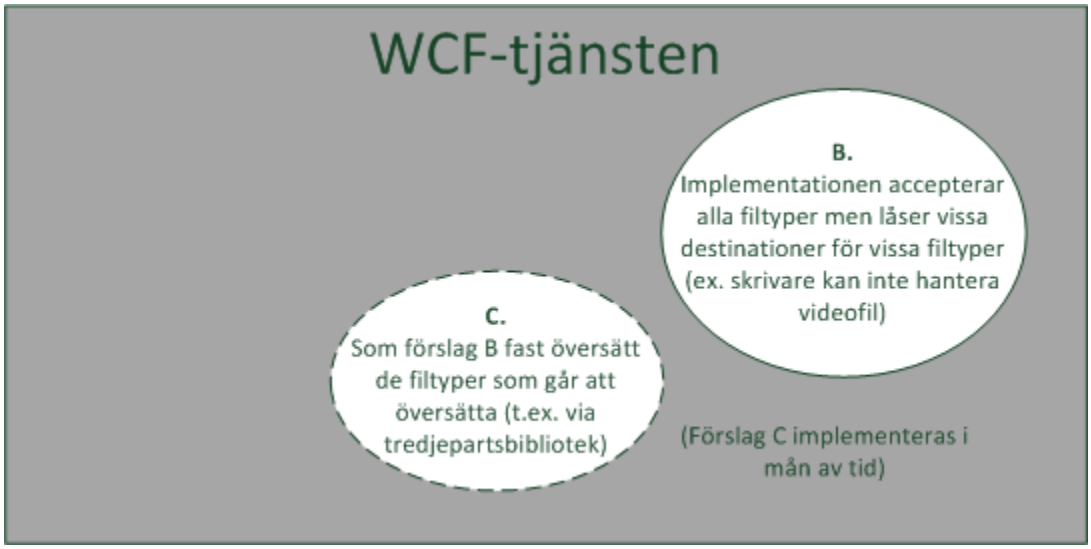
I del 2 har vi valt lösningsförslag 2-C, XML v.2. Det ur vår synpunkt bästa förslaget hade varit en kombination av förslag 1-A och förslag 2-B, XML v.1. Vi valde dock bort förslag 2-B av den anledningen att det inte finns någon given plats att definiera en standardmapp på i Diamond. Den databas som WCF-tjänsten använder innehåller i sitt ursprungliga tillstånd ingen tabell för att lagra data som är generell för tjänsten, utan endast tabeller för att lagra data relaterad till specifika rapporter, distributionskanaler och destinationer. Att definiera en standardmapp i själva programkoden i WCF-tjänsten är inte heller något bra alternativ eftersom det då krävs programmering för att byta standardmapp. Vi ser dock inga hinder för att man i framtiden skulle kunna implementera förslag 2-B genom att exempelvis utöka databasen med en tabell som innehåller generell data relaterad till WCF-tjänsten.

Det finns två anledningar till att vi valde förslag 2-C framför förslag 2-A, XML med bitström. För det första är vi, som tidigare nämnts, inte säkra på huruvida det är möjligt att få BizTalk att konvertera en fil till en bitström. För det andra så är förslag 2-C lättare att emulera än förslag 2-A och då vi inte haft tillgång till någon BizTalk-implementation att använda i testsyfte är detta en viktig faktor. För att testköra förslag C behövde vi bara skapa en XML innehållande en giltig sökväg till en rapport. För att testköra förslag 3-A skulle vi däremot vara tvungna att konvertera vår testrapport till en bitström för att sedan spara denna ström i en XML-fil, vilket skulle göra testningen till en långt mer komplicerad process.

I del 3 har vi valt lösningsförslag 3-B, att låsa destinationer för vissa filtyper. Motiveringen till detta beslut är att förslag 3-B är den mest flexibla, och på sikt, skalbara lösningen, då detta förslag medför att i stort sett alla filtyper kan accepteras som indata i systemet. Då majoriteten av systemets destinationer kan hantera vilken filtyp som helst så ser vi i nuläget ingen anledning att begränsa vilka filtyper som accepteras som indata, och därmed anser vi att förslag 3-B är bättre än förslag 3-A. Det finns dessutom goda möjligheter att i framtiden bygga ut förslag 3-B med hjälp av tredjepartsbibliotek, för att stödja översättning av de filformat som inte kan hanteras av en eller flera av systemets destinationer (förslag 3-C). Resultatet av detta skulle bli ett flexibelt system som är lätt att vidareutveckla. Det mest önskvärda förslaget är med andra ord förslag 3-C, men på grund av begränsad tid har vi valt att fokusera på förslag 3-B.

# 4 Implementation

I detta avsnitt kommer vi beskriva hur vi har implementerat den valda lösning som beskrevs i avsnitt 3. Den största utmaningen i detta projekt har varit att få Diamond att hantera och distribuera förformaterade rapporter, och samtidigt hålla den ursprungliga funktionaliteten intakt. Vi har i så stor mån som möjligt försökt att använda oss av den funktionalitet som fanns i Diamond innan vår modifikation. Anledningen till detta är bland annat att vi ville undvika att implementera vår funktionalitet som ett specialfall som körs parallellt med den tidigare implementationen. En fördel med att sammanfoga vår modifikation med tidigare existerande funktionalitet är att personer som varit bekanta med den tidigare implementationen av WCF-tjänsten kan känna igen sig i den nya implementationen.

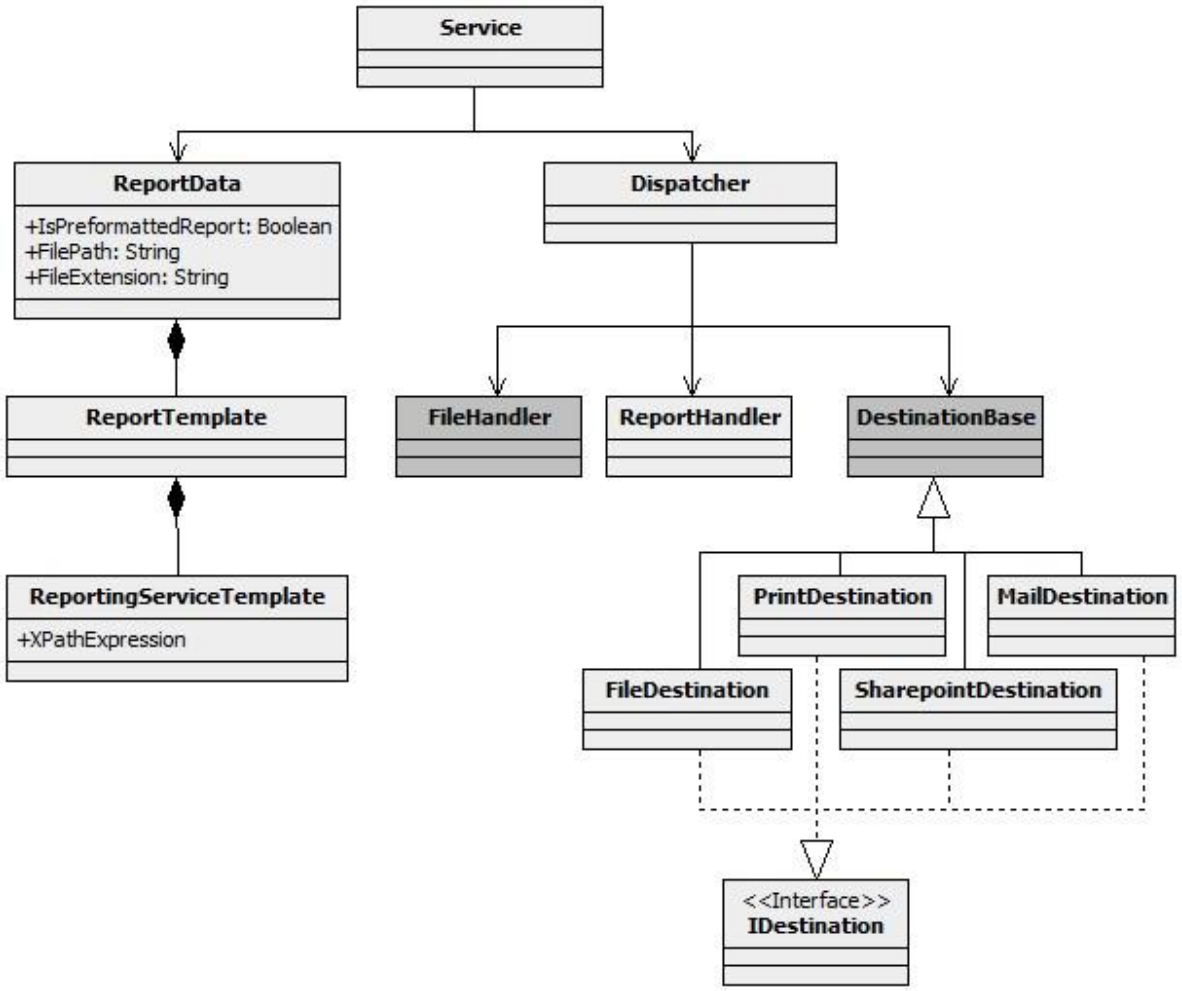



**Figur 4-1: Vald lösning för WCF-tjänsten.**

Vår implementation har begränsats till vissa specifika delar av Diamond. Framförallt är det WCF-tjänsten, den tjänst som samordnar Diamonds olika komponenter, som modifierats. Modifikationen har inneburit att utöka tjänstens funktionalitet så att den kan hantera det specialfall som distributionen av en förformaterad rapport innebär. Modifikationen har utförts enligt det lösningsförslag vi valt för del 3, nämligen 3-B, Lås destinationer för filtyper (Figur 4-1). Vi har även gjort vissa ändringar i WCF-tjänstens interna design för att anpassa dess programkod till den nya funktionaliteten. Slutligen har vårt projekt inkluderat modifikation av den SQL-databas som Diamond använder sig av, då denna i sitt ursprungliga tillstånd inte hade någon given plats för de data som den nya funktionaliteten behöver.

Avsnittet inleds med en statisk beskrivning av WCF-tjänsten, vilken beskriver tjänstens arkitektur. Den följs av en dynamisk beskrivning som beskriver tjänstens flöde. Därefter följer en detaljerad beskrivning av hur vår implementation av tjänsten fungerar. Denna beskrivning utgår från det nya användningsfall, distribution av en förformaterad rapport, som tillkommit i och med modifikationen av tjänsten. Avsnittet avslutas med våra slutgiltiga kommentarer kring implementationen, vilka bland annat innefattar rekommendationer för vidareutveckling av WCF-tjänsten.

## 4.1 Statisk beskrivning




**Figur 4-2: Klassdiagram.**

Figur 4-2 visar ett förenklat klassdiagram som representerar relevanta delar av WCF-tjänsten. Syftet med diagrammet är att beskriva de viktigaste klasserna i tjänsten samt deras roller. Alla attribut, förutom de vi har lagt till under modifikationen, har utelämnats.

Tjänsten startas genom ett anrop till klassen `Service`. Klassen `Service` har till uppgift att delegera uppgifter åt övriga delar av tjänsten, samt koordinera dessa. Den vänstra delen av klassdiagrammet visar den del av tjänsten som hanterar representationen av den rapport eller fil som ska distribueras. `ReportData` är det objekt som representerar rapporten inom tjänsten. Ett `ReportData`-objekt innehåller bland annat ett `ReportTemplate`-objekt. Detta objekt representerar, namnet till trots, inte någon mall. Objektets syfte är att associera ett `ReportData`-objekt med en eller flera Reporting Services-mallar. På detta vis kan flera

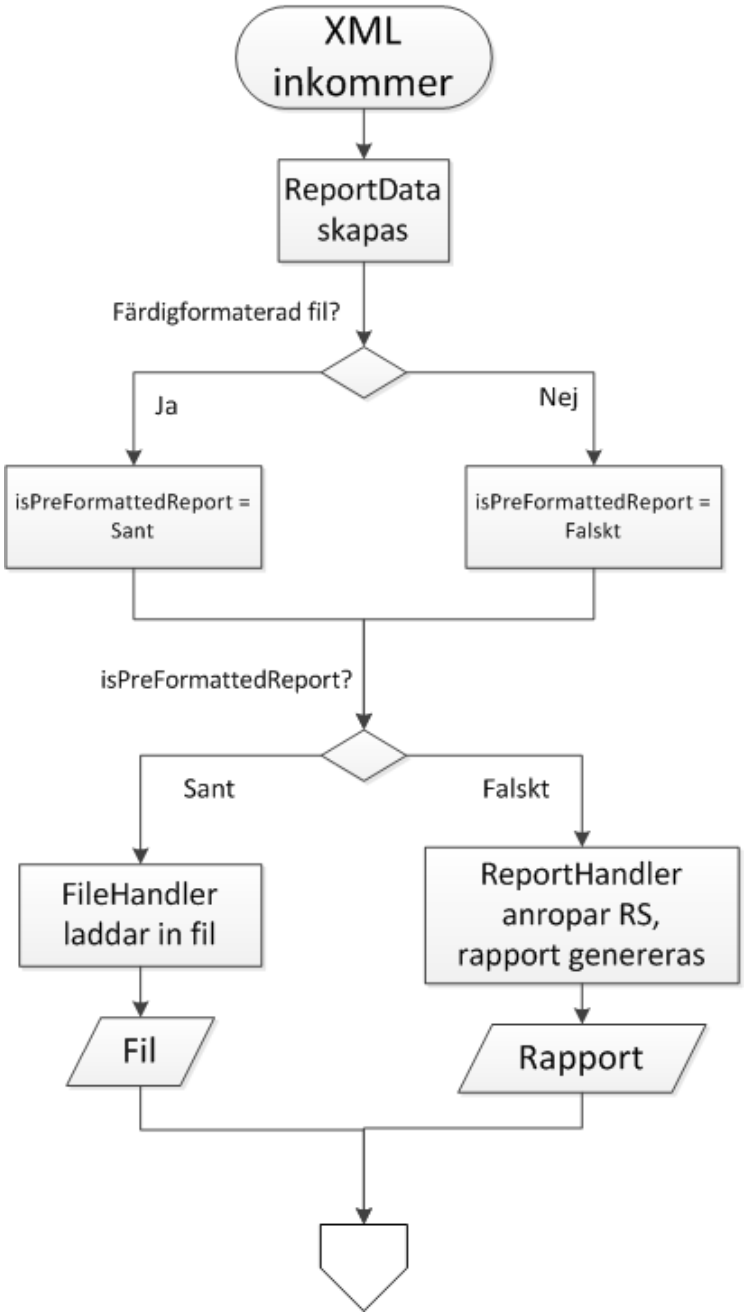
rapporter, med olika mallar, genereras utifrån samma data. Reporting Services-mallarna representeras i `ReportTemplate`-objekten av `ReportingServicesTemplate`-objekt.

Klassdiagrammets högra del visar de klasser som ansvarar för skapandet och distributionen av rapporten. `Dispatcher` är den klass som samordnar dessa uppgifter. `Dispatcher` skapar en `ReportHandler` i de fall då en rapport ska genereras. Vi har lagt till en ny klass, `FileHandler`, som anropas då en extern fil ska läsas in. Klasserna `ReportHandler` och `FileHandler` ansvarar för att förbereda rapporten eller den färdiga filen för distribution. Klassen `Dispatcher` ansvarar även för distributionen till respektive destinationsklass. Det finns i nuläget fyra olika destinationstyper. Deras motsvarande klasser publicerar ett gränssnitt med namnet `IDestination`. Klasserna ärver även av en superklass, `DestinationBase`. Denna klass fanns inte innan vår modifikation. Vi la till den för att få bort redundans som vi hittade i programkoden. Vissa funktioner var nämligen identiska i alla destinationsklasser.

`ReportData` innehåller tre nya attribut. Det första, `IsPreFormattedReport`, la vi till för att hålla reda på i fall ett `ReportData`-objekt representerar en förformaterad rapport eller en RS-rapport. Variabeln `FilePath` används för att spara sökvägen till den fil som ska distribueras, och `FileExtension` används för att spara filens suffix. Eftersom detta suffix används på ett flertal ställen så har vi valt att spara det under en egen variabel. En annan lösning hade varit att hämta ut suffixet från filens sökväg varje gång det behövdes.

I `ReportingServicesTemplate` har vi lagt till ett attribut, `XPathExpression`. Detta attribut används när sökvägen till den förformaterade rapporten hämtas ut ur den inkommande XML-filen. Vi har använt `ReportingServicesTemplate` för att avgöra om ett `ReportData`-objekt representerar en förformaterad rapport eller en RS-rapport. En mer ingående förklaring kring hur detta går till följer i avsnitt 4.3.3.

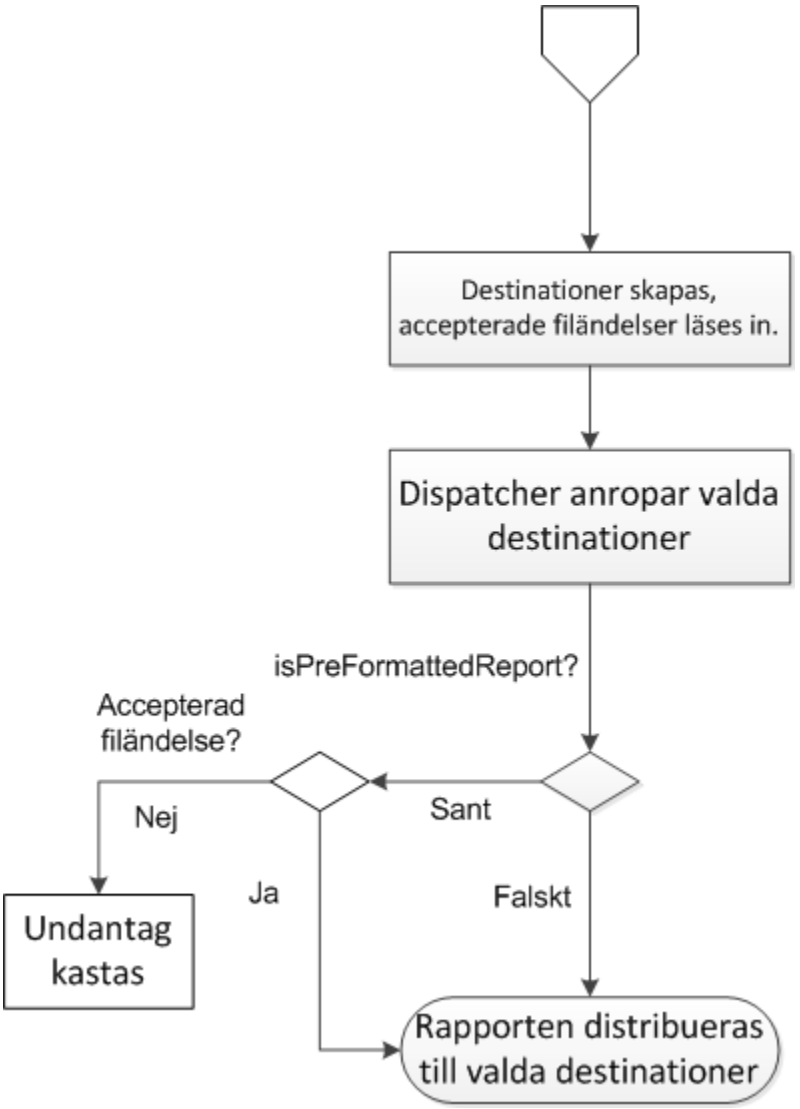
## 4.2 Dynamisk beskrivning




**Figur 4-3: Flödesdiagram del 1.**

Figur 4-3 visar flödet genom WCF-tjänsten när en XML-fil skickas in till tjänsten och dess representation inom tjänsten skapas. Först skapas ett `ReportData`-objekt som representerar den rapport eller fil som ska distribueras. Objektet innehåller den data som finns i XML-filen, samt de attribut som vi lagt till, och som nämndes i avsnitt 4.1. `ReportData` undersöks med syfte att ta reda på om det representerar en förformaterad rapport eller data som ska användas för att generera en RS-rapport. Representerar `ReportData` en förformaterad rapport sätts

`IsPreFormattedReport` till sant, annars sätts den till falskt. Om `IsPreFormattedReport` är sant så skapas en `FileHandler` som läser in rapportfilen. Är `IsPreFormattedReport` falskt så skapas istället en `ReportHandler` som med hjälp av RS genererar en rapport med önskat data.




**Figur 4-4: Flödesdiagram del 2.**

Figur 4-4 visar flödet genom WCF-tjänsten när rapporten ska distribueras. Första aktiviteten är att skapa de destinationer rapporten ska distribueras till, samt läsa in vilka filsuffix dessa kan hantera. Hanteringen av filsuffix utgör en del av vår modifikation. `Dispatcher` sköter sedan själva distributionen genom att be de valda destinationerna att distribuera rapporten. Om `IsPreFormattedReport` är sant så undersöks filsuffixet hos den rapport som ska distribueras. Är detta filsuffix accepterat av de aktuella destinationerna så distribueras rapporten till dessa destinationer. Är filsuffixet inte accepterad så genereras ett undantag. Om `IsPreFormattedReport` är falskt så distribueras rapporten till valda destinationer direkt, enligt samma tillvägagångssätt som innan vår modifikation av WCF-tjänsten.

## 4.3 Exempel: distribution av hyresfaktura

Följande beskrivning utgår från det användningsfall då WCF-tjänsten används för att distribuera en förformaterad rapport. Exemplet avser distribution av en hyresfaktura, vilken redan existerar och ligger lagrad på disk som en fil av valfritt format. Utseendet hos den XML-fil som ska användas för att distribuera hyresfakturan visas i Figur 4-5.

### 4.3.1 Den inkommande XML-filens innehåll

Det första som händer när WCF-tjänsten anropas är att den skickade XML-filen läses in. Den inlästa filen används sedan för att skapa ett `ReportData`-objekt som representerar den förformaterade rapporten, i detta fall en hyresfaktura. Det data som finns i XML-filens kropp (det vill säga i `ReportData`-elementet) är, vilket tidigare nämnts, organiserad som olika element, där varje element motsvarar en viss kategori av data. I det aktuella exemplet (Figur 4-5) innehåller XML-filens kropp tre element: ett element som specificerar sökvägen till den hyresfaktura som ska distribueras, ett som anger vad denna rapport ska heta vid distribution, samt ett som anger var rapporten ska sparas vid distribution till disk.

```
<ReportMessage>

   <Header>
      <ReportTemplateName>FärdigFakturaTemplate</ReportTemplateName>
      <DistributionName>Hyresgäster</DistributionName>
      <Created>2011-10-18</Created>
   </Header>

   <ReportData>
      <filePath>sökväg/till/hyresfaktura.filsuffix</filePath>
      <distrFileName>Hyresfaktura_20111018.filsuffix</distrFileName>
      <destinationPath>sökväg/för/output</destinationPath>
   </ReportData>

</ReportMessage>
```

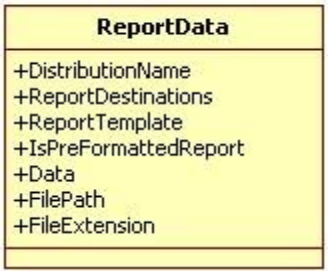
**Figur 4-5: XML för distribution av färdig fil.**

För att avgöra vilken kategori de olika elementen i XML-filen motsvarar, matchas elementens namn mot information som finns lagrad i databasen. I det aktuella exemplet antar vi att det finns information i databasen som anger att elementet med namnet `filePath` motsvarar en sökväg till den faktura som ska distribueras. Vi antar även att det finns information som anger att elementen `distrFileName` och `destinationPath` motsvarar filnamn vid distribution, samt sökväg till platsen där rapporten ska sparas. Namnet på dessa element måste inte vara samma namn som anges i exemplet, utan kan vara godtyckliga namn. Det enda kravet är att namnet överensstämmer med den information som finns lagrad i databasen. En mer detaljerad beskrivning av hur ovan nämnda matchning går till följer i delavsnitt 4.3.3. Elementen `distrFileName` och `destinationPath` är inte obligatoriska, utan kan utelämnas ur XML-filen. Detta till skillnad från sökvägselementet vilket är obligatoriskt. Den information som specificeras i de två frivilliga elementen kan även specificeras i databasen som egenskaper hos

en destination. Diamond letar först efter informationen i XML-filen, och om informationen inte finns där så hämtas istället motsvarande information från databasen.

### 4.3.2 ReportData - XML-filens representation i WCF-tjänsten

Inom WCF-tjänsten används, som sagt va,r ett `ReportData`-objekt för att representera den rapport som ska distribueras. Ett `ReportData`-objekt innehåller all information tjänsten behöver för att kunna läsa in och distribuera rapporten.




**Figur 4-6: ReportDatas attribut.**

Nedan följer en lista över attributen hos `ReportData` (Figur 4-6), samt en kort beskrivning av vad de innehåller:

- `DistributionName`: Namnet på den distributionskanal som rapporten ska distribueras via.
- `ReportDestinations`: En lista över de destinationer som ingår i ovan nämnda distributionskanal.
- `ReportTemplate`: Ett objekt som används för att associera ett `ReportData`-objekt med ett eller flera `ReportingServicesTemplate`-objekt. Syftet med detta beskrivs längre fram i rapporten.
- `IsPreFormattedReport`: En boolean som används för att avgöra om `ReportData`-objektet representerar en förformaterad rapport eller en blivande RS-rapport. Variabeln kommer i det aktuella exemplet ha värdet sant.
- `Data`: Motsvarar de data som finns inuti XML-filens kropp.
- `FilePath`: Sökvägen till den förformaterade rapport som ska distribueras.
- `FileExtension`: Filsuffixet hos den förformaterade rapport som ska distribueras

Attributet `DistributionName` laddas in från elementet med samma namn i XML-filen. Detta element finns, som tidigare nämnts, i XML-filens huvud och är ett obligatoriskt element för alla XML-filer som Diamond kan hantera. I det aktuella exemplet innehåller `DistributionName`-elementet namnet ”Hyresgäster”. Detta namn används sedan för att från databasen hämta in de destinationer som ingår i distributionskanalen med samma namn. Destinationerna sparas under attributet `ReportDestinations`. Attributet `ReportTemplate` hämtas från databasen. Detta är möjligt eftersom innehållet i elementet `ReportTemplateName`, vilket också finns med i XML-filens huvud, används för att hämta

motsvarande `ReportTemplate` från databasen. Elementet `ReportTemplateName` innehåller i det aktuella exemplet ”FärdigFakturaTemplate”, vilket med andra ord är namnet på det `ReportTemplate` som kommer läsas in från databasen.

### 4.3.3 Reporting Services Template

Ett `ReportTemplate`-objekt är associerat med ett eller flera `ReportingServiceTemplate`(s). Även dessa läses in från databasen, och sparas sedan i en lista. Anledningen till att det kan finnas flera `ReportingServicesTemplate`-objekt per `ReportTemplate`, och därmed per `ReportData`-objekt, är att man då utifrån samma data kan generera flera Reporting Services-rapporter med olika utseende. Låt säga att man till exempel vill generera en hyresfaktura och hyresfakturakopia, vilka båda ska innehålla samma information men under olika rubriker. Man kan då utnyttja ovan nämnda funktionalitet för att, utifrån samma data, generera två olika rapporter. I det aktuella exemplet kommer det dock bara skapas ett `ReportingServicesTemplate`. Anledningen till detta beskrivs längre fram i delavsnittet.

Ett `ReportingServicesTemplate`-objekt innehåller två för oss relevanta attribut. Det ena, `ReportingServiceReference`, används för att tillhandahålla en referens till den Reporting Services-mall som ska användas. I det aktuella fallet kommer `ReportingServiceReference` inte innehålla någon referens till RS, eftersom en sådan inte behövs när det är en förformaterad rapport som hanteras av WCF-tjänsten. Innan vår modifikation av tjänsten var denna referens obligatorisk. Vi valde att istället göra den frivillig, med andra ord att tillåta att `ReportingServiceReference` kan sakna värde. Detta medför att attributet kan användas för att avgöra om det är en förformaterad rapport eller en RS-rapport som hanteras av tjänsten. Hur detta går till i praktiken kommer att beskrivas längre fram i avsnittet.

Det andra attributet av intresse i ett `ReportingServicesTemplate`-objekt är `XPathExpression`, en variabel som vi har lagt till under vår modifikation. `XPathExpression` är en klass som ingår i .NET 4 (10). Instanser av denna klass används för att definiera innehållet i en XML-fil, eller närmare bestämt vilka element filen kan innehålla, samt hur dessa är nästlade inuti varandra (11).

```
<ReportMessage>

   <Header>
      <ReportTemplateName>FärdigFakturaTemplate</ReportTemplateName>
      <DistributionName>Hyresgäster</DistributionName>
      <Created>2011-10-18</Created>
   </Header>

   <ReportData>
      <filePath>sökväg/till/hyresfaktura.filsuffix</filePath>
      <distrFileName>Hyresfaktura_20111018.filsuffix</distrFileName>
      <destinationPath>sökväg/för/output</destinationPath>
   </ReportData>

</ReportMessage>
```

**Figur 4-7: XML-fil med sökvägselementet markerat.**

I WCF-tjänsten används attributet `XPathExpression` för att specificera var i XML-filen man kan hitta sökvägen till den förformaterad rapport som ska läsas in och distribueras. Värdet hos attributet läses in från motsvarande kolumn i databasen. I det aktuella exemplet (Figur 4-7), där XML-filen innehåller ett `filePath`-element som är nästlad inuti elementet `ReportData`, måste motsvarande `XPathExpression` innehålla strängen `"/ReportData/filePath/text()"`. Strängen anger att det inuti elementet `ReportData` finns ett nästlat element med namnet `filePath`. Uttrycket `"text()"` i strängen anger att all text som ligger inuti `filePath` ska hämtas. Notera att ingen validering av värdet hos `XPathExpression` sker i detta läge. Skulle XML-filen sakna `filePath`-element så kommer det aktuella `ReportingServicesTemplate`-objektets `XPathExpression` inte få något värde. Konsekvenserna av detta beskrivs i delavsnitt 4.3.5.

### 4.3.4 Metadata

När ett `ReportData`-objekt, dess motsvarande `ReportTemplate` och listan med `ReportingServicesTemplates` har skapats, sker inläsning av metadata[4] från databasen. Dessa metadata är knutna till det eller de `ReportingServicesTemplate`(s) som är associerade med det specifika `ReportData`-objektet. Metadatan består av `XPathExpressions` som beskriver vilka övriga element den inkommande XML-filens kropp innehåller. I databasen finns metadatan lagrat i en egen tabell, vilken kopplas ihop med `ReportingServicesTemplate`-tabellen med hjälp av en främmande nyckel. Det är viktigt att inte blanda ihop metadatans `XPathExpressions` med `XPathExpression` för `ReportingServicesTemplates`, då dessa finns lagrade på olika platser och används i olika syften. Metadatans `XPathExpressions` läses in från databasen som en lista och

[4] Metadata kan definieras som data som beskriver annan data.

används sedan för att hämta ut data ur XML-filen. Den hämtade datan sparas sedan till motsvarande `ReportingServicesTemplate`-objekt. På detta vis kan man enkelt specificera det önskade innehållet hos en XML-fil genom att i databasen ange vilka element som kan ingå.

```
<ReportMessage>

   <Header>
      <ReportTemplateName>FärdigFakturaTemplate</ReportTemplateName>
      <DistributionName>Hyresgäster</DistributionName>
      <Created>2011-10-18</Created>
   </Header>

   <ReportData>
      <filePath>sökväg/till/hyresfaktura.filsuffix</filePath>
      <distrFileName>Hyresfaktura_20111018.filsuffix</distrFileName>
      <destinationPath>sökväg/för/output</destinationPath>
   </ReportData>

</ReportMessage>
```

**Figur 4-8: XML-fil med de metadata-relaterade elementen markerade.**

I det aktuella exemplet innehåller XML-filen två element utöver det obligatoriska sökvägselementet, nämligen: `distrFileName` och `destinationPath` (Figur 4-8). För att XML-filen ska kunna hanteras av Diamond behöver ovan nämnda metadata innehålla ”`/ReportData/distrFileName/text()`” och ”`/ReportData/destinationPath/text()`”. WCF-tjänsten matchar sedan innehållet i XML-filen mot metadatan och får på så vis reda på var i XML-filen respektive data (filnamn vid distribution och sökväg till plats där filen ska sparas) finns placerad. Skulle XML-filen innehålla ett eller flera element som inte finns representerade bland metadatan så kommer dessa elements data inte läsas in av WCF-tjänsten.

### 4.3.5 IsPreFormattedReport

Hittills har WCF-tjänsten inte behövt (eller kunnat) avgöra om det aktuella `ReportData`-objektet som hanteras representerar en färdig fil eller inte. När inläsningen av den eller de `ReportingServicesTemplate`(s) som är associerade med `ReportData`-objektet är klar så blir det dock möjligt för tjänsten att avgöra just detta. En kontroll sker av det första `ReportingServicesTemplate`-objektet i listan, för att se om detta objekt har någon referens till Reporting Services eller inte. I det aktuella exemplet kommer det inte finnas någon sådan referens eftersom `ReportData`-objektet representerar en färdig fil. WCF-tjänsten går då över till att undersöka `ReportingServicesTemplatets XPathExpression`-attribut för att se ifall detta attribut har något värde. I det aktuella fallet antas attributet ha värdet ”`/ReportData/filePath/text()`”. Om attributet inte har något värde så genereras ett undantag, eftersom avsaknaden av Reporting Services-referens medför att `XPathExpression` måste ha ett värde. Har `XPathExpression` däremot ett värde så görs antagandet att detta

värde är en sträng som hänvisar till det element där filens, i detta fall hyresfakturans, sökväg finns. Detta `XPathExpression` används sedan för att hämta ut aktuell sökväg ur XML-filen. `ReportData:s` variabel `IsPreFormattedReport` sätts därefter till sant (Tabell 4-1).

| Värde hos XPathExpression | Värde hos RS-referens | Resultat |
|---|---|---|
| Sökväg | Referens/(inget) | IsPreFormattedReport = sant |
| (inget) | (inget) | Undantag genereras |
| (inget) | Referens | IsPreFormattedReport = falskt |

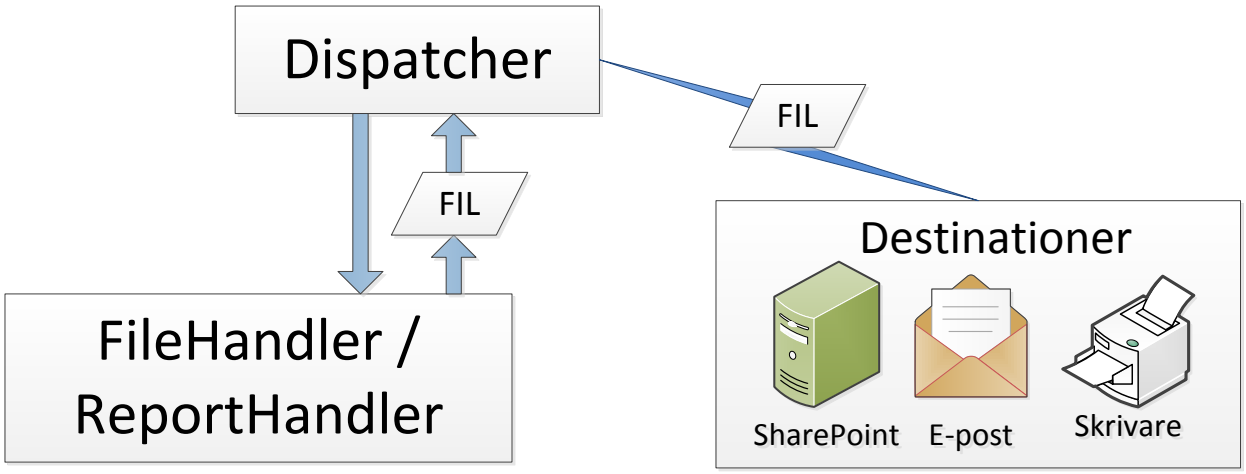
**Tabell 4-1: Relation mellan XPathExpression och RS-referens.**

Det finns två anledningar till att en kontroll bara görs av det första av alla `ReportingServicesTemplates` i listan. För det första så finns det i nuläget bara stöd för att distribuera en förformaterad rapport per inkommande XML-fil, vilket medför att det alltid finns ett och endast ett `ReportingServicesTemplate` per `ReportData`-objekt vid distribution av en fil. Det är i teorin möjligt att modifiera WCF-tjänsten så att flera filer, med olika `ReportingServicesTemplates,` och därmed olika metadata, kan distribueras med en enda XML-fil men denna funktion är i nuläget inte implementerad. För det andra är det, vid distribution av en eller flera Reporting Services-genererade rapporter, underförstått att alla associerade `ReportingServicesTemplates` måste ha en referens till Reporting Services. Skulle det första av dessa templates sakna en sådan på grund av felaktigheter i databasen, så fångas detta fel med stor sannolikhet upp när `XPathExpression`-variabeln undersöks. Denna variabel antas nämligen sakna värde i de fall då det är en Reporting Services-genererad rapport som ska distribueras. Detta, i kombination med att avsaknad av både Reporting Service-referens och `XPathExpression`-värde genererar ett undantag, gör så att felet fångas upp. Skulle det istället vara så att ett eller flera övriga associerade `ReportingServiceTemplates` saknar Reporting Service-pekare så kommer detta fel fångas upp senare under exekveringen av WCF-tjänsten.

`ReportData`-objektet är nu färdigt och sparas till databasen, varifrån dess data senare hämtas av Reporting Services. Efter att `ReportData` och dess associerade objekt har skapats och sparats till databasen, är det bara inläsning av filen samt distribution som återstår.

### 4.3.6 Distribution

`Dispatcher` är den klass i WCF-tjänsten som koordinerar distributionen av rapporter och filer. Det är denna klass som ser till att rapporter genereras, alternativt filer läses in, och skickas till angivna destinationer (Figur 4-9).



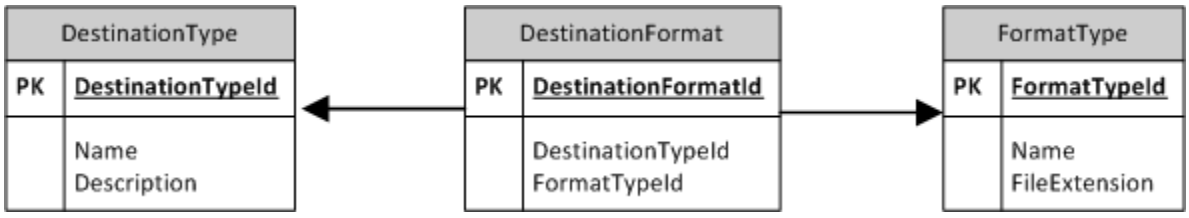

**Figur 4-9: Dispatchers roll.**

I `Dispatcher` sker en kontroll som, med hjälp av variabeln `IsPreformattedReport`, avgör om det rör sig om en färdig fil eller en Reporting Services-genererad rapport som hanteras. I det aktuella exemplet har `IsPreformattedReport` värdet sant, eftersom det rör sig om en färdig hyresfaktura som ska distribueras. Klassen `FileHandler` innehåller bland annat funktionen `LoadExistingFile`, vilken med hjälp av den i `ReportData` angivna sökvägen läser in filen som ska distribueras. Filen görs om till en bitström och sparas i `ReportData`-objektet, vilket sedan returneras till `Dispatcher`. Hade det istället rört sig om en Reporting Services-genererad rapport som skulle distribueras, så skulle det varit klassen `ReportHandler` som användes. `ReportHandler`s roll hade då varit att, utifrån angiven data, låta RS generera en rapport som sedan förbereds för distribution och returneras till `Dispatcher`.

När inläsningen av hyresfakturan är klar så skickas dess motsvarande `ReportData`-objekt vidare till de destinationsklasser som motsvarar fakturans destinationer.

I destinationsklasserna har vi implementerat en kontroll som avgör om filformatet hos filen som ska distribueras stöds av den aktuella destinationen. Detta görs genom att man matchar det filsuffix som finns sparat i `ReportData`-objektet mot en lista över filformat som stöds av den aktuella typen av destination. Varje destinationstyp har en egen sådan lista, vilken läses in från

databasen. För att kunna implementera denna funktion har vi lagt till två tabeller i databasen: `FormatType` och `DestinationFormats` (Figur 4-10). `FormatType` innehåller olika filtyper, representerade med ett namn och ett filsuffix. `DestinationFormats` kopplar ihop en destinationstyp med det eller de filformat destinationstypen ska kunna hantera.




**Figur 4-10: Utbyggnad av databasen.**

Alternativet till att spara denna information i databasen hade varit att i varje destinationsklass explicit ange de filformat destinationen ska acceptera. Vi ansåg dock att en sådan lösning inte är hållbar i längden, eftersom den kräver modifikation av programkoden varje gång man vill ändra vilka filtyper en destinationstyp kan hantera. Med den lösning vi valt så krävs bara en ändring i databasen för att uppnå samma resultat.

Om filformatet hos den fil som ska distribueras stöds av den aktuella destinationen så distribueras filen. Stöds inte filformatet så genereras ett undantag. Om vi antar att hyresfakturan i det aktuella exemplet ska distribueras till en skrivare samt sparas på disk. Vidare, om vi antar att fakturan kommer att få filnamnet ”Hyresfaktura_20111018.filsuffix” vid distributionen - detta finns specificerat i XML-filen – så sker en kontroll som undersöker om skrivardestinationen kan hantera filsuffixet ”.filsuffix”. Kan detta filsuffix hanteras så distribueras fakturan till skrivaren, annars genereras ett undantag. Vid distribution till disk spelar filformatet ingen roll. Vilket filformat som helst kan distribueras på dessa vis. Vi har därför lagt till ett filformat som vi kallar för ”any”, vilket representerar alla tänkbara filformat, och angivit detta som accepterat format vid distribution till disk. Efter en kontroll som bekräftar att fildestinationen (vilket är den destination som motsvarar distribution till disk) klarar av suffixet ”.filsuffix” så sparas fakturan till den sökväg som finns angiven i XML-filen, nämligen ”sökväg/för/output”.

## 4.4 Vidareutveckling – lösningförslag 3-C

När vi var klara med implementationen av lösningsförslag B för WCF-tjänsten (se delavsnitt 3.3.2) insåg vi att användbarheten hos denna lösning var mycket begränsad i de fall då man vill distribuera en rapport till skrivare. Det enda som går att distribuera till skrivaren med lösning 3-B är bilder. Vi började därför skissa på en vidareutveckling, i enlighet med förslag 3-C (delavsnitt 3.3.3). Inledningsvis fokuserade vi på att hitta ett sätt att konvertera en PDF till en eller flera bilder, eftersom det är just bilder som skickas till skrivaren vid anrop till denna. Vi hittade ingen funktionalitet för detta i C# men fann däremot en mängd tredjepartsbibliotek varav de flesta fanns som gratis testversioner. Vi valde ett av dessa bibliotek, ”PDF Rasterizer.NET” (12), och implementerade förslag 3-C med detta bibliotek.

Vid implementationen skedde modifikationer av klassen `PrintDestination`. Vi lade bland annat till en ny funktion för utskrift, vilken anropas i de fall då det handlar om en förformaterad rapport som ska skrivas ut. I denna funktion finns även en kontroll som undersöker om rapporten är av typen PDF, och om så är fallet så används ”PDF Rasterizer.NET” för att konvertera PDF-filen till en eller flera bildfiler. Dessa filer skickas sedan till skrivaren för utskrift.

Då ”PDF Rasterizer.NET” inte är gratis vid användning i kommersiella projekt så ska denna implementation endast ses som ett exempel på hur ovan nämnda funktionalitet kan implementeras i Diamond.

För att slippa användandet av tredjepartsbibliotek så började vi undersöka andra sätt att lösa ovan nämnda problem. Vi kom fram till att från WCF-tjänsten göra ett anrop till ett separat program som kan skriva ut PDF-filer, och på detta sätt utnyttja denna funktion. Vi gjorde en prototyp-implementation där ett anrop görs till Adobe Acrobat Reader (AAR) när en PDF ska skrivas ut. Vi försökte i denna implementation få AAR att startas i så kallat ”silent mode”, vilket innebär att programmet körs i bakgrunden utan något grafiskt gränssnitt. Vi lyckades dock inte få AAR att startas på detta vis. Det fanns inte heller tid för att, inom examensarbetets tidsram, fördjupa sig mer i hur man åstadkommer detta. Av den anledningen gick vi inte vidare med implementationen av lösningen.

# 5 Diskussion och erfarenheter

Projektet att utöka Diamond så att det stöder godtyckliga filtyper har inneburit en hel del problem och utmaningar. Den första utmaningen vi stötte på var att skaffa en översikt över, och förståelse för, WCF-tjänstens befintliga programkod. Denna kod var uppdelad i ett stort antal klasser, vilka i sin tur hade mer eller mindre komplicerade relationer till varandra. Någon dokumentation över tjänsten fanns inte. För att kunna ta fram rimliga lösningsförslag var vi tvungna att ha en övergripande förståelse för hur tjänsten fungerade, vilket medförde att vi i början av projektet fick ägna mycket tid på att studera programkoden. Vi kom till en punkt då vi ansåg oss redo att undersöka potentiella lösningsförslag. Detta innebar inte att vi vid det laget hade full förståelse för hur WCF-tjänsten fungerade vid exekvering. Denna förståelse fick vi istället utveckla successivt, i samband med implementationen av det valda lösningsförslaget.

En annan stor utmaning har varit att designa och implementera en lösning som säkerställer den önskade funktionaliteten utan att vara alltför komplex. Då vi hade begränsat med tid, och dessutom begränsad förståelse för detaljerna kring hur WCF-tjänsten fungerade, så fokuserade vi på att hitta enkla implementationslösningar. Vi modifierade sedan dessa lösningar för att effektivisera dem, och göra dem mer lättförstådda för en utomstående. Genom att arbeta på detta sätt fick vi en möjlighet att bygga upp vår förståelse för tjänsten gradvis, och fokusera på en liten del av tjänsten i taget. Vidare kunde vi kompensera för bristen på dokumentation. Det hjälpte oss dessutom att undvika att lägga för mycket tid på att försöka förstå de mindre relevanta detaljerna kring hur tjänsten fungerar.

Under implementationen har vi strävat efter att, i så stor utsträckning som möjligt, integrera vår funktionalitet med den befintliga funktionaliteten. Vi ville undvika att göra våra modifikationer till specialfall. Med andra ord, funktionaliteten för att distribuera en förformaterad rapport skulle vara en lika självklar del av WCF-tjänsten som den befintliga funktionaliteten. Vårt mål blev därmed att försöka återanvända så mycket som möjligt av denna funktionalitet. Detta ledde till att vi tvingades göra vissa kompromisser vid implementationen. Vi hade bland annat inte tänkt ha något `ReportingServicesTemplate` i de fall då det är en förformaterad rapport som hanteras. Anledningen till detta är att Reporting Services inte anropas vid hantering av sådana rapporter, vilket får förekomsten av `ReportingServicesTemplates` att kännas förvirrande. Det visade sig dock vara mer eller mindre nödvändigt att använda dem, då viktiga delar av den befintliga funktionaliteten var beroende av dem. Det skulle i teorin varit möjligt att kringgå detta beroende, något som dock hade krävt en omstrukturering av stora delar av WCF-tjänsten. En sådan omstrukturering ingick inte i vårt projekt, och fick dessutom inte plats inom projektets tidsram. För att slippa strukturera om tjänsten skulle man istället kunna byta namn på `ReportingServicesTemplate` till något mer generellt, eftersom det i och med vår modifikation representerar mer än bara Reporting Service-mallar.

Även efter modifikationen av Diamond finns det en betydande skillnad mellan hanteringen av förformaterade rapporter och hanteringen av RS-rapporter. Denna skillnad består i att RS-rapporter när som helst kan, utifrån sparat data i databasen, återskapas och distribueras på nytt. Detta är inte möjligt vid hantering av förformaterade rapporter, eftersom dessa inte sparas till

databasen. Sökvägen till rapporterna sparas visserligen, men det kan inte garanteras att dessa fortfarande är giltiga efter att rapporterna distribuerats. Ovan nämnda funktionalitet kan dock imiteras, antingen genom att man ser till att hålla sökvägarna i databasen giltiga eller genom att de förformaterade rapporterna sparas på disk, SharePoint eller liknande. Rapporterna kan på detta vis hämtas och distribueras på nytt vid behov.

Den viktigaste erfarenheten vi tagit med oss från projektet är en fördjupad förståelse för de problem det innebär att modifiera ett befintligt system. Ett sådant projekt innebär en serie utmaningar, till exempel att designa och implementera effektiva och väl integrerade modifikationer, att behålla befintlig funktionalitet intakt, samt att följa de designprinciper systemet är implementerat utifrån. Detta kräver förståelse för resonemanget bakom den befintliga programkodens arkitektur. Då modifikation och underhåll av befintliga system tycks vara vanliga aktiviteter inom den bransch vi valt så kommer vi med stor förmodan ha nytta av dessa erfarenheter i det framtida arbetslivet.

# 6 Sammanfattning och slutsatser

Uppgiften i detta examensarbete har varit att utöka funktionaliteten i det befintliga rapport- och distributionssystemet Diamond. Från början hanterade systemet enbart rapporter som, utifrån en inkommande XML-fil, genererades av Reporting Services (RS). Målet med vårt arbete har varit att få systemet att även hantera så kallade förformaterade rapporter. Med förformaterade rapporter menas rapporter som redan skapats av ett utomstående system eller en användare. Resultatet av examensarbetet är ett system där man kan använda de tidigare implementerade distributionskanalerna för att distribuera rapporter av valfritt format.

Vi valde en lösning som innebär att man i den ingående XML-filen specificerar en sökväg till den förformaterade rapport som ska distribueras via Diamond. Utöver denna tillagda information måste man även specificera en speciell mall (`ReportTemplate`) som är skapat för att hantera förformaterade rapporter. Denna mall används bland annat för att avgöra om den rapport som hanteras av WCF-tjänsten är en RS-rapport eller en rapport som ska genereras av RS. I de fall då en rapport ska genereras avgör ovan nämnda mall hur rapporten ska se ut. Rör det sig om en förformaterad rapport används denna mall istället för att hämta ut sökvägen till rapporten.

Vi skapade en ny klass som ansvarar för att läsa in den förformaterade rapporten och förbereda den för distribution. När den förformaterade rapporten är inläst så hanteras den på samma sätt som en av RS genererad rapport. För att avgöra om filformatet hos den rapport som ska distribueras accepteras av de destinationer den ska skickas till så var vi tvungna att lägga in en kontroll för det. Denna kontroll implementerades i varje destinationsklass. För att lagra vilka filformat de olika destinationerna kan hantera har vi byggt ut databasen och gjort plats för denna information.

Efter att ovan nämnda implementation var klar, fortsatte vi med att göra en prototyp-implementation enligt lösningsförslag 3-C (se avsnitt 3.3.3). Vi skapade närmare bestämt funktionalitet för att skriva ut PDF-dokument i WCF-tjänsten, med hjälp av ett tredjepartsbibliotek vid namn ”PDF Rasterizer.NET” (12). Vi provade dessutom att göra en implementation där vi lät tjänsten anropade ett utomstående program, ”Adobe Acrobat Reader”, för samma ändamål.

Funktionaliteten för att skriva ut PDF-filer, samt på sikt andra filformat, skulle med fördel kunna vidareutvecklas. Vår rekommendation till Sogeti är att köpa in ett tredjepartsbibliotek, eller hitta ett fristående program, som konverterar PDF-filer samt övriga önskade filformat till bilder. Hittar man ett sätt att konvertera PDF-filer till bilder så kan all den funktionalitet för utskrift som redan finns implementerad i Diamond användas. Implementationen där ”PDF Rasterizer.NET” och AAR används är enbart gjord för att statuera ett exempel på hur problemet med PDF-utskrift kan lösas. Att valet av tredjepartsbibliotek föll på just ”PDF Rasterizer.NET” innebär inte att detta bibliotek är det bästa för ändamålet. Anledningen till att vi valde detta bibliotek är att vi fann enkla och tydliga exempel på hur det kan användas i en implementation.

En utvärdering av, och jämförelse mellan, andra befintliga tredjepartsbibliotek uppmuntras därmed. Några exempel på sådana bibliotek är följande:

- GhostScript (13)
- PDF2Image (14)
- GdPicture.NET SDK (15)
- DynamicPDF Rasterizer for .NET (16)

En annan tänkbar vidareutveckling skulle kunna vara att implementera ett grafiskt gränssnitt som gör det möjligt att testa Diamonds distributionsfunktionalitet. Med hjälp av ett sådant gränssnitt skulle systemet enkelt kunna demonstreras för potentiella kunder som saknar djupgående kunskap om datorer och system.

Sammanfattningsvis så är vi på det stora hela nöjda med vårt examensarbete. Vi har lyckats åstadkomma en lösning som uppfyller de specificerade kraven. Diamond kan i och med vår modifikation hantera förformaterade rapporter, och vår kod är väl integrerad med den tidigare befintliga programkoden. Projektet har varit lärorikt och utvecklande, och dess omfattning har varit realistisk med hänsyn till den tidsram vi hade. Att ha arbetat med att modifiera befintlig programkod, som är utvecklad av annan part, känns som en nyttig erfarenhet. Det har även varit intressant att få arbeta på plats hos en uppdragsgivare, i detta fall Sogeti, då det gav en insikt i hur en framtida arbetssituation som systemutvecklare skulle kunna te sig.

# 7 Referenser


1. **Graves, Mark.** *Designing XML Databases.* s.l. : Prentice Hall, 2002. 0-13-088901-6.

2. **Young, Michael J.** *XML Steg för steg.* u.o. : Pagina, 2002. 91-636-0713-1.

3. **Woolston, Daniel.** *Foundation of BizTalk Server 2006.* 2007. ISBN-13 (pbk): 978-1-59059-775-0.

4. **Bustamante, Michele Leroux.** *Learning WCF.* u.o. : O'Reilly Media, 2007. ISBN 978-0-596-10162-6.

5. **MacKenzie, Matthew, o.a.** *OASIS.* [Online] den 2 Augusti 2006. [Citat: den 4 September 2011.] http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf. soa-rm-cs.

6. WCF Test Client. *MSDN.* [Online] Microsoft. [Cited: Januari 10, 2012.] http://msdn.microsoft.com/en-us/library/bb552364.aspx.

7. **Al., Paul Turley et.** *Professional SQL Server reporting services.* s.l. : Wrox Press Ltd, 2004. 9780764576942.

8. SQL Server Reporting Services. *Microsoft TechNet.* [Online] Microsoft. [Citat: den 4 September 2011.] http://technet.microsoft.com/en-us/library/ms159106.aspx.

9. Introducing Reporting Services Programming. *Microsoft MSDN.* [Online] Microsoft. [Citat: den 4 September 2011.] http://msdn.microsoft.com/en-us/library/aa237738%28v=sql.80%29.aspx.

10. XPathExpression Class. *Microsoft Developer Network.* [Online] Microsoft. [Cited: December 19, 2011.] http://msdn.microsoft.com/en-us/library/system.xml.xpath.xpathexpression.aspx.

11. **Atul, Kahate.** *XML and related technologies.* s.l. : Pearson Educational, 2009. 978-81-317-1865-0.

12. PDFRasterizer.NET Overview. *Tall Components.* [Online] Tall Components. [Cited: Januari 10, 2012.] http://www.tallcomponents.com/pdfrasterizer3-overview.aspx.

13. Ghostscript. *Ghostscript.com.* [Online] [Cited: December 20, 2011.] http://ghostscript.com/.

14. PDF2Image. *PDFTRON.* [Online] PDFTron Systems. [Cited: December 20, 2011.] http://www.pdftron.com/pdf2image/?gclid=CIi-_sOYkK0CFdAumAodRxsOlQ.

15. *GdPicture Imaging Technologies.* [Online] [Cited: December 20, 2011.] http://www.gdpicture.com/?gclid=CNO5qcKYkK0CFSZ0mAodyHe5mA.

16. DynamicPDF™ Rasterizer for .NET. *Dependable Developer Components.* [Online] ceTe Software. [Cited: December 20, 2011.] http://www.dynamicpdf.com/Dot_NET_Dynamic_PDF_Components_Libraries_Rasterizer.csp.

17. **Fowler, Glenn.** cql – A Flat File Database Query Language. *AT&T Labs Research.* [Online] 1994. [Citat: den 4 September 2011.] http://www2.research.att.com/~gsf/publications/cql-1994.pdf.