HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information

Examensarbete i datavetenskap 10p

C-nivå

Vårterminen 2006

Generering av XSLT-dokument

För transformering av textdatabaser till XML

Daniel Ekedahl

HS-IKI-EA-06-605

Generering av XSLT-dokument

Examensrapport inlämnad av Daniel Ekedahl till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Förnamn Efternamn.

7 juni 2006

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Generering av XSLT-dokument

Daniel Ekedahl

Sammanfattning

Detta examensarbete analyserar huruvida en användare behöver medverka när ett XSLT-skript, med syfte att transformera textfilsdatabaser till XML-dokument, genereras. XSLT-skripten skapas med hjälp av en XSD-fil och en textfilsdatabas samt att användaren mappar dessa värden till XML-dokumentet för att bygga XML-dokumentets struktur. För att utvärdera detta har en prototyp tagits fram som en ”proof-of-concept”. Prototypsystemet visar att en transformering från en textfilsdatabas till ett XML-dokument med hjälp av en XSD-fil inte går att automatgenerera fullt ut. Det medför att en användare måste göra en mappningen mellan textfilsdatabasen till XML-dokumentet. Nyttan av prototypen är att en användare aldrig behöver förstå XSLT-skriptning utan behöver bara mappa värderna i en grafisk miljö som sedan genererar XSLT-skriptet.

Nyckelord: XML, XSD, XSLT, Transformering

Tack!

Jag skulle vilja tacka min familj för deras stöd som alltid funnits. Fredrik Ekengren på Aptic för ett bra förslag till ett roligt examensarbete. Min handledare Fredrik Johansson för bra handledning genom hela projektet. Maria Carlstedt som har varit till stor hjälp för att få ett bra språk genom detta arbete.

Sist men inte minst vill jag tacka alla vänner som har motiverat mig på ett eller annat sätt.

Innehållsförteckning

Figur översikt

Tabell översikt

1 Introduktion

Utbyte av data har blivit allt mer väsentligt i det digitaliserade samhället. Organisationer utbyter data mellan varandra. Ett problem som finns är att organisationerna inte har ett standardiserat format ännu. Små organisationer gör ofta sina egna små program och kan använda t.ex. enkla recordbaserade textfiler. Det finns även äldre programvaror som använder denna typ av textfiler. Organisationerna behöver inte bara kunna utbyta informationen, de måste också kunna transformera data för att deras programvaror ska kunna använda den nya informationen. Flera organisationer går över till Extensible Markup Language (XML)(Hunter 2003) för att transformeringen ska fungera smidigt. Dock är det flera organisationer som fortfarande har textfiler som skulle behöva transformeras över till XML-dokument.

För att transformera data finns det bland annat XSLT-skript(Hunter 2003) som definierar upp vad källfilen använder för format och vilken struktur målfilen ska få. Dessa skript kan bli komplexa och är svåra att skriva. Dessa skript bör skrivas av experter. För att förenkla det för användare bör dessa skript autmoatgenereras. Utan utökning av XSLT så gick inte detta med XSLT 1.0. Detta har dock löst med hjälp av nya funktioner i XSLT 2.0. De har bland annat lagt in sökning med hjälp av reguljära uttryck. Detta gör att det enkelt går att sortera ut data från en textfil och lägga in dess data i ett XML-dokument.

För att verifiera om rätt målfil har skapats finns det XSD-filer som används för att verifiera att rätt struktur på filen har skapats. Med hjälp av denna XSD-filen går det att skapa ett träd för hur strukturen på XML-dokumentet ska genereras. Med hjälp av detta träd kan en användare enkelt mappa data från en textfil till hur XML-filen ska genereras. Sedan så genereras hela skriptet och användaren behöver inte kunna något om XSLT-skriptning.

Detta examensarbete ska undersöka hur ett transformeringsverktyg kan underlätta transformering av textfilsdatabaser till XML-dokument genom att göra ett praktiskt experiment där Extensible Stylesheet Language Transformation (XSLT) används för att transformera data. Den prototypen som kommer att skapas kommer att kunna läsa in en XSD fil för att kunna skapa någon typ av trädstruktur som användaren kan använda sig av när den bygger upp XSLT filen. XSD filen är en väsentlig del när skapandet av XSLT filen ska göras. Detta beror på att XSD filen tillhandaha hur strukturen på det slutgiltiga XML-dokumentet ska vara. Skapandet av XSLT filen kommer att vara helt grafiskt vilket gör att användare inte kommer att behöva kunna något om själva XSLT skriptandet eftersom prototypen genererar detta skriptet.

Det resultatet som denna undersökning visar är att det inte går att göra en transformering av textdatabaser till XML-dokument helt automatgenererat. Detta beror på att eftersom inte textdatabaserna är beskrivande går det inte att matcha olika värden mot olika XSD-filer. Det visar dock tydligt att en användare behövs för att mappa de olika värderna i textdatabasen till de olika värderna i XSD-filen.

Det framgår också att XSD-filer är mer komplexa i sin uppbyggnad en vad DTD-filer är. Detta har fördelar i att XSD-filer kan beskriva mer om vad för typ av attribut som ska lagras och om dessa värden har en viss längd på det slutgiltiga attributet. Det är inget krav på att dessa finns med, det är upp till skaparen av XSD-filen om dessa värden ska finnas med.

Den metoden som har valts är en implementations lösning för att kunna få en proof-of-concept. Den har använt sig av en bottom-up metod under implementationen. Det innebär att prototypen implementerar de byggstenar som körs sist i prototypen och för att kunna simulera data ovanför används drivers som innehåller testdata. När sedan den koden är implementerad kommer de drivers som användes som testdata att implementeras och drivers ovanför dem kommer att finnas för att kunna simulera data.

I kapitel 2 kommer att beskriva de olika delarna inom XML och XSLT. Utöver XML och XSLT kommer beskrivning på hur XSD filer är uppbyggda. Det är dessa tre delar som är väldigt visentliga.

Kapitel 3 tar upp problemet med att transformera textfilsdatabaser till XML och hur denna rapport ska angripa detta problem. Den går även djupare in på en problemprecisering vad detta arbete går ut på.

Kapitel 4 kommer att presenterar den metoden som denna rapport kommer att använda när denna angriper problemet.

Kapitel 5-6 analyserar prototypen och presenterar ett resultat av vad analysen kom fram till. Det presenteras även en slutsats och framtida arbeten i dessa kapitel.

2 Bakgrund

Information som finns i olika organisationer sparas ofta på olika sätt. Det finns inte en standard som gäller. Detta har varit ett problem för organisationer. För att en organisation ska kunna ta ett dokument från en annan organisations system till deras egna system så kan de behöva transformera det till deras egna format. Detta fungerar bra om organisationen bara har kontakter med ett fåtal andra organisationer och de kan skriva specialprogram som transformerar data mellan de olika formaten. Dessa problem har fått organisationerna att se sig om efter någon typ av standard som de i framtiden skulle kunna använda. Den standard som många organisationer börjar använda är Extensible Markup Language (XML). Det är bra att de har börjat inse att någon typ av standard skulle behövas. Problemet kvarstår dock eftersom många organisationer idag antingen inte vill eller inte kan byta sina system som inte använder XML för det skulle bli för dyrt för dem. Problemet som finns är att många av dessa system har sina egna regler för hur data ska sparas. En lösning är att transformera data till XML för att få en gemensam plattform att fortsätta att jobba ifrån. Det är dock kostsamt om varje organisation ska göra sina egna transformationsverktyg.

2.1 XML

Hunter (2003), XML är framtaget för att på ett strukturerat sätt ordna data. Detta görs med hjälp av så kallade markups. Ett universellt språk som kombinerar både enkelheten i vanlig text med att kunna lägga in anvisningar på hur det ska vara strukturerat vore en ideal lösning. Idén att försöka att utbyta information mellan olika system med ett gemensamt format har funnits ända sedan olika system har varit ihopkopplade. Ett första försök att utbyta information mellan olika system med universellt utbytbara format gjordes med hjälp av SGML. SGML är ett textbaserat språk som kan användas för att med hjälp av metadata visa hur något är organiserat, metadata som används är även självbeskrivande. SGML var designat för att beskriva data på ett standardiserat sätt.

Hunter (2003), HyperText Markup Language (HTML) är ett känt språk som bygger på SGML. HTML använder många delar i SGML för att beskriva informationen på ett universellt sätt. Idén var den att alla HTML-dokument som skapas ska kunna ses i en applikation som var kompatibel att förstå HTML, dessa applikationer kallas för webbläsare. Webbläsaren skulle inte bara kunna visa det aktuella dokumentet utan om sidan var länkad till andra dokument så skulle denna kunna visa detta också. Eftersom HTML bygger på att det är textbaserat kunde vem som helst göra ett HTML-dokument. Många skrivprogram t.ex. Word stödjer även att dokumenten kan sparas ner till HTML.

Att göra en jämförelse med HTML är bra eftersom det var en tidig version av ett markup-språk. Det visade även hur kraftfulla dessa språk är. Det finns inte speciellt många taggar i HTML men det var tillräckligt på den tiden då det skapades. I dagens e-handelstid så börjar det bli lite föråldrat som mycket teknik blir efter några år. HTML togs fram för att beskriva hur webbsidor skulle byggas upp. Detta var när ingen visste vad som skulle komma. Men eftersom det visade hur i stort sett vem som helst enkelt kunde använda dessa nästan helt självbeskrivande taggarna för att kunna göra sina egna sidor var det en bra utgångspunkt för att fortsätta att ta fram XML.

Hunter (2003), SGML är så pass komplicerat att det inte passar sig vid dataöverföringar via webben. Dock har HTML blivit en stor succé. Tyvärr är det begränsad i vad det kan göra. De taggar som HTML använder kan inte säga något om vad det är för typ av information utan bara var den ska visas. För att lösa dessa problem så uppkom XML. XML är en delmängd av SGML med samma mål fast när de tog fram XML skulle så mycket som möjligt av komplexiteten i SGML elimineras. XML är designat för att vara kompatibelt med SGML. Det innebär att alla XML-dokument som följer reglerna för XML-syntaxen kommer även att följa SGML-syntax. Detta gör att XML-dokument kommer att kunna läsas av existerande SGML-verktyg. Det betyder dock inte att ett dokument som är skrivet i SGML även kommer vara ett XML-dokument.

Arbetet med att göra XML enkelt att använda har lyckats bra. Det är enkelt att beskriva sina data som ska sparas i XML. Att XML-dokument är kompatibla med SGML är en fördel. Ett utav målen som W3C gruppen satte upp, att XML skulle vara enkelt, har de lyckats med.

Hunter (2003) tar upp att det är viktigt att inse att XML är ett sätt att skapa ett språk som möter XML-kriterierna på ett standardiserat sätt. Ett exempel vore om data ska utbytas mellan system, det skulle exempelvis kunna vara ett namn. Istället för att bara skriva själva namnet i en textfil skulle dessa data kunna organiseras i ett XML dokument. Det skulle kunna se ut som i tabell 1.

```
<namn>
     <förnamn>Kalle</förnamn>
     <efternamn>Olsson</efternamn>
</namn>
```

Tabell 1: Exempel på XML

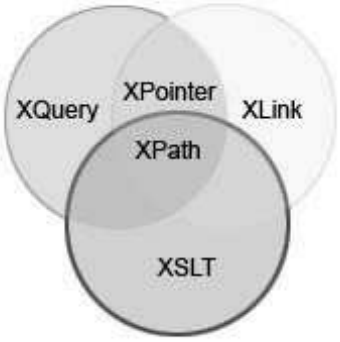
Det är nu ganska enkelt att förstå utifrån tabell 1 varför XML anses vara självbeskrivande. För att lättare förstå hur de olika XML-taggarna är kopplade kan träd skapas. (Hong et al., 2001) använder sig av trädstrukturer för att påvisa hur olika DTD-strukturer kommer att ändra sig när de ska transformeras. Det finns vissa nackdelar med detta sätt att beskriva data. Det gör att storleken på data som ska skickas kommer att ökas, ibland enormt. Målet med XML är inte att få små datafiler utan att få en struktur på data som används och få det så enkelt som möjligt för användarna och utvecklarna. XML är extensible och det menas att taggarna kan skrivas på olika sätt med hjälp av attribut och element, men det är endock samma information som är lagrad fast i olika strukturer. Detta gör att det är lätt för utvecklaren att välja ett sätt som passar denna. Detta kan jämföras med HTML som inte är extensible. Det innebär att det bara går att använda de taggar som redan är definierade och inte lägga till några nya, vilket det skulle kunna göras i XML.

2.2 XSLT

Luis Martín Díaz (2002) beskriver transformering så som att det finns ett behov av att transformera data från ett format till ett annat för att uppnå kompatibilitet mellan informationssystemen. Det är denna definition som används i detta arbete. Denna artikel diskuterar vidare hur vidare XML kan användas för utbyte av datainformation inom organisationen. Den tar även upp problem så som att det kan bli många olika dokument som ska transformeras.

Doug Tidwell (2001), Extensible Stylesheet Language for Transformations (XSLT) är framtaget av W3C för att transformera XML-data till andra format. Dessa andra format som XSLT kan transformera till kan vara PDF, jpeg-bilder eller javakod. XSLT-stylesheet är en beskrivning av regler för hur något ska transformeras. Sedan genomförs det av en XSLT-processor. W3C har definierat en standarder för stylesheet. Det är den äldre CSS som används flitigt när HTML dokument används. Denna äldre standard har dock vissa brister. CSS kan inte ändra ordningen på ett dokument, så ska vissa delar filtreras bort så kommer den inte göra detta. CSS kan inte göra beräkningar, t.ex. skulle man vilja göra en summering på pris kommer inte CSS att klara av detta.

Doug tidwell (2001) fortsätter att diskutera varför XSLT har tagits fram. XSLT är i grunden XML-dokument. Det betyder att ett XSLT-dokument kan beskriva hur en transformering av ett annat XSLT-dokument ska göras för att skapa ett tredje. XSLT bygger på mönsterigenkänning. Det betyder att XSLT letar efter givna mönster och när den hittar dessa mönster så ska den utföra något. XSLT är designat att det ska vara optimerat och ska klara av att köra flera stylesheet simultant. När XSLT körs använder sig XSLT av iterationer och rekursion för att loopa igenom ett dokument. De fördelar XSLT kommer med är t.ex. att data ska distribueras på flera plattformar som webben, mobiltelefoner och handdatorer. Då kan data skapas i ett strukturerat dokument och sedan transformeras över till det formatet som önskas. Det kan även vara att det finns dokument i flera olika format.



Figur 1: Hur begreppen hänger ihop (http://www.w3schools.com/xquery/default.asp)

Figur 1 är en beskrivning hur de olika begreppen sammanfaller. Denna bild visar tydligt hur XSLT är relaterat till de olika delarna i språket. Det som dock kan nämnas är att vissa av dessa standarder är framtagna för att andra saknar viss funktionalitet. Det är som XPath inte räcker till för att hitta alla element så då tog de fram XPointer för att kunna täcka upp det som saknas.

2.2.1 XSLT 1.0 och 2.0

1998 presenterades den första versionen av hur XSLT 1.0 skulle användas. Efter det kom det några ytterliggare uppdateringar och slutet av 1999 kom det slutgiltiga hur XSLT 1.0 skulle användas (Clark, 1999). XSLT 1.0 var designat för att transformera enbart XML dokument till antingen nya XML dokument eller andra format.

XSLT 2.0 fick sin första publika release 2001 och har parallellt med XPath 2.0 uppdaterats och den senaste versionen är daterad november 2005 (Michael Kay, 2005). XSLT 2.0 har utvecklats och blivit ett mer kraftfullt transformeringsspråk jämfört med XSLT 1.0. De delar som är implementerade och som är viktigt för detta arbete är xsl:analyze-string som ger fördelarna att reguljära uttryck kan användas. Detta är viktigt för att kunna separera informationen i en textdatabas. Unparsed text är en ny funktion som också introduceras i XSLT 2.0. Denna funktion läser in en fil som en sträng som sedan kan bearbetas med analyze-string för att dela upp den i användbara informationsdelar.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="2.0">
      <xsl:param name="input-uri" as="xs:string"/>
          <xsl:output indent="yes"/>
          <xsl:template name="main">
          <xsl:variable name="in" select="unparsed-text($input-uri)"/>
      <namn>
      <xsl:analyze-string select="$in" regex="\n">
      <xsl:non-matching-substring>
          <row>
<xsl:analyze-string   select="normalize-space()"       regex="(([A-Z][a-
z]+)([A-Z][a-z]+))\s">
               <xsl:matching-substring>
                    <cell>
                         <xsl:value-of select="regex-group(1)"/>
                    </cell>
               </xsl:matching-substring>
          </xsl:analyze-string>
          </row>
      </xsl:non-matching-substring>
      </xsl:analyze-string>
     </namn>
     </xsl:template>
</xsl:stylesheet>
```

Tabell 2: Exempel i XSLT 2.0 med reguljära uttryck

2.2.2 XPath

XPath är en syntax som beskriver delar i ett XML-dokument. Med XPath kan man referera till det första <namn> elementet eller ett attribut i ett element. Det går även att söka på t.ex. ett namn i ett element. XSLT-stylesheet använder XPath när den gör *match* och *select*. (Doug Tidwell, 2001)

Detta är en viktig del när transformeringar ska göras. Detta beror på att det gäller att veta vilka attribut som ska användas och kunna söka upp dessa.

```
xmlDoc.selectNodes("/computerstore/computer[0]")
```

Tabell 3: Exempel på hur XPath kan användas

Detta exempel visar hur första computer noden väljs från computerstore elementet, XML-data finns i appendix A.

```
xmlDoc.selectNodes("/computerstore/computer[price>600]/name")
```

Tabell 4: Exempel på hur XPath kan användas

Detta exempel väljer ut de computer noder som har ett pris större en 600 och returnerar namnet på dessa.

Trots att XPath är en central del har det gjorts ganska liten forskning kring hur det går att göra det snabbare och mer effektivt. Eftersom det ofta körs på en begränsad dator, CPU och minnesmängd, så är det viktigt att XPath är effektivt. (Georg et al., 2005)

Eftersom XPath används flitigt när transformeringar ska till är det viktigt att XPath är så effektivt som det kan bli. Är det stora dokument som ska transformeras så kan det vara ganska stora tidsskillnader på hur lång tid en körning tar om resurserna utnyttjas effektivt eller inte.

2.2.3 XQuery

Eftersom XML börjar gå mer och mer mot en standard för att utbyta och lagra data så krävs det att det finns ett väl fungerande XML-frågespråk. Detta kräver att frågespråket som är väl anpassat för sökning och navigering genom multipla nivåer av XML-objekt. XQuery är framtaget för detta. Det är ett frågespråk utvecklat av W3C XML Query Working Group och det går mot att bli standarden för att ställa frågor mot XML-data. XQuery ger stöd för att bygga stora klasser av frågor. (Kevin et al., 2005)

När data handhålls gäller det att det går att sortera ut just den/det data som ska användas. Det är detta frågespråket används till. Det går att köra utan ett frågespråk så länge det finns testdata vilket oftast bara är små mängder. Men när det blir implementerat och ska köras gäller det att det går att sortera ut just det data som behövs.

```
for $x in doc("AppendixA")/computerstore/computer
where $x/price>500
order by $x/name
return $x/name
```

Tabell 5: Exempel på hur XQuery kan använas

Tabell 5 är ett exempel på hur computerelementet gör en selection där price är större än 500, ordnar dem efter namn och sedan returnerar bara namnet.

2.2.4 XPointer

XPointer är baserat på en väl använd teknologi vilket är Text Encoding Initiative ”extended pointer”. Extended pointer bidrar med axlar för att kunna navigera i träd. Extended pointer har blivit implementerat i flera SGML-baserade navigeringssystem. För att lösa vissa problem med XPath som finns som gör att det blir lite begränsat i vissa aspekter så togs XPointer fram. Vissa delar från XPointer har införts i XPath men det finns fortfarande vissa delar som inte finns med i XPath som finns i XPointer som är bra. (Steven, 1999)

Det har varit så länge i databranschen, att när vissa delar saknas så istället för att börja om från början så läggs det till nya paket och så blir det ihopplock av paket som lappar över varandra. I detta fall så är det en standard som behöver byggas på och det

löser problemen som har uppkommit och det är ju det som är syftet med dessa utökningar.

```
<mydog xlink:type="simple"
xlink:href="appendixA#xpointer(category('Server'))">
```

Tabell 6: Exempel på hur XPointer används

Detta exempel refererar till ett element i ett specifikt dokument.

XPointer har dock vissa egenskaper som gör att det går att skriva om detta uttryck för att få det mindre.

```
<mydog xlink:type="simple"  xlink:href="appendixA#Server">
```

Tabell 7: Enklare omskrivning från tabell 6

Dessa två exempel hänvisar till samma element i ett visst dokument.

2.2.5 XLink

XLink är till för att skapa länkar inom och mellan XML-dokument. I XML kan alla element göras till ett länkande element. Det är väsentligt eftersom XML inte har några fördefinierade element. XLink-syntaxen och utvecklingsmöjligheter är inspirerade från succén av HTML. (Erik et al., 2001)

Dessa punkter är inspirerad av Erik & Christopher (2001)

- Vilket XML-element som helst kan skapas till en länk
- XLink kan använda XPointer för att ta sig till vilket ställe som helst i dokumentet.
- XML kan använda XLink för att importera text.
- XPointers kan definiera ett intervall av XML-markup för att referera till en delmängd av ett dokument.

```
<?xml version="1.0"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
<homepage    xlink:type="simple"    xlink:href="http://www.his.se">Visit
University of Skövde</homepage>
</homepages>
```

Tabell 8: Exempel på hur XLink används

xmlns:xlink är till för att få tillgång till xlink-attributen. Homepage är exempel på hur en link fungerar.

2.3 XML-validering

Informationsutbyte har ökat markant och detta betyder att mycket av de data som skickas måste kunna valideras så att det är korrekt data som skickas. XML har två olika standarder för validering som de använder. Det är den äldre Document Type Definition (DTD) som är begränsad när det gäller validering. För att lösa dessa

begränsningar så finns XML Schema Definition (XSD), XSD är helt uppbyggt av XML vilket gör att den inte har samma begränsningar som DTD har. Det gör att XSD kan få alla fördelar som XML ger. (Giovanna et al., 2005)

Genom att använda XSD så kommer valideringsdokumenten att vara i XML format vilket inte var fallet innan när DTD användes. Detta är en stor fördel om nya XML-dokument ska skapas där vissa element inte finns med. Då går det enkelt att transformera dessa dokument också vilket hade varit annorlunda med DTD eftersom DTD har sin egen struktur som gör att XML-transformeringar med XSLT inte kan genomföras. I 2.3.1 visas ett exempel på hur ett DTD-dokument kan se ut och om samma sak ska åstadkommas i XSD visas i 2.3.2. Det som går att få ut om en jämförelse görs är att XSD har fler attribut som den kan validera mot. Det går bl.a. att få ut om det är ett korrekt bestallnings-ordernummer. Det går även att få ut om datumet är inom kalendertiden. Dessa saker saknas i DTD vilket gör att det är ganska begränsat till skillnad från XSD vad den kan validera mot.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT anvandare (anvandar-id , bestallnings-order+ , order-datum)>
<!ELEMENT anvandar-id (#PCDATA)>
<!ELEMENT bestallnings-order EMPTY>
<!ATTLIST bestallnings-order del-nummer CDATA #REQUIRED
     antal CDATA #REQUIRED >
<!ELEMENT order-datum EMPTY>
<!ATTLIST order-datum dag CDATA #REQUIRED
     manad CDATA #REQUIRED
     ar CDATA #REQUIRED >
```

Tabell 9: DTD-exempel

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="anvandare">
      <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element ref="anvandar-id"/>
                  <xsd:element                    ref="bestallnings-order"
maxOccurs="unbounded"/>
                  <xsd:element ref="order-datum"/>
                  </xsd:sequence>
           </xsd:complexType>
      </xsd:element>
<xsd:element name="anvandar-id" type="xsd:string"/>

<xsd:element name="bestallnings-order">
           <xsd:complexType>
                  <xsd:attribute name="del-nummer" use="required">
                        <xsd:simpleType>
                              <xsd:restriction base="xsd:string">
                                 <xsd:pattern   value="[0-9]-[0-9]{4}-[0-
9]"/>
                              </xsd:restriction>
                        </xsd:simpleType>
                   </xsd:attribute>
          <xsd:attribute          name="quantity"          use="required"
type="xsd:integer"/>
         </xsd:complexType>
     </xsd:element>
```

```
<xsd:element name="order-datum">
         <xsd:complexType>
            <xsd:attribute name="dag" use="required">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:integer">
                     <xsd:maxInclusive value="31"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="manad" use="required">
               <xsd:simpleType>
                   <xsd:restriction base="xsd:integer">
                       <xsd:maxInclusive value="12"/>
                   </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
          <xsd:attribute name="ar" use="required">
            <xsd:simpleType>
               <xsd:restriction base="xsd:integer">
                  <xsd:maxInclusive value="2100"/>
               </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Tabell 10: Exempel på XSD-scheman

2.4 Typer av XSD-filer

De typer av XSD-filer som prototypsystemet klarar av är enkla strukturer. Dels klarar den av de exempel som finns i appendix C. Detta gör att den blir begränsad. Det beror på att det går att göra länkar i XSD-dokumenten som gör att vissa värden hämtas från andra strukturer och sätter in dessa i den första strukturen. Det är dock något som går att bygga ut så att den klarar detta.

```
<xs:element name="anstald" type="personinformation"/>
<xs:element name="student" type="personinformation"/>

<xs:complexType name="personinformation">
  <xs:sequence>
    <xs:element name="firstanamn" type="xs:string"/>
    <xs:element name="efternamn" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Tabell 11: Exempel på interna länkar

I tabell 11 beskrivs hur ett personinformations-objekt kan användas av flera olika element. Detta gör att varje element måste undersökas om det finns referenser till andra objekt och om det finns måste dessa hanteras på något vis.

```
<xs:element name="employee" type="fullpersoninformation"/>

<xs:complexType name="personinformation">
  <xs:sequence>
    <xs:element name="firstanamn" type="xs:string"/>
```

```
    <xs:element name="efternamn" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninformation">
  <xs:complexContent>
    <xs:extension base="personinformation">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="stad" type="xs:string"/>
        <xs:element name="land" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Tabell 12: Exempel på en bas referens

2.5 Relaterade arbeten

Det finns endel olika program och artiklar som berör det område som detta examensarbete behandlar. Detta stycke tar upp dessa och diskuterar vad detta examensarbete tillhandahåller som inte redan finns i dessa produkter.

2.5.1 Sylusstudio

Stylusstudio (DataDirect Technologies, 2006) är ett komersiellt program som tillhandahåller transformeringar mellan olika format men framförallt inom XML. Det går att bygga upp XSLT-skript inne i produkten men då kräver det att användaren av produkten kan notationen för XSLT. Det finns dock visst stöd när användaren ska skapa dessa skript med hjälp av dropdown listor där användaren kan välja de nödvändiga attributen. Det finns även stöd i programmet för att transformera textdatabas-filer men det är bara för enskilda rader som är antingen recordbaserade eller teckenseparerade, med tillexempel komma. Det går att bygga upp strukturer för hur XML-dokumenten ska vara strukturerad. Eftersom den enbart klarar av en rad är den något begränsad. Det går också bara att få ut XML-filen, så själva XSLT-skriptet för hur transformeringen ska gå till berör de inte alls. Eftersom det är ett komersiellt program så är det svårt att veta om de använder XSLT över huvud taget när de gör sina transformeringar från textfilsdatabaser.

2.5.2 XFlat

XFlat (Bob Lyons, 2006) är ett XML-schema för att definiera upp textfilsdatabaser. Dels är den begränas vad som gäller funktionallitet. Den strukturen som textfilsdatabasen använder sig av kommer slut XML-dokumentet att få. Detta gör att det inte går att ange om det är ett attribut till ett visst element. Den kommer alltid att lägga alla data mellan två elements taggar. Detta gör att om det slutgiltiga XML-dokumentet använder sig av attribut i elementen så måste det till en transformering till. Detta görs smidigast med XSLT. Detta gör att dels så måste en användare först lära sig XFlat notationen för att kunna transformera det till XML för att sedan använda sig av XSLT för att kunna transformera det till det slutgiltiga XML-dokumentet. Detta gör att det är överflödigt eftersom detta går att åstadkomma med XSLT 2.0. Då behöver inte användaren lära sig två olika notationer för att transformera en textdatabasfil till ett XML-dokument. Detta kräver även att använder kan skripta eftersom det inte finns någon grafisk miljö som användaren kan använda

sig av. Det gör att detta skript språket lämpar sig framförallt åt utvecklare som redan är vana vid XML och skriva skript.

2.5.3 Relaterad forskning

(Paula, 2003) diskuterar i sin artikel två problem. Dels om hur olika DTD-scheman kan användas för att automatiskt matcha DTD-scheman mellan varandra. Det han gör är att han tar ett käll DTD-schema och ett mål DTD-schema för att kunna jämföra skillnader mellan dessa. När det är gjort görs en transformering från käll XML-dokumentet och transformeras över till mål XML-dokumentet. Det görs även statistiska på hur bra kvalite det slutgiltiga XML-dokumentet får. Det andra som tas upp som en del i artikeln är att det även har tagit fram en algoritm för själva transformeringen.

3 Problemområde

”Manual translation of the XML documents is time consuming and thus especially unacceptable for web services, where the information sources change frequently” (Hong, Harumi, & Elke, 2001, pp 1). I denna teknikålder ska allting gå snabbare att utföra. Detta gör att automatisering och förenkling bör utvecklas. Detta gäller inte bara XML utan även transformering av alla typer som lagrar data.

Det har gjorts fortsatta arbeten att automatisera det än mer. Detta är dock bara mellan XML-dokument och de använder den äldre DTD-standarden. (Paula, 2003) För att förenkla arbetet har forskare kollat på markup-processer. Redan 1981 kom det ett förslag på hur en generalisering av markup-dokument skulle fungera. Istället för att använda taggar som de gör idag i XML så var förslaget som t.ex. :p. som i detta fall representerade en paragraf. (C.F.Goldfarb, 1981) Han fortsätter och diskuterar att detta är bra eftersom de inte blir beroende av en applikation, formateringsstilar eller processningsystem.

Boukottaya (2004) beskriver problemet som finns när en transformering med XML ska göras. De problem som Boukottaya tar upp är att bördan faller på en människa som först måste analysera inte bara semantiken utan även strukturen på källfilen och på slutfilen. När detta är gjort så måste denna person även manuellt skripta XSLT filen som ska användas. Denna artikel försöker att hitta en lösning på hur de kan automatisera transformeringen. De använder XML-scheman och ska automatiskt ta fram XSLT-scheman. Dessa prototyper använder sig av XSLT 1.0. Detta handlar också bara om mappning mellan XML-dokument. Det den också påvisar är att en människa måste vara med och i värsta fall sitta själv och analysera de olika schemana för att få det rätt. Det är som Hong, Harumi, & Elke (2001) tar upp att automatisering är ett måste i framtiden och en lösning på automatisering av transformering är en bit på vägen för effektivisering.

Dessa arbeten handlar om att transformera redan skapade XML-dokument. Att transformera en textfil med XSLT 1.0 är svårt eftersom denna inte har stöd för reguljära uttryck. Det har de dock löst i XSLT 2.0 med hjälp av `xsl:analyze-string` där det går att ställa reguljära uttryck och på det sättet kunna analysera en textfil och kunna transformera den till XML. (Michael Kay, 2005). Eftersom XSLT 2.0 först kom ut officiellt november 2005 så har det inte gjorts mycket forskning kring denna delen och de nya funktioner som har lagts med. Det är därför intressant att undersöka hur dessa funktioner fungerar.

3.1 Problemprecisering

Målet med detta arbete är att finna ett generiskt sätt att utifrån en textfilsdatabas och en XSD-fil kunna generera en XSLT-fil för att transformera textfilsdatabasen till en giltig XML-fil som ska få samma struktur som XSD-filen kräver. För att åskådliggöra detta ska en prototyp skapas. Prototypsystemet kommer visa hur mycket som går att generera och hur mycket påverkan från en människa som kommer att behövas för att få det hela att fungera.

3.2 Avgränsningar

Det finns vissa problem som inte är implementerade för i XSLT men som det har gjorts forskning på. Det är bl.a. ” Firstly, some templates within an XSLT stylesheet may never be applied during execution, regardless of the XML source being input. We call such templates *unreachable templates*.” (Ce et al., 2004). Detta är något som inte kommer att tas i beaktning när prototypen görs. ”Thirdly, the XSLT program itself may loop forever on some XML input(s). This is the problem of XSLT termination.” (Ce & James, 2004) Detta kan bli ett problem om XSLT-filerna genereras på ett felaktigt vis. Det kommer dock vara accepterat om det händer. Anledningen till att andra punkten inte tas med är för att den handlar om felaktiga DTD-valideringar vilket inte kommer att användas. Dessa problem har dock undersökts och hittats lösningar på i den refererade artikeln.

3.3 Förväntat resultat

Det resultat som förväntas är att det inte går att helt automatgenerera denna typ av process. Detta beror på att eftersom textdatabaserna inte innehåller någon typ av extra information om vad varje attribut betyder så går det inte att automatiskt generera hela vägen utan att det behövs någon typ av mänsklig påverkan som mappar de olika attributen mellan var i textdatabasen attribut finns, till var i XML dokumentet det ska transformeras.

4 Metod

I metoddelen pressenteras olika metoder som kan användas för att samla in information till ett examensarbete. Det presenteras vilken metod detta arbete använder samt fördelar och nackdelar.

4.1 Metodpresentation

Berndtsson (M.Berndtsson et al., 2004) presenterar några olika metoder som kan användas när ett forskningsproblem ska lösas. Den metoden som lämpar sig för detta forskningsproblem är en implementationslösning.

Detta arbete kommer att använda sig av en implementationslösning. Detta beror på att arbetet kommer att fokuseras på hur enkelt det går att generera fram script för att transformera textdatabaser till XML-dokument. Berndtsson (M.Berndtsson, J.Hansson, B.Olsson, & B.Lundell, 2004) tar upp valideringen som ett problem när program blir implementerade. Den som implementerar måste veta att den algoritmen som implementeras verkligen stämmer för att en korrekt validering av den implementerade del ska kunna valideras.

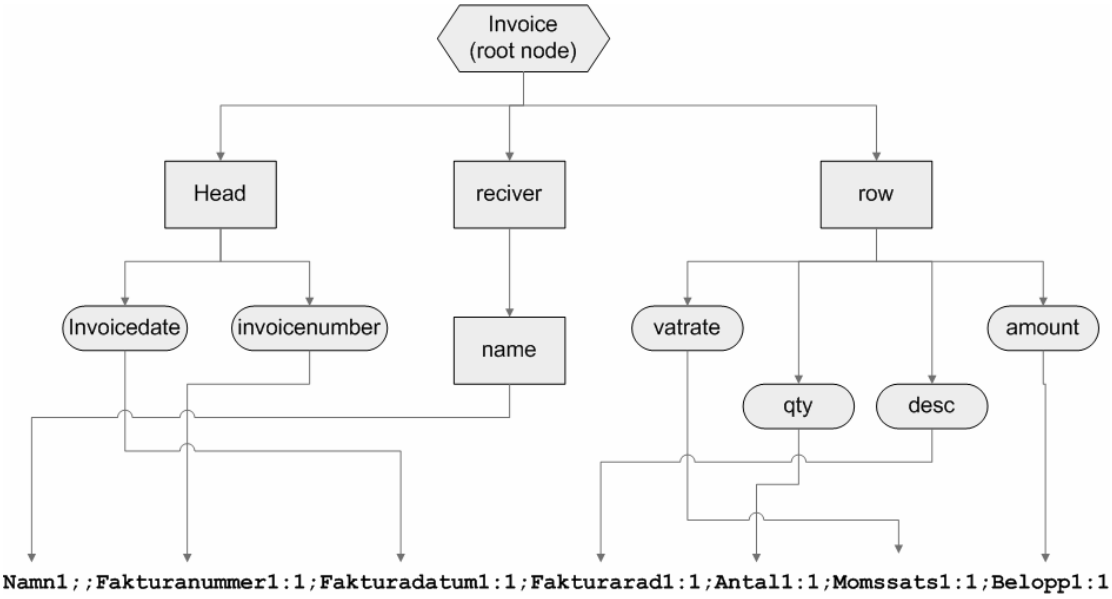
Detta arbete kommer att fokusera på en implementation för att få en ”proof of concept” på hur XSLT 2.0 stödjer transformering av text databaser till XML-dokument. Om dessa transformeringar skulle göras i XSLT 1.0 så var programmeraren tvungen att göra en egen utökning av XSLT 1.0. Det prototypsystemet ska visa är om det går att göra dessa transformeringar utan att tillägg till XSLT-biblioteken behöver göras.

Att göra en ”proof of concept” istället för att använda en litteraturstudie är en fördel om det inte har forskats inom området. Det vill säga, om det finns lite eller ingen existerande forskning inom området kan det bli svårt att komma fram till något verkligt svar mer än att det saknas forskning inom området. Det kan överbryggas om en ”proof of concept” används eftersom det skapas något nytt som bygger på delar av området där det saknas information.

4.2 Metodval och metod genomförande

Eftersom detta arbete handlar om att ta fram en prototyp över hur XSLT 2.0 stödjer transformering av text databaser till XML-dokument kommer en implementations-lösning att genomföras. Implementationen kommer först att ta fram designdokument för hur implementationen ska ske. Dessa dokument kommer att bestå av klassdiagram som visar hur implementationen är sammankopplad. När dessa dokument är framtagna kommer en implementation att göras utefter designdokumenten.

Denna implementation kommer att använda sig av en bottom up metod. Det innebär att drivers för överliggande delar i koden görs och så implementeras de underliggande delarna. Detta gör att implementationen görs bakifrån. Det gör även att en inkrimentiell implementation används. Det innebär att koden hela tiden testas av och när den är testad och allting fungerar så görs de överliggande drivers till korrekt kod och dess överliggande kod görs till drivers. (Christian W.Dawson, 2005) Dessa drivers består av statisk data vilket kommer efter att implemetationen är gjort att bestå av dynamisk data. Efter att implementationen är gjord kommer implementationen att utvärderas och ett resultat och en slutsats kommer att presenteras.



Figur 2: Mappning mellan XSD-träd till en sekvenssträng

Utvärderingen kommer att ske med hjälp av given testdata som går att återfinna i appendix C-D. För att enklare förklara testdata som används så har det gjorts en trädstruktur som går att återfinnas i figur 2 som beskriver hur ett XSD-trädet mappas mot en teckenseparerad sträng. Eftersom de data som ska transformeras består av sekvenser som är en utav de simplaste datastrukturer, så finns det inte många olika strukturer. De två som skiljer sig mest är record-baserade där varje fält har en fixerad längd och den andra är att ett visst tecken skiljer de olika värdena. Utöver dessa kan de andra strukturerna ses som en lång sekvens med någon typ av skiljetecken mellan varje värde. Även om det är en ny rad så kan det tecknet som symboliserar en ny rad ses som ett skiljetecken åt. Då blir det bara att det finns två olika tecken som symboliserar samma sak.

5 Resultat och diskussion

I detta kapitel kommer resultat från implementationen av prototypen att presenteras. Dels kommer prototypen i helhet att diskuteras och vilket resultat som prototypen genererar. Det kommer även att finnas ett diskussionsavsnitt som tar upp hur prototypen är relevant. I denna del kommer även diskussioner om hur prototypen kan förbättras.

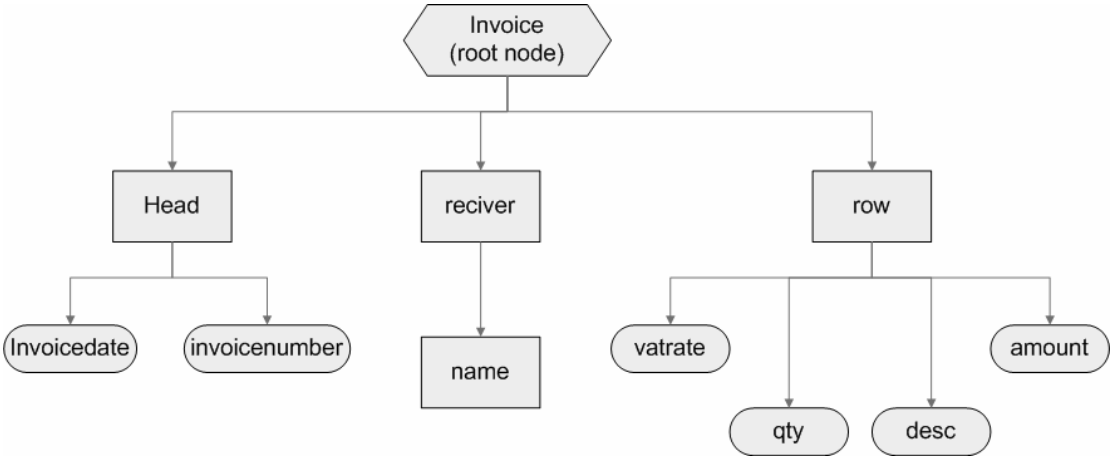
5.1 XSLT

XSLT har fått nya funktioner i och med version 2.0. Detta gör att nya möjligheter finns som var svåra eller helt omöjliga innan i version 1.0.

De delar som XSLT 2.0 har utökats med har givit detta språk stor potensial att kunna hantera fler datastrukturer på ett enklare sätt som var svårt i version 1.0. Detta beror på att dessa regulära uttryck som nu går att använda gör att det går att utvinna data från textdatabaser som inte är uppbyggda av XML-taggar. Det är framförallt denna funktion som prototypsystemet som har skapats använder sig av. Detta beror på att det var just textdatabaser som skulle transformeras och då passar just regulära uttryck bra för att kunna matcha strängar i dessa filer. Prototypsystemet visar dels att det är möjligt att transformering mellan textdatabaser till XML-dokument med XSLT 2.0 via de nya tilläggen som är gjorda. Detta gör att istället för att göra utökningen från XSLT 1.0 med dessa nya funktioner så kan XSLT 2.0 användas vilket gör att flera använder och testar av de nya funktionerna om något fel skulle uppstå i de nyimplementerade funktionerna. Det gör även att utvecklaren kan utnyttja alla de nya funktionerna och inte bara en viss specifik funktion som denna behövde just då.

5.2 Analys av prototypen

Prototypsystemet utvärderas med hjälp av data som går att hitta i appendix C och D. Appendix C tar upp XSD filen som använd för att generera trädstrukturen och D visar en sträng som är record-baserad och en struktur som är separerad med semikolon. När en rad är record baserad så innebär det att varje värde har en fixerad längd. Detta gör att det är enkelt att sortera ut vad för värde som ligger var i filen. När en textfil är separerad av/med något speciellt tecken, i detta fall av/med semikolon, så söks strängen igenom efter semikolon och mellan dessa delar så finns det ett värde. Dessa värden ligger alltid på samma plats. Eftersom dessa strängar kan anses vara sekvenser, då värdena i strängarna alltid kommer i samma ordning så handlar det att hitta sekvenserna och översätta dessa till trädstrukturer. För att göra denna koppling så används ett XSD-dokument som beskriver vilka värden som finns och hur dessa är ordnade. Dessa XSD-dokument som bygger på XML kan tolkas som trädstrukturer. Den XSD-fil som användes när prototypen testades finns i appendix C. Den struktur som användes när prototypen testades finns beskriven som en trädstruktur i figur 3.

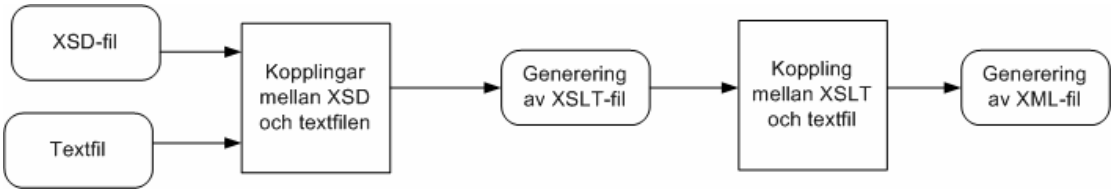


Figur 3: XSD-filen beskrivet som en trädstruktur

Figur 3 har tre entiteter. Den har först en root nod som varje rekursion börjar och slutar med. I denna modell är det invoice. Sedan finns det fyrkanter som representerar ett element. Om den sista noden i trädet är ett element så kommer texten i denna att omslutas först med en tag, sedan kommer texten och sedan kommer elementets sluttag. Detta är en skillnad från attributen som är i runda boxar. Dessa attribut kommer att läggas på den överliggande nodens element som attribut. Detta medför vissa begränsningar. Det gör bland annat att text i element och att använda attribut på samma gång som inte går i prototypsystemet. Det finns även fall då det ska gå att ha flera rekursioner inom en del. Det ska till exempel gå att lägga till flera värden av row till samma invoice. Detta är dock inte implementerat i prototypsystemet.

5.3 Flödet i prototypen

Det flöde som sker i prototypsystemet illustreras i figur 4. Ett mer detaljerat flödesschema kan återfinnas i appendix H.



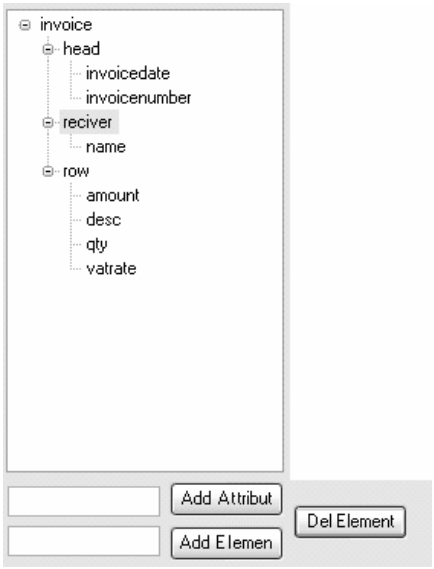
Figur 4: Flödet i prototypen

Det som sker är att först väljs två filer ut. Det är dels XSD-filen som används för att skapa trädstrukturen. Sedan så väljs textfilen som ska transformeras. Första vyn på valet av de två filerna kan återfinnas i appendix B. När dessa filer är valda ska andra vyn användas. Denna vy är central för hela prototypen. Det är här alla kopplingar mellan de olika filerna görs. Denna vy är ganska rörig eftersom det finns många valmöjligheter. Detta skulle kunna göras bättre genom att en flerstegs-wizard läggs in. Efter att kopplingarna mellan filerna har lagts till är det bara att generera sitt XSLT-skript. När detta skript är genererat går det att se hur det fungerar och hur skriptet ser ut. Detta görs i tredje vyn. Om det är något som inte stämmer eller om det är en expert som vill förändra något i XSLT-skriptet så går det. Om användaren har valt vilken textfil som ska användas är det bara att generera XML-filen, i annat fall måste

användaren välja vilken textfil som ska transformeras. Detta går att se i vy tre. I fjärde vyn återfinns den färdiga genererade XML-koden. Det är detta som ska vara det slutgiltiga resultatet om XSLT-skriptet har lyckats. För att verkligen se om XML-filen använder rätt format kan XSD-filen användas för att verifiera att XML-filen är korrekt.

5.4 Användning av prototypen

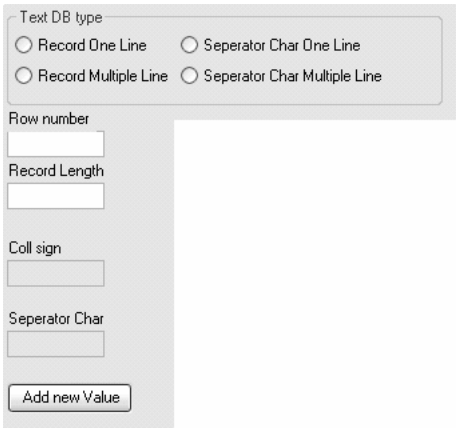
Prototypsystemet är först och främst beroende av andra vyn. Det är denna del som är den centrala delen när prototypen körs. Det är här som användaren kopplar samman strukturen från XSD-filen med hur textfilen är uppbyggd. Figur 5 visar hur ett träd från en XSD-fil kan illustreras.



Figur 5: Trädstruktur från XSD-fil

En full illustration av hela fönstret går att återfinna i Appendix G. I denna vy går det att enkelt editera hur strukturen ska se ut om det har blivit fel när XSD-filen skulle laddas in eller att det är fel i XSD-filen. Det går även att skapa trädstrukturer om det inte skulle finnas en XSD-fil. Ett problem med denna vy är dock att det inte går att se om den sista noden i trädet är ett element eller ett attribut vilket kan vara ett problem. Detta är avgörande när trädet ska skapas och kan vara till hjälp när mappningen ska göras.

Denna andra vy som illustreras i figur 6 beskriv hur nya värden lägts till.



Figur 6: Vy över vad användaren kan fylla i

Användaren väljer dels vilken typ av textfil som ska användas, om det är en record-baserad eller om det är en textfil som är separerad med hjälp av något speciellt tecken. Beroende på vilken typ av fil som användaren väljer att den ska använda kommer de fyra huvudrutorna att aktiveras och deaktiveras beroende på val. De fyra rutorna är ordnade som så att första rutan uppifrån säger i vilken ordning ett värde kommer i textfilen. Andra textrutan beskriver hur lång den raden är som record. Alltså hur många tecken som används mellan varje värde. Tredje textrutan används om det är multipla rader och ett signifikant tecken används i början av raden. Fjärde och sista textrutan används när separation med ett tecken ska användas. Då fylls det i vilket tecken som används för att separera de olika värdena.

5.5 Typer av textfiler

Eftersom textfilerna kan ses som sekvenser så är det enkelt att klassificera in dessa filer i några få generiska typer. Dels är det de filerna som har värdena med en specifik längd, dessa brukar kallas för record-baserade. Dessa filer är enkla att definiera upp eftersom längden på varje atribut aldrig ändrar sig. Ett exempel på hur en sådan rad kan se ut finns i tabell 13.

```
Namn1   Fakturanummer1   Fakturadatum1 Fakturarad1     Antal1  Momssats1
Belopp1
```

Tabell 13: Exempel på en record-baserad post

Sedan finns det de filerna som har en typ av tecken som avskiljer raderna. Detta kan vara, som finns i testdatan, semikolon. Det innebär att varje rad delas upp av segment med detta tecken som separerar segmenten. Det gör att det bara är att söka på dessa tecken och när dessa hittas så vet den att mellan start på raden till första tecken finns det ett värde. Mellan två separeringstecken är det ett värde och mellan sista separeringstecken och slut på raden finns det sista tecknet. Exempel på hur en sådan rad kan se ut finns i tabell 14.

```
Namn1;;Fakturanummer1:1;Fakturadatum1:1;Fakturarad1:1;Antal1:1;
Momssats1:1;Belopp1:1
```

Tabell 14: Exempel på textsträng med ; som skiljetecken

Det är dessa två typer av filer som prototypsystemet hanterar. Detta gör den begränsad.

Det finns dock lite mer komplexa strukturer av textfiler. Det finns de som använder sig av flera rader och lägger olika värden på varje rad. Det gör att det blir enkelt att lägga till rekursiva delar genom att bara lägga till en viss typ av rad flera gånger. Detta är illustrerat i tabell 15.

```
00  - Start post - 2006-02-15
01  Namn1:1   Fakturanummer1   Fakturadatum1  Momssats1
02 Fakturarad1:1     Antal1:1  Belopp1:1
02 Fakturarad1:2     Antal1:2  Belopp1:2
02 Fakturarad1:3     Antal1:3  Belopp1:3
01  Namn2:1   Fakturanummer2   Fakturadatum2 Momssats2
02 Fakturarad2:1     Antal12:1  Belopp2:1
02 Fakturarad2:2     Antal12:2  Belopp2:2
99 - Slut post –
```

Tabell 15: Exempel på rekursiv data med record-baserade atribut

5.6 Mänsklig påverkan

De problem som upptäcks när XSLT-dokument genereras från textdatabaser är att de fullt ut inte kan automatgenereras. Det kan dock byggas grafiska gränssnitt som gör att skripten inte behöver handknackas i en editor. Detta gör att inte så erfarna använda kan använda XSLT som underliggande språk för att transformera textdatabaser till XML-dokument. Det som dock alltid måste göras är att peka ut vilka element i textdatabasen som tillhör vilket XML-attribut. Detta måste alltid göras eftersom textdatabas filen inte är beskrivande. Det går dock att göra statistiska förslag på vilket attribut som är mest troligt för ett visst element i textdatabasen. Dessa statistiska förslag kan innebära att vissa element eller attribut pekas ut som mer troliga en andra. Det beror lite på hur väl XSD-filen är beskriven. Om dessa filer är bra beskrivna så går det dels att få ut vilken typ av värden som ska finnas i dessa element och dels hur stora dessa element är. Med hjälp av dessa data går det att hjälpa användaren att ge förslag på mer eller mindre troliga fält som ska användas. Detta kan vara bra om det finns många fält att mappa. Men det är bara mappningen mellan de olika elementen som är nödvändig för att genereringen av XSLT-skriptet ska fungera.

5.7 Utvärdering av prototypen

Utvärderingen av prototypen gick till som så att XSD-dokument från appendix C användes och valdes i första vyn, appendix B. Dess data som återfinns i appendix C kommer från Aptic. Sedan så valdes en textdatabas som skulle utvärderas. De som valdes kan återfinnas i appendix D. När dessa sedan är valda så genereras en trädstruktur från XSD-filen. Dessa strukturer som prototypsystemet klarar av är enkla. När dessa strukturer är genererade så ska en användare mappa fälten i textdatabasen mot trädstrukturen.

Det är denna del som inte går att få automatgenererad. Det måste finnas någon typ av mänsklig påverkan som vet hur strukturerna på textdatabasen och vad det slutgiltiga XML-dokumentet ska få för struktur. Det är här den andra vyn används. Hur denna vy används förklaras mer i avsnitt 5.4. För att få den mer användarvänlig skulle denna vy behöva göras om. Det skulle kunna handla om att göra om denna till en mer wizzard-liknande struktur där användaren klickar sig framåt och markerar vad den vill lägga till. Eller om en mer logisk struktur på gränssnittet används.

Det finns vissa brister i prototypsystemet när record-baserade delar används. Detta beror på att när dessa delar ska specificeras var de finns i textdatabasen och i mappningen så måste de göras i ordning.

Figur 7: Hur record baserade strängar kan uttryckas

Om figur 7 används som exempel så syns det ganska fort att varje del representeras av en viss längd av tecken. För att prototypen ska fungera helt så måste alla dessa avstånd finnas med i rätt ordning. Detta beror på att när det reguljära uttrycket körs så kommer den att dela upp varje rad i olika segment utefter hur långt varje element är.

```
<xsl:analyze-string select="."
regex="(.{{8}})(.{{14}})(.{{17}})(.{{16}})(.{{8}})(.{{11}})(.{{8}})">
```

Tabell 16: Exempel på reguljärt uttryck med recordbaserade databaser

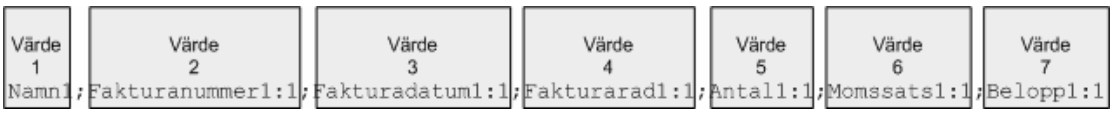
Så ett exempel på hur det reguljära uttrycket kan uttryckas visas i tabell 16. Det detta reguljära uttryck gör är att den tar och analyserar en viss rad. Plockar ut de första 8 tecknen och lägger dessa i en referens som går att anropas. Sedan tar den de 14 nästa tecknen och lägger i en ny referens. På detta viset jobbar den sig igenom hela strängen och delar upp den i olika fragment.

```
<name><xsl:value-of select="regex-group(1)"/></name>
```

Tabell 17: Exempel på att hämta ett visst record element

För att få tag i ett visst fragment del så går det att göra ett anrop som i tabell 17. Det den gör är att den lägger in det som finns i regex-group(1) mellan namn taggarna. Hur hela det skriptet ser ut går att återfinna i appendix E.

När en teckenseparerad textdatabas används finns det lite mer möjligheter att inte behöva specificera upp lika mycket var varje värde finns. Eftersom den vet vilket tecken som den ska separera med behöver användaren bara veta vilken position den har i textdatabasen och med vilket element i XSD-trädet som den ska länkas emot. Det tecken som används som separerings-tecken kommer att exkluderas.



Figur 8: Exempel på uppdelning i teckenseparerad textdatabas

Figur 8 visar tydligt hur de olika värdena delas upp. Detta gör det betydligt lättare för användaren eftersom den enbart behöver välja vart i textdatabasen det hänvisade värdet finns.

Till skillnad från den record-baserade lösningen där varje segment var tvungen att först delas upp och sedan anropas som tabell 17 visar så gör den teckenbaserade lösning detta i på samma ställe. Det gör dock att den inte är speciellt optimerad men det gör också att den inte behöver specificera upp alla värdena i tabellen.

```
<name>
<xsl:for-each select="tokenize(., ';')">
    <xsl:if test="position() = 1">
        <xsl:value-of select="current()"/>
    </xsl:if>
</xsl:for-each>
</name>
```

Tabell 18: Exempel på hur teckenseparerad textdatabas väljer värde

Det som görs i exemplet som finns i tabell 18 är att för varje ”;” så adderas position() med ett. Detta gör att mellan två stycken ”;” så befinner den sig på en specifik position. Om en viss position är som i detta exempel 1 så kommer dess värde att läggas till. Detta gör att det kommer att bli ett litet fel i slutet av varje sträng när det kommer en ny radbrytning. När det kommer till det sista värdet i strängen och det finns en radbrytning direkt efter denna kommer ”**
” att skrivas ut **.

När användaren har definierat var de olika värdena är och mot vilket element eller attribut det ska transformeras till så är det generering av XSLT-skriptet som görs. Dessa går att hitta i appendix E. I prototypsystemet så finns det en liten bugg som gör att \n inte går att användas. Detta beror på att den tolkar detta som ett nytt radkommando och ersätter detta tecken med det. Detta gör att användaren måste själv ändra detta manuellt.

```
<xsl:analyze-string select="$in" regex="\n">
```

Tabell 19: Vart manuell ändring måste göras i XSLT-skriptet

Om det är en expert så kan denna gå igenom skriptet. Detta är dock mer till för en debug-fönster alternativt för expertanvändare om de behöver kontrollera något. En vanlig användare ska aldrig behöva se det utan den ska bara kunna gå direkt till det färdiga XML-dokumentet. Det flödet som har visats kan återfinnas i appendix H.

5.8 Utökningar av prototypen

prototypsystemet har i nuläget stöd för de två enklaste sekvenstyperna när en sekvens finns på en enskild rad. Detta gör den något begränsad. Den kod som är gjord är dock implementerad så att det finns delar att utöka för att få den att fungera med multipla rader och rekursioner i textfilerna. Prototypen är också begränsad i hur avancerade XSD-filer som går att läsas in. Länkar mellan olika strukturer klarar den inte av. Den klarar dock av att läsa in raka XSD-filer. Den kollar bara i dagsläget på om det är ett attribut eller ett element och var i trädet dessa delar ligger. Det går även här att utöka denna sökning så att den klarar av mer komplexa strukturer.

6 Slutsats

De slutsatser som kan dras från detta arbete är att transformering mellan textfiler till XML inte är helt enkelt. Eftersom textfilerna inte har några beskrivande delar vad varje värde är blir det svårt att automatiskt matcha olika tabeller. Det går att matcha om det bara är siffror i en viss tabell eller om ett visst värde alltid har samma längd. Dock blir det svårt om det finns flera värden som har samma längd. Det skulle till exempel kunna vara om ett värde har ett startdatum och ett annat har ett slutdatum. Om XSD-filen är skriven med max och min längd samt att den bara får innehålla tal så går det kanske att göra en matchning mellan dessa två värden. Men det kräver bra matchnings strängar. Dessa problem kan undvikas om det bara är matchning mellan XML-dokument. Då kan taggar matchas mellan de olika dokumenten som ska skapas.

Att använda XSD-filer istället för att använda DTD-filer ger dock större möjligheter till att kunna matcha bättre. Det gör också att det blir mycket mer komplext. Jämförelser mellan DTD och XSD görs i kapitel 2.3. Det går även här se tydligt att DTD är begränsad och vilka fördelar XSD medför genom att det går att få in mer information i dessa taggar. En annan stor fördel med XSD är att det är en beskrivning i XML vilket inte DTD är.

De utökningar som XSLT 2.0 erbjuder gör att även transformationer mellan textfiler till XML eller till vilket format som helst utan att XSLT behöver utökas är en möjlighet. Detta gör att utvecklare inte behöver komma fram till sina egna lösningar utan flera personer kan testa och använda samma kod vilket gör att det får en mer stabilitet och fel kan lättare hittas.

Det som kan konstateras är att XSLT inte är helt trivialt och att det kräver experter för att kunna skriva de skript som krävs för att kunna utföra de önskade transformationerna. Det gäller inte bara komplexa skript utan även enklare skript är svåra att skriva. Om det dock går att kapsla in det i en grafisk miljö finns det potential att även icke-experter klarar av att skapa enklare transformationer. Det gör även att skapandet av transformationsskripten går snabbare att skapa och enkla misstag kan lättare undvikas.

6.1 Egna reflektioner

De reflektioner jag har fått fram är att ta fram XSLT-skript är tidsödande och det blir lätt fel. Detta gör att ju mer det går att automatgenerera och koppla samman delar via grafiska gränssnitt ju mindre fel blir det. Detta kräver dock att de grafiska gränssnitten stödjer alla de delar som en användare vill göra. Om detta inte är fallet så måste en expert in och göra det sista. Detta är också något jag konstaterade att om det finns en grafisk del så är det mycket lättare för en vanlig användare att använda det och en expert inom området behöver inte finnas på plats.

Ett problem som jag kom fram till var att XSD-skript kan byggas upp betydligt mer komplext än dess föregångare DTD. Detta har de fördelar som att mer information om vad för typ av värden som ska finnas i varje rad. Om det ska vara en viss längd på dessa med mera. Det gör att matchning mellan XSD och textfiler skulle kunna göras och kunna utesluta vissa värden. Det skulle även gå att göra någon typ av rankingsystem, att vissa värden förväntas vara mer troliga en andra. Detta skulle underlätta ytterliggare om det finns många värden i trädstrukturen. Ett problem som finns när matchning i textfiler ska göras är att det inte alltid finns förklaringar på vad

värderna i textfilen symboliserar. Detta är en stor nackdel som XML inte har eftersom alla taggarna ska vara beskrivande för vad varje värde är till för.

6.1.1 Förbättringar av prototypen

För att få prototypsystemet att användas ute hos företag behövs vissa utökningar och ändringar göras. Dels tas vissa begränsningar upp i 5.6 där dels multipla rader och begränsade XSD-strukturer tas upp som stora begränsningar i prototypen om den ska användas ute hos företag. En annan del som borde förbättras är en enklare wizzardvy för att lägga till attributen. Det fungerar som det är nu om man har en mindre mängd attribut. Men efter det börjar det bli rörigt vad som är ditlagt och hur de olika delarna hänger ihop. Något som vore bra är att kunna visa vad för del i textfilen som matchar vilken del i trädstrukturen. Det viktigaste som är en stor svaghet är att prototypsystemet inte klarar av multipla rader och rekusioner.

En del som vore viktig att gå vidare med är XSLT-optimering av prototypen. Detta beror på att dessa skript kommer att tas fram en gång men de kommer förmodligen att köras många gånger. Detta skript som används kommer även att i vissa fall arbeta på stora mängder data vilket gör att om den är optimerad så kan det finnas tidsvinster att göras.

6.2 Framtida Arbeten

I detta kapittel pressenteras några olika förslag till framtida arbeten som går att utföra efter detta examensarbete.

När XSD-filen läses in så sorteras bara element och attribut ut. prototypsystemet klarar inte av att följa länkar eller komplexa strukturer av XSD-träd. Det går att skriva typer som återanvänds vid flera olika ställen i en XSD-fil. Exempel på ett enkelt exempel finns i 2.4. Om det är välskrivna XSD-strukturer så kan storlek på element och attribut läsas ut. Det går till exempel att säga att ett visst attribut måste innehålla ett minst visst antal tecken till ett visst max antal tecken. Det går även att beskriva att ett visst attribut bara får innehålla tal eller bara bokstäver. Det går även att beskriva exakt hur en sträng ska representeras så som ett personnummer som först måste innehålla 8 siffror, sedan ett bindesträck och sist 4 stycken nummer. Dessa delar kan vara till hjälp när användaren ska mappa XSD-trädet till textfilen. Om användaren vet att vissa delar ska vara en viss längd så kan denna få hjälp med att matcha dessa värden.

För att få en uppfattning hur mycket nyutveckling som fortfarande använder textdatabaser istället för att använda XML-dokument från början bör en undersökning göras i detta område. Om allt fler går över till XML-dokument skulle helt automatgenererade rutiner för transformering kunna göras eftersom forskarna tar fram transformeringsrutiner och matchningsscheman för att kunna göra snabbare transformeringar helt automatiserat. Detta gör att om fler går över till XML-dokument och inte använder textdatabaser så kan ett utbyte mellan olika organisationer ske snabbare och helt automatiserat.

Referenser

A.Boukottaya, C.Vanoirbeek, F.Paganelli, & Abou, K. (2004). Automating XML documents transformations: a conceptual modelling based approach. Australian Computer Society, Inc., pp. 81-90.

Bob Lyons. (06). *The XFlat Language*. http://www.infoloom.com/gcaconfs/WEB/philadelphia99/lyons.HTM . 06. Ref Type: Electronic Citation. Hämtad 2006-06-03

C.F.Goldfarb (1981). A generalized approach to document markup. ACM Press, pp. 68-73.

Ce, D. & James, B. (2004). Static analysis of XSLT programs. Australian Computer Society, Inc., pp. 151-160.

Christian W.Dawson 2005, *Projects in Computing and Information Systems: A Student's Guide* Addison Wesley.

DataDirect Technologies. (06). *Stylus Studio*. http://www.stylusstudio.com/ . 06. Ref Type: Electronic Citation. Hämtad 2006-06-03

David Hunter, Kurt Cagle, & Chris Dix 2003, *Beginning XML, 2nd Edition: XML Schemas, SOAP, XSLT, DOM, and SAX 2.0* Wrox Press.

Doug tidwell 2001, *XSLT* O'Reilly & Associates, Inc..

Erik, T. R. & Christopher, R. M. 2001, *Learning XML* O'Reilly & Associates, Inc..

Georg, G., Christoph, K., Reinhard, P., & Luc, S. (2005). The complexity of XPath query evaluation and XML typing. *J.ACM*. vol. 52. nr. 2, pp. 284-335.

Giovanna, G., Marco, M., & Daniele, R. (2005). Impact of XML schema evolution on valid documents. ACM Press, pp. 39-44.

Hong, S., Harumi, K., & Elke, A. R. (2001). Automating the transformation of XML documents. ACM Press, pp. 68-75.

James Clark. (99). *XSL Transformations (XSLT) Version 1.0*. James Clark. http://www.w3.org/TR/xslt . 99. W3C. Ref Type: Electronic Citation. Hämtad 2006-06-03

Kevin, B., Don, C., Latha S.Colby, Fatma Özcan, Hamid, P., & Yu, X. (2005). Extending XQuery for analytics. ACM Press, pp. 503-514.

Luis Martín Díaz, Erik Wüstner, & Peter Buxmann (2002). Inter-organizational document exchange: facing the conversion problem with XML. ACM Press, pp. 1043-1047.

M.Berndtsson, J.Hansson, B.Olsson, & B.Lundell 2004, *Planning and Implementing your Final Year Projekt with Success!*, 2nd printing 2004 edn, Springer.

Michael Kay. (05). *XSL Transformations (XSLT) Version 2.0*. Michael Kay. http://www.w3.org/TR/2005/CR-xslt20-20051103/ . 05. W3C. Ref Type: Electronic Citation. Hämtad 2006-06-03

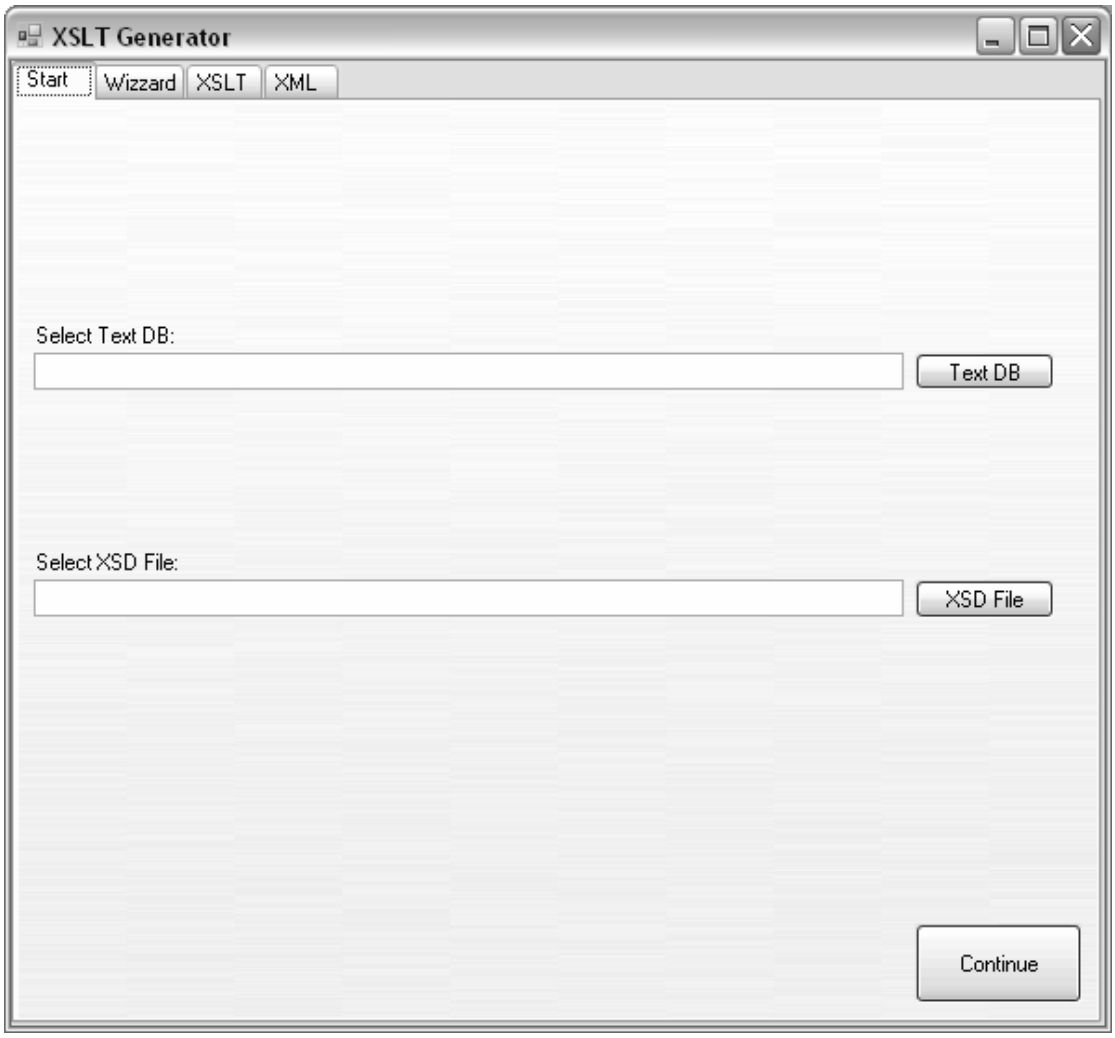
Paula, L. (2003). Automating XML document structure transformations. ACM Press, pp. 26-28.

Steven, J. D. (1999). XML linking. *ACM Comput.Surv*. vol. 31. nr. 4es, p. 21.

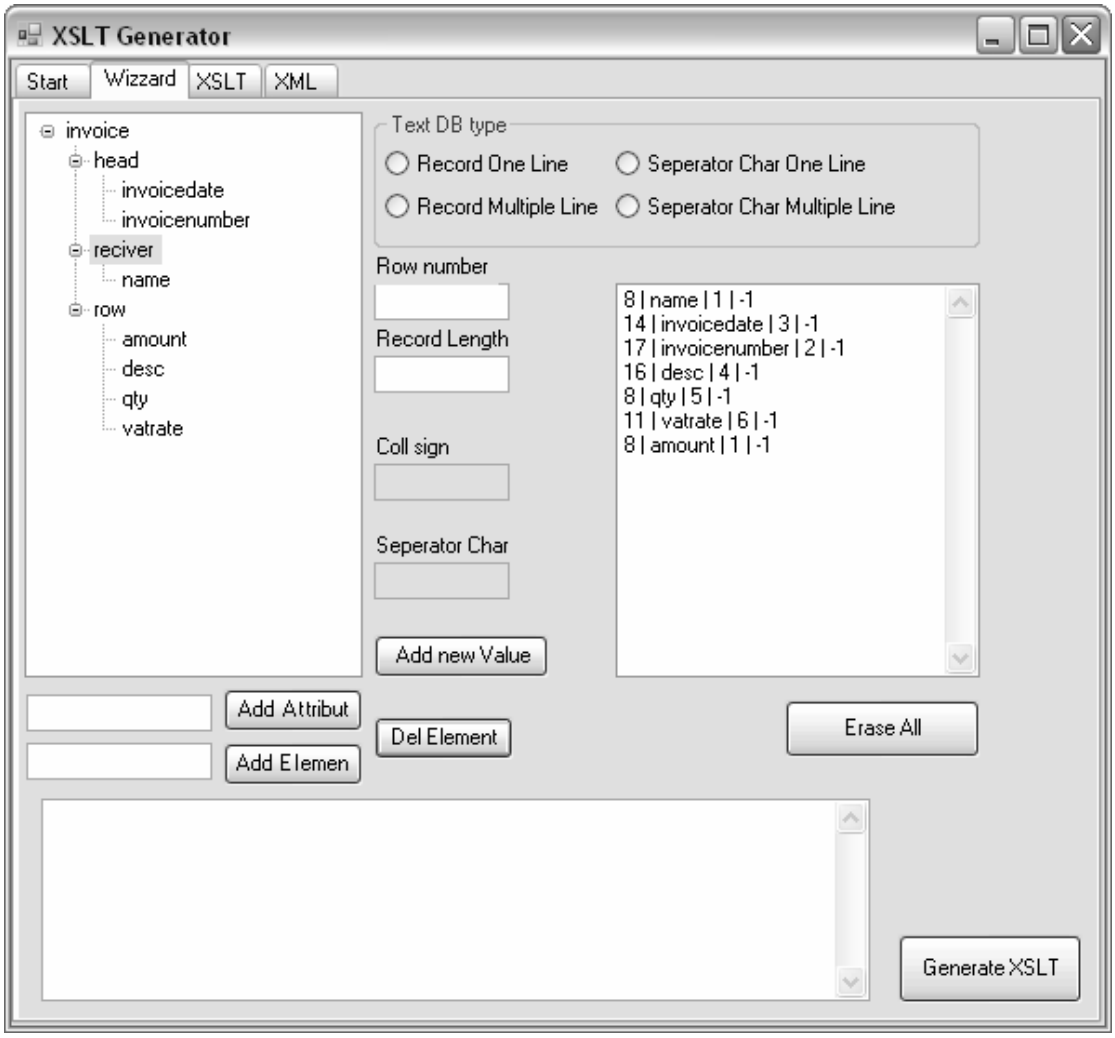
Appendix A – XML exempel kod

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<computerstore>
<computer category="Personal">
  <name>MPS4356</name>
  <company>IBM</company>
  <year>2005</year>
  <price>700.00</price>
</computer>
<computer category="Server">
  <name>MSV5437</name>
  <company>Compaq</company>
  <year>2002</year>
  <price>400.00</price>
</computer>
<computer category="personal">
  <name>MPS4327</name>
  <company>Compaq</company>
  <company>HP</company>
  <year>2004</year>
  <price>600.00</price>
</computer>
</computerstore>
```

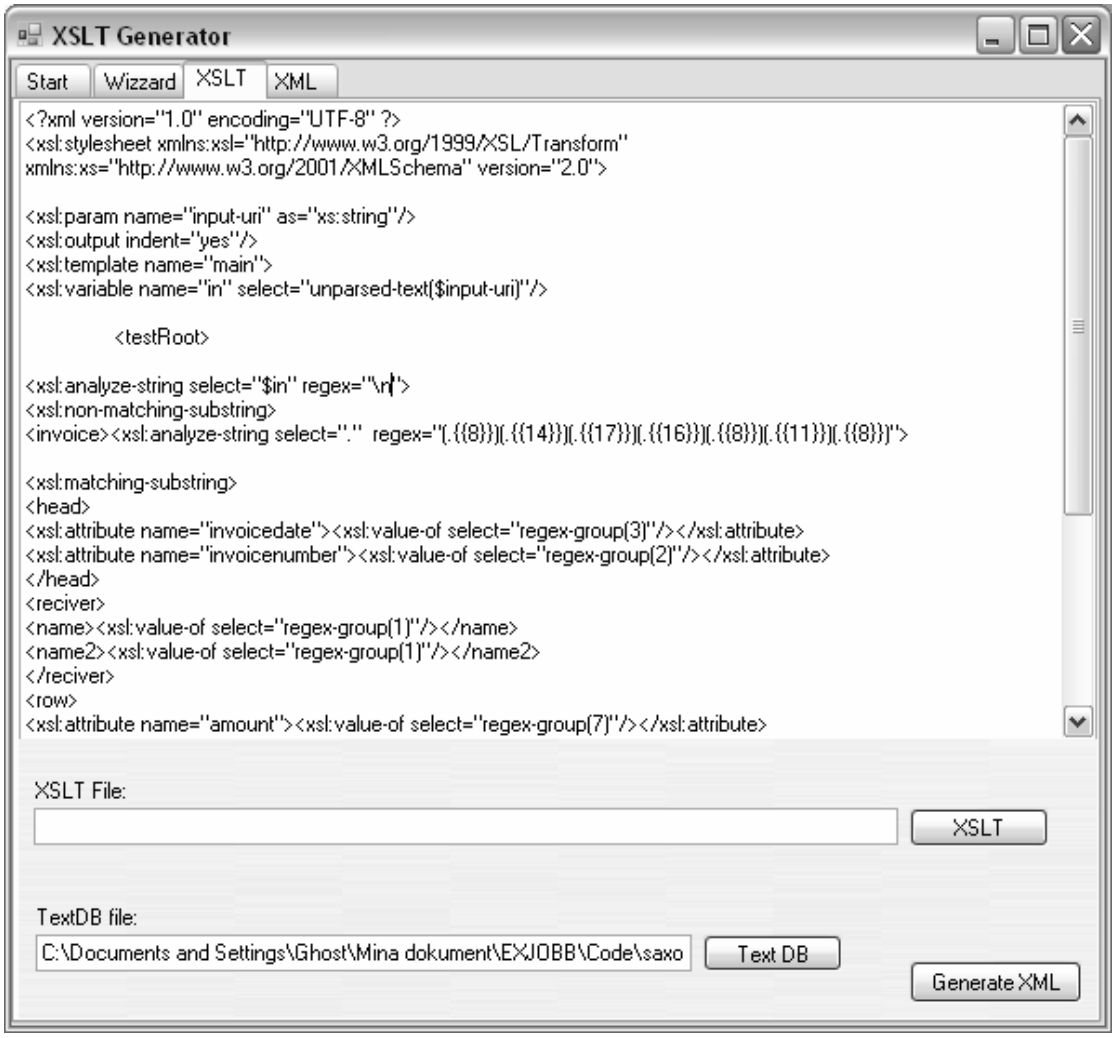
Appendix B – Skärmdumpar från prototypen



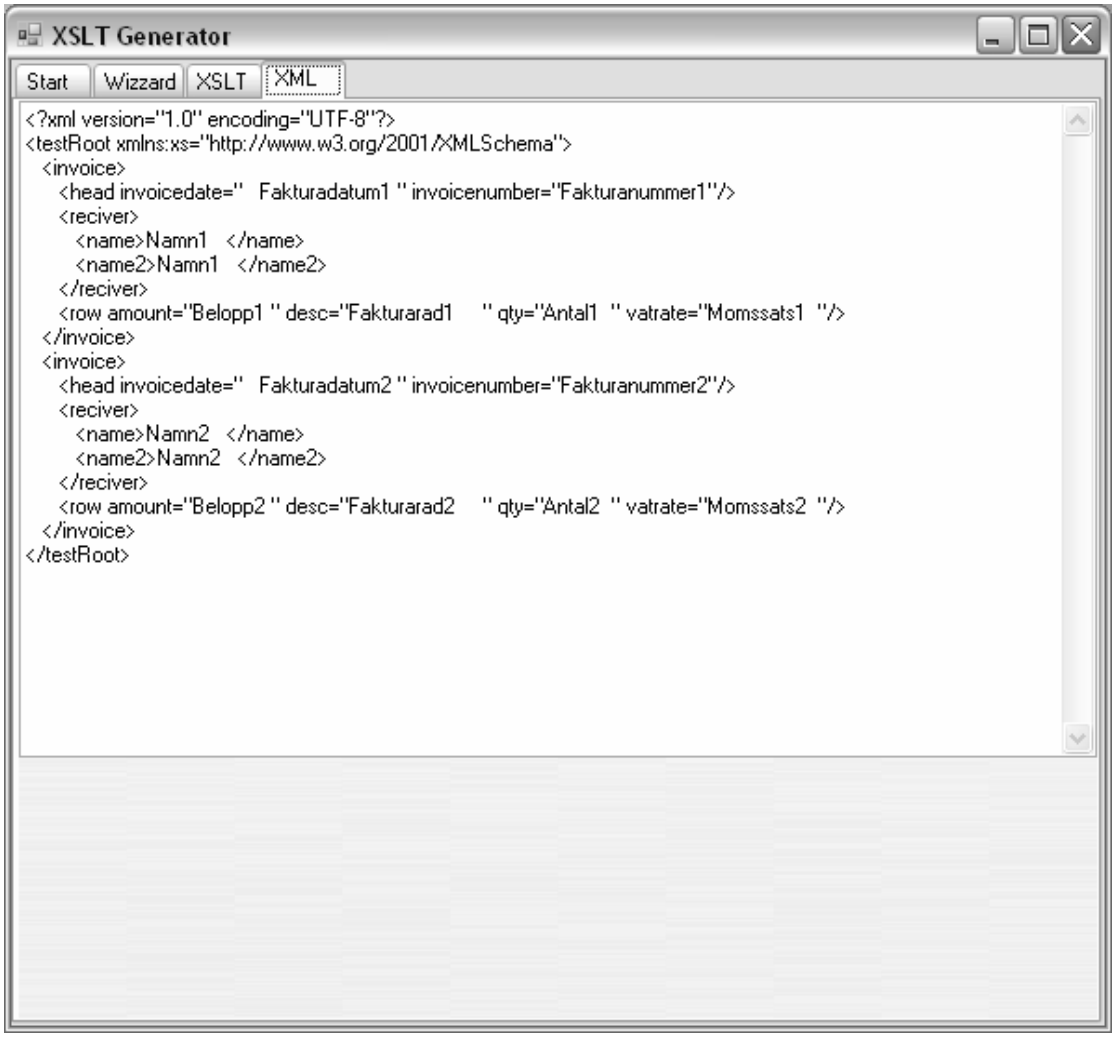
Skärmdump på första vyen av prototypen



Skärmdump på andra vyen i prototypen



Skärmdump på tredje vyen av prototypen



Skärmdump på fjärde vyen av prototypen

Appendix C – Exempel kod av XSD dokument

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema                                attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="test">
<xs:complexType>
          <xs:sequence>
               <xs:element maxOccurs="unbounded" name="invoice">
                    <xs:complexType>
                         <xs:sequence>
                              <xs:element name="head">
                                   <xs:complexType>
<xs:attribute name="invoicenumber" type="xs:string" use="required" />
                                        <xs:attribute  name="invoicedate"
type="xs:string" use="required" />
                                   </xs:complexType>
                              </xs:element>
                              <xs:element name="reciever">
                                   <xs:complexType>
                                        <xs:sequence>
     <xs:element maxOccurs="unbounded" name="name" type="xs:string" />
                                        </xs:sequence>
                                   </xs:complexType>
                              </xs:element>
     <xs:element maxOccurs="unbounded" name="row">
                                   <xs:complexType>
                                        <xs:attribute         name="desc"
type="xs:string" use="required" />
                                        <xs:attribute          name="qty"
type="xs:string" use="required" />
                                        <xs:attribute       name="amount"
type="xs:string" use="required" />
                                        <xs:attribute      name="vatrate"
type="xs:string" use="required" />
                                   </xs:complexType>
                              </xs:element>
                         </xs:sequence>
                    </xs:complexType>
               </xs:element>
          </xs:sequence>
     </xs:complexType>
</xs:element>
</xs:schema>
```

XSD diagramet som används för att utvärdera om utdata är korrekt.

Appendix D – Exempel av textfiler

```
Namn1;;Fakturanummer1:1;Fakturadatum1:1;Fakturarad1:1;Antal1:1;
Momssats1:1;Belopp1:1
Namn2:1;Namn2:2;Fakturanummer2;Fakturadatum2;Fakturarad2;Antal2;
Momssats2;Belopp2
```

Testdata med komma separering

```
Namn1   Fakturanummer1   Fakturadatum1 Fakturarad1     Antal1  Momssats1
Belopp1
Namn2   Fakturanummer2   Fakturadatum2 Fakturarad2     Antal2  Momssats2
Belopp2
```

Testdata som är record baserad

Appendix E – XSLT utdata

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" version="2.0">

<xsl:param name="input-uri" as="xs:string"/>
<xsl:output indent="yes"/>
<xsl:template name="main">
<xsl:variable name="in" select="unparsed-text($input-uri)"/>

     <testRoot>

<xsl:analyze-string select="$in" regex="\n">
<xsl:non-matching-substring>
<invoice><xsl:analyze-string                                        select="."
regex="(.{{8}})(.{{14}})(.{{17}})(.{{16}})(.{{8}})(.{{11}})(.{{8}})">

<xsl:matching-substring>
<head>
<xsl:attribute      name="invoicedate"><xsl:value-of      select="regex-
group(3)"/></xsl:attribute>
<xsl:attribute     name="invoicenumber"><xsl:value-of     select="regex-
group(2)"/></xsl:attribute>
</head>
<reciver>
<name><xsl:value-of select="regex-group(1)"/></name>
</reciver>
<row>
<xsl:attribute         name="amount"><xsl:value-of         select="regex-
group(7)"/></xsl:attribute>
<xsl:attribute          name="desc"><xsl:value-of          select="regex-
group(4)"/></xsl:attribute>
<xsl:attribute           name="qty"><xsl:value-of          select="regex-
group(5)"/></xsl:attribute>
<xsl:attribute        name="vatrate"><xsl:value-of        select="regex-
group(6)"/></xsl:attribute>
</row>
</xsl:matching-substring>
</xsl:analyze-string>
</invoice>
</xsl:non-matching-substring>
</xsl:analyze-string>
     </testRoot>

</xsl:template>
</xsl:stylesheet>
```

XSLT scriptet med record baserad textfil

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" version="2.0">

<xsl:param name="input-uri" as="xs:string"/>
<xsl:output indent="yes"/>
<xsl:template name="main">
<xsl:variable name="in" select="unparsed-text($input-uri)"/>

     <testRoot>

<xsl:analyze-string select="$in" regex="\n">
<xsl:non-matching-substring>
<invoice>
<head>
<xsl:attribute    name="invoicedate"><xsl:for-each    select="tokenize(.,
';')"><xsl:if         test="position()         =         4"><xsl:value-of
select="current()"/></xsl:if></xsl:for-each></xsl:attribute>
<xsl:attribute   name="invoicenumber"><xsl:for-each   select="tokenize(.,
';')"><xsl:if         test="position()         =         3"><xsl:value-of
select="current()"/></xsl:if></xsl:for-each></xsl:attribute>
</head>
<reciver>
<name><xsl:for-each select="tokenize(., ';')"><xsl:if test="position() =
1"><xsl:value-of select="current()"/></xsl:if></xsl:for-each></name>
</reciver>
<row>
<xsl:attribute      name="amount"><xsl:for-each      select="tokenize(.,
';')"><xsl:if         test="position()         =         8"><xsl:value-of
select="current()"/></xsl:if></xsl:for-each></xsl:attribute>
<xsl:attribute       name="desc"><xsl:for-each       select="tokenize(.,
';')"><xsl:if         test="position()         =         5"><xsl:value-of
select="current()"/></xsl:if></xsl:for-each></xsl:attribute>
<xsl:attribute        name="qty"><xsl:for-each        select="tokenize(.,
';')"><xsl:if         test="position()         =         6"><xsl:value-of
select="current()"/></xsl:if></xsl:for-each></xsl:attribute>
<xsl:attribute      name="vatrate"><xsl:for-each      select="tokenize(.,
';')"><xsl:if         test="position()         =         7"><xsl:value-of
select="current()"/></xsl:if></xsl:for-each></xsl:attribute>
</row>
</invoice>
</xsl:non-matching-substring>
</xsl:analyze-string>
     </testRoot>

</xsl:template>
</xsl:stylesheet>
```

XSLT skriptet med seperations tecken textfil

Appendix F – XML utdata

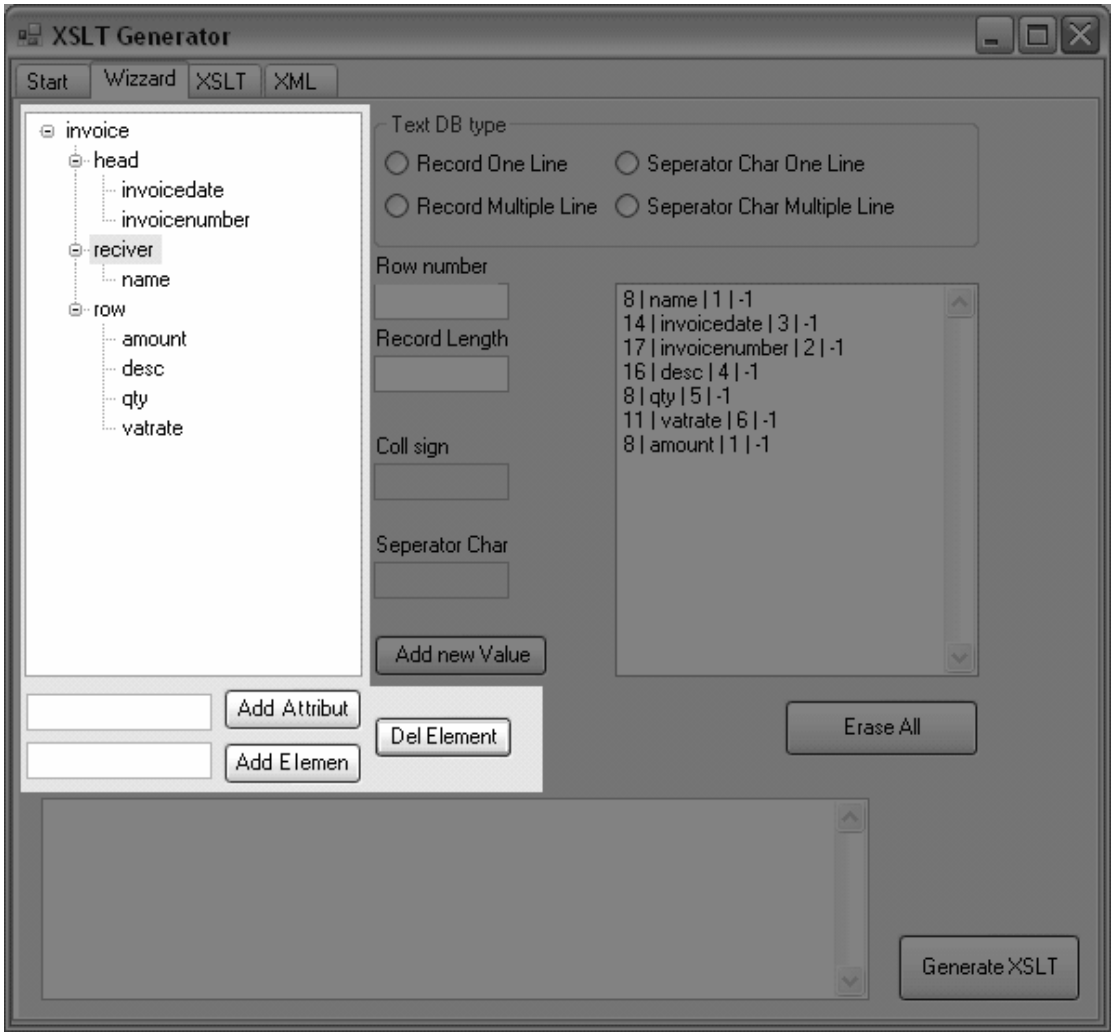
```
<?xml version="1.0" encoding="UTF-8"?>
<testRoot xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <invoice>
      <head       invoicedate="                    Fakturadatum1       "
invoicenumber="Fakturanummer1"/>
      <reciver>
         <name>Namn1   </name>
      </reciver>
      <row  amount="Belopp1  "  desc="Fakturarad1        "  qty="Antal1   "
vatrate="Momssats1  "/>
   </invoice>
   <invoice>
      <head       invoicedate="                    Fakturadatum2       "
invoicenumber="Fakturanummer2"/>
      <reciver>
         <name>Namn2   </name>
      </reciver>
      <row  amount="Belopp2  "  desc="Fakturarad2        "  qty="Antal2   "
vatrate="Momssats2  "/>
   </invoice>
</testRoot>
```

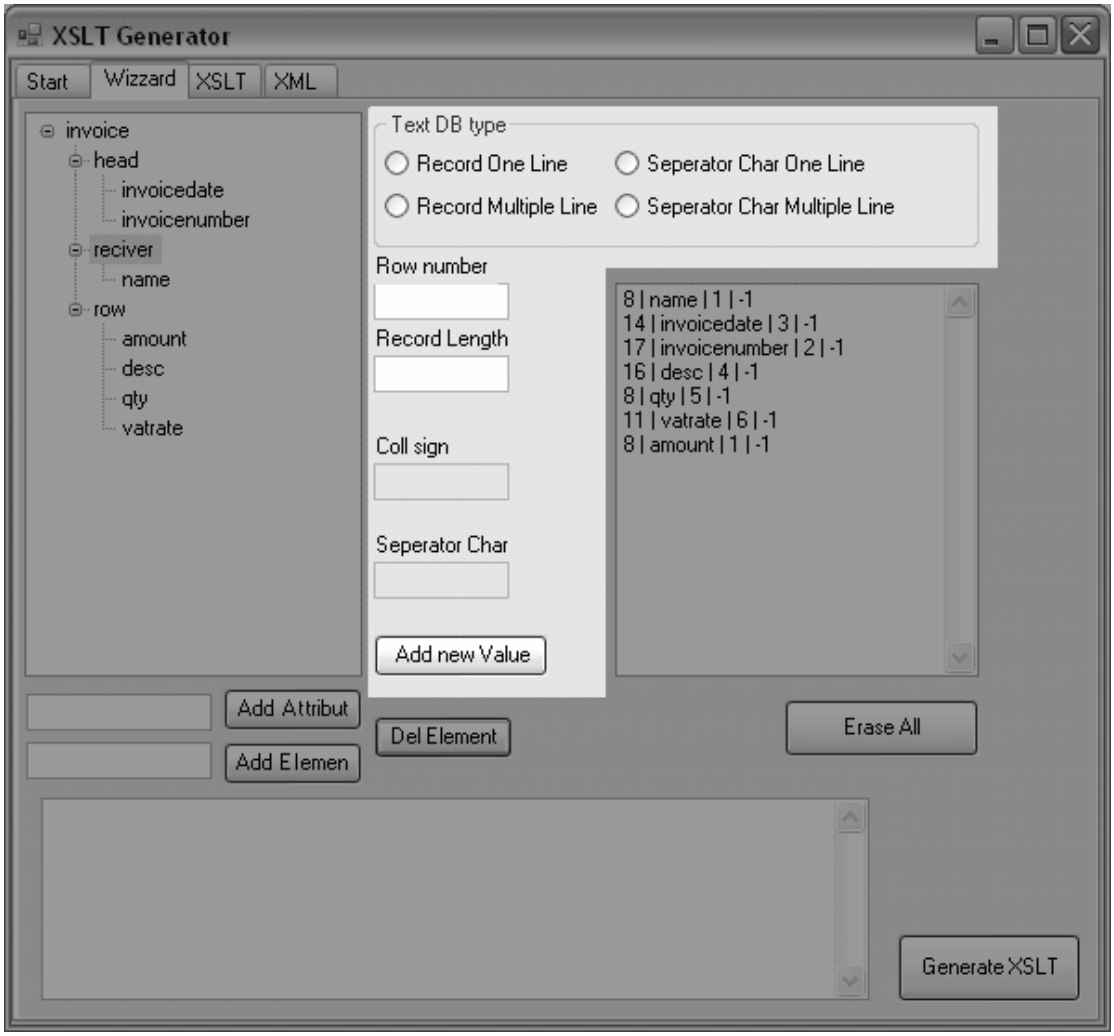
XML data med record baserad textfil

```
<?xml version="1.0" encoding="UTF-8"?>
<testRoot xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <invoice>
      <head                                  invoicedate="Fakturadatum1:1"
invoicenumber="Fakturanummer1:1"/>
      <reciver>
         <name>Namn1</name>
      </reciver>
      <row  amount="Belopp1:1&#xD;"  desc="Fakturarad1:1"  qty="Antal1:1"
vatrate="Momssats1:1"/>
   </invoice>
   <invoice>
      <head invoicedate="Fakturadatum2" invoicenumber="Fakturanummer2"/>
      <reciver>
         <name>Namn2:1</name>
      </reciver>
      <row     amount="Belopp2"     desc="Fakturarad2"     qty="Antal2"
vatrate="Momssats2"/>
   </invoice>
</testRoot>
```

XML data med separations textfil

Appendix G – Wizzard vyn




XSLT Generator
Start
Wizzard
XSLT
XML
invoice
head
invoicedate
invoicenumber
reciver
name
row
amount
desc
qty
vatrate
Text DB type
Record One Line
Seperator Char One Line
Record Multiple Line
Seperator Char Multiple Line
Row number
Record Length
Coll sign
Seperator Char
Add new Value
8 | name | 1 | -1
14 | invoicedate | 3 | -1
17 | invoicenumber | 2 | -1
16 | desc | 4 | -1
8 | qty | 5 | -1
11 | vatrate | 6 | -1
8 | amount | 1 | -1
Add Attribut
Add Elemen
Del Element
Erase All
Generate XSLT

Appendix H – Flödesdiagram övder prototypen