HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i Informationssystemutveckling 20p
C-nivå
Vårterminen 2005

Flexibel extraktion av data från XMI-dokument

utan tillgång till DTD eller XML-schema

Linus Koszinowski

HS-IKI-EA-05-308

Flexibel extraktion av data från XMI-dokument
- utan tillgång till DTD eller XML-schema

Examensrapport inlämnad av Linus Koszinowski till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information.

2005-06-07

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: _____

Handledare för examensarbetet: Henrik Gustavsson

Flexibel extraktion av data från XMI-dokument
- utan tillgång till DTD eller XML-schema

Linus Koszinowski

Sammanfattning

Undersökningen behandlar problem som tidigare forskning har uppmärksammat kring användandet av standarden XMI. XMI är avsedd för utbyte av modeller mellan modelleringsverktyg. Problem som uppmärksammats i användningen av XMI är att information om modeller inte alltid utbyts korrekt mellan verktyg. Detta kan resultera i att viktig information om modellerna går förlorad.

Undersökningen har till syfte att undersöka huruvida det är möjligt att extrahera data från modeller sparade i XMI-format utan tillgång till DTD eller XML-schema. En sådan möjlighet skulle kunna användas för att rädda information om modeller som annars skulle gå förlorad.

Den metod som har används för undersökningen är litteraturstudie, experiment och implementation samt hypotesprövning. Studien har resulterat i en implementation som är avsedd för att extrahera och presentera information från XMI-dokument.

Den slutsats som dras från undersökningen är att det går att extrahera data från modeller sparade i XMI-dokument förutsatt att sättet på vilken XMI-standarden har tillämpats är känt sedan tidigare.

Nyckelord: XMI, UML, XML

Innehållsförteckning

1 Introduktion 1

2 Bakgrund 2

2.1 UML 2

2.1.1 Diagram 3

2.2 XML 4

2.2.1 XML-dokument 6

2.2.2 DTD och XML-schema 7

2.2.3 XML-Länkar 8

2.2.4 Representation av data i XML 8

2.2.5 XML i applikationer 9

2.3 XMI 10

2.3.1 Versioner 11

3 Problem 12

3.1 Problemområde 12

3.2 Problemprecisering 14

3.3 Avgränsningar 14

4 Metod 16

4.1 Metodval 16

4.1.1 Hypotesprövning 16

4.1.2 Litteraturstudie 16

4.1.3 Experiment och implementation 16

5 Genomförande 18

5.1 Litteratur- och teoristudier 18

5.2 Experiment 18

5.3 Implementation 22

5.4 Hypotesprövning 22

6 Resultat och analys 23

6.1 Delmål 1 - Rangordning av UML-element 23

6.2 Delmål 2 - Val av lämplig implementationsplattform 23

6.3 Delmål 3 – Konfigurationsmöjligheter 24

6.3.1 Klassexperimenten 25

6.3.2 Associationsexperiment 26

6.3.3 Arvsexperimenten 27

6.3.4 Sammanfattning delmål 3 28

6.4 Delmål 4 - Implementation 28
6.5 Hypotesprövning 32
6.6 Slutsats 33
7 Diskussion 34
7.1 Kritisk granskning av arbetet 34
7.2 Diskussion runt resultatet 34
7.3 Fortsatt arbete 35
Referenser 36
Bilaga 1 Klassdiagram
Bilaga 2 Extractor
Bilaga 3 XMIDocument
Bilaga 4 XMIObjectList
Bilaga 5 XMIObject
Bilaga 6 XMIHelper
Bilaga 7 XMLFileFilter

1 Introduktion

Det sägs att en bild säger mer än tusen ord. I fallet med grafiska modeller och diagram av informationssystem är det i högsta grad sant. Ett diagram kan förklara ett system som skulle kräva tusentals ord att beskriva. Det behöver inte heller bli mer lättförståeligt för att systemet förklaras i ord, snarare tvärt om. En förklarande text kan, särskilt om den är tekniskt avancerad eller krånglig, missförstås eller feltolkas. En grafisk framställning kan istället hjälpa till att sammanfatta och tydliggöra viktig information som annars skulle riskera att döljas i exempelvis en förklarande text (Nationalencyklopedin, 2005).

Grafiska modeller kan vara ytterst användbara för dokumentationen av informationssystem (Alhir, 1998). Det tar förhållandevis kort tid att ögna igenom en grafisk modell för att få en grundläggande förståelse för ett systems utformning i förhållande till att läsa ett antal sidor skriftlig dokumentation. Finns det grafiska modeller att titta på kan dessa underlätta förståelsen för ett systems programkod, något som annars kan var både tidskrävande och krångligt. Dessa grafiska modeller över informationssystem skapas oftast i program som kallas för modelleringsverktyg (Persson, 2004). I ett modelleringsverktyg ritas informationssystemet upp med hjälp av symboler som visar systemets beståndsdelar samt deras inbördes relationer. De modeller som skapas i ett visst modelleringsverktyg är ofta specifika såtillvida att de endast kan hanteras av just det verktyget. Det kan alltså vara svårt att öppna en fil i ett annat program än det som använts för att skapa filen.

Det är ett generellt problem inom datavärlden att information hanteras på olika sätt (Nader, 1998). Ett sätt att bemöta detta problem är att använda sig av en standard. Inom mjukvaruindustrin finns det en uppsjö exempel på olika standarder, till exempel UML (Unified Modeling Language), XML (eXtensible Markup Language) och XMI (XML Metadata Interchange).

XMI är en standard avsedd för att göra det möjligt att utbyta modeller mellan modelleringsverktyg (Grose, Doney & Brodsky, 2002). Att denna standard finns innebär dock ingen garanti för att det faktiskt går att överföra en modell mellan olika verktyg. Detta beror på att olika modelleringsverktyg hanterar informationen på så vitt skilda sätt. Detta betyder att problemet kvarstår, det vill säga trots att det finns en standard kan modellerna ändå inte användas tillfullo i andra verktyg än dem som skapat dem.

2 Bakgrund

I bakgrunden presenteras de olika standarder som på olika sätt förekommer i den här undersökningen. I kapitel 2.1 beskrivs UML, följt av en redogörelse av XML i kapitel 2.2. Slutligen handlar kapitel 2.3 om XMI.

2.1 UML

Object Management Group (OMG, 2003a) menar att det är lika viktigt för utveckling och vidareutveckling av informationssystem att ha en modell av systemet som det är att ha ritningar innan ett hus byggs. OMG är ett ideellt konsortium för att skapa specifikationer för dataindustrin.

UML är en industristandard som idag förvaltas och vidareutvecklas av Object Management Group (OMG). UML står för Unified Modeling Language och är ett visuellt språk för att beskriva, utveckla och dokumentera framförallt informationssystem. Det går även att använda UML för att exempelvis modellera processer i en affärsverksamhet eller andra inte nödvändigtvis mjukvarubaserade system (OMG, 2003a). UML består av delar som visat sig fungera väl från andra objektorienterade modelleringsspråk (OMG, 2003a).

Avsikten med UML är att tillhandahålla användare med ett uttrycksfullt visuellt modelleringsspråk för att utveckla och utbyta meningsfulla modeller (OMG, 2003a). Tanken är också att UML ska vara enkelt och precist så att det genast går att börja använda i utvecklingsprojekt (Alhir, 2003).

UML består av en notation och en metamodell (Fowler, 2000). Notationen är de grafiska detaljerna som syns i modellerna. De symboler som används för att representera klasser, associationer, arv och så vidare. Notationen är själva syntaxen i modelleringsspråket (Fowler, 2000). Det finns statiska element (klasser, attribut och relationer) och dynamiska element (objekt, meddelanden). Metamodellen beskriver relationerna mellan de olika symbolerna och hjälper till att specificera hur en välstrukturerad modell som är syntaktiskt korrekt ska se ut (Fowler, 2000). Metamodellen är ibland en beskrivning av UML i just UML (Fowler, 2000).

Tack vare ett gemensamt språk går det att kommunicera kring ett problem eller problemområde. Kommunikation kring exempelvis systemkrav för ett informationssystem kan i vanligt språk bli alltför oprecist och rörigt om systemet som ska förklaras är komplext. Programkod däremot är precis, men alltför detaljerad (Fowler, 2000). UML och modeller kan då vara en bra mellanväg. Språket gör det möjligt att samarbeta och kommunicera kring olika delar i systemet (Alhir, 2003).

En modell är en representation av något som är gjord med avsikten att förenkla. En modell behöver inte vara fullständig utan endast, för uppgiften, relevant information behöver finnas med. UML och modeller kan användas för att förtydliga viktiga delar av system (Fowler, 2000). Modellerna används då det annars skulle vara svårt att uppfatta eller förstå ett komplext system i sin helhet. Kring modeller av systemet går det att skapa en gemensam förståelse för kraven och systemet som helhet (Alhir, 2003).

Det finns en rad olika versioner av UML-specifikationen. Den första officiella versionen, UML 1.0 , kom 1997 (Fowler, 2000). Den nuvarande officiella versionen är 2.0 och skiljer sig på många sätt från den tidigare versionen UML 1.5 (OMG, 2003a; OMG, 2003c; OMG, 2004).

2.1.1 Diagram

UML använder sig av tolv olika typer av diagram för att ur olika vinklar åskådliggöra olika delar i ett system och deras samband till varandra (OMG, 2003a). De flesta UML-diagram är 2-dimensionella grafiska framställningar som är uppbyggda av ikoner, symboler, relationer (*paths)* och textinformation (OMG, 2003a). De olika diagrammen är uppdelade i tre huvudgrupper; strukturella diagram, beteendediagram och styrningsdiagram.[1]

Ett exempel på ett UML-diagram som är mycket vanligt förekommande är klassdiagrammet (Fowler, 2000). Klassdiagram används för att avbilda strukturen av ett program, hur de olika klasserna i programmet hänger ihop och relaterar till varandra samt vilka egenskaper de har (Alhir, 2003). Några av de element som klassdiagram utgörs av är klasser, associationer och arv. Genom att koppla ihop de olika elementen på olika sätt går det att bygga upp en statisk mall över hur olika klasser i programmet relaterar till varandra (OMG, 2003a).

Klasser representerar entiteter som har samma egenskaper och beteende (Alhir, 1998). Dessa får de bland annat genom *attribut* och *operationer*. En klass symboliseras av tre olika boxar staplade på varandra. Den översta innehåller klassnamnet, den mellersta attributen och den undre operationer.

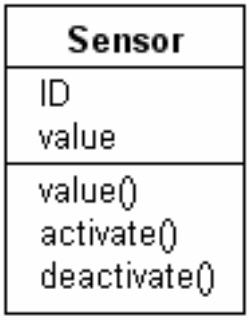


Fig. 2.1. Exempel på en klass

Associationer representerar relationer mellan två eller flera klasser (Alhir, 1998). De symboliseras av ett streck mellan de två associerade klasserna.

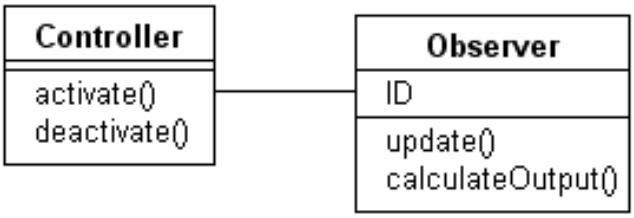


Fig. 2.2. Exempel på en association mellan två klasser

[1] Tillgänglig på Internet: http://www.omg.org/gettingstarted/what_is_uml.htm [Hämtad 2005-06-04]

Arv, eller generalisering. Barnklasser ärver egenskaper från sin ”förälder”.

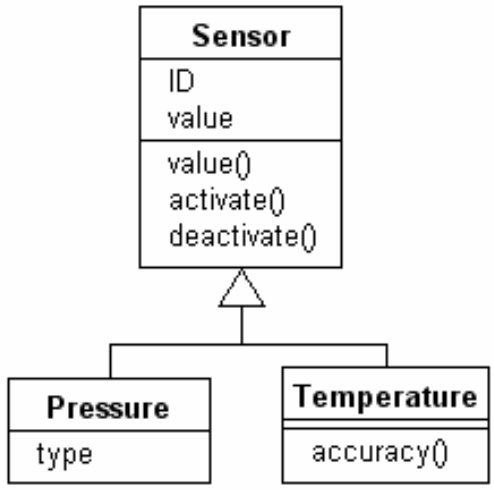


Fig. 2.3. Exempel på en arvshierarki.

2.2 XML

XML står för e**X**tensible **M**arkup **L**anguage och är en förhållandevis ny företeelse i datavärlden. World Wide Web Consortium (W3C) är det standardiseringsorgan som ansvarar för och tillhandahåller XML.

XML är en standard för att strukturera och utbyta data över webben (Elmasri & Navathe, 2004) och kan beskrivas som en fortsättning på HTML (Åström, 1999; Azelia, 2002). HTML är den teknik som huvudsakligen används för att formatera och strukturera hemsidor på Internet. XML har dock många fler användningsområden än HTML, eftersom det inte är bundet till något visst applikationsområde. XML kan användas närhelst man vill representera strukturerad information på ett generellt och standardiserat vis (Azelia, 2002).

Både XML och HTML är så kallade markeringsspråk och sådana använder ”taggar” för att märka upp olika typer innehåll i en datamängd. Uppmärkningen hjälper program att hantera informationen i dokument, exempelvis talar HTML-taggarna om för en webbläsare hur den ska visa informationen på en hemsida.

HTML och XML använder samma sätt, samma typ av taggar, för att märka upp informationen. En starttagg identifieras av text inom vinkelparenteser, ”<…>”, därefter följer den uppmärkta informationen som avslutas med en sluttagg. En sluttagg identifieras av startaggens namn med ett inledande ”/”-tecken inom vinkelparenteser; </…>. Nedan visas ett mycket enkelt exempel på hur informationen i en HTML-sida kan märkas upp:

```
<HTML>
     <HEAD>
          <TITLE>Sidans titel</TITLE>
     </HEAD>
<BODY>
     <H1>Rubrik – H står för engelskans Header, 1 står
     för att det handlar om den största/första
     rubriken</H1>
     <P>Detta är ett textstycke. P står för engelskans
     Paragraph</P>
     <P>Ytterligare ett textstycke</P>
```

```
</BODY>
</HTML>
```

En tagg kan också innehålla *attribut* vilket är ytterligare information som är förknippad med en viss tagg. Ett attribut anges i starttaggen och har ett namn och ett värde. En tagg kan innehålla ett godtyckligt antal olika attribut.

```
Exempel:
<tagg attribut="värde"></tagg>
<bok titel="bokens titel">
     ...
     Bokens innehåll
     ...
</bok>
```

Både HTML och XML härstammar från SGML (Standard Generalized Markup Language). SGML togs fram på 1980-talet för att tillgodose grundläggande behov av att kunna göra datalagring oberoende av speciella programpaket eller specifika programleverantörer (North & Hermans, 1999).

SGML är mycket kraftfullt och avancerat vilket gör det svårt och tidsödande att utveckla i (Åkerström, 1999). Det har gjorts försök att anpassa SGML till webben men dessa har inte lyckats (North & Hermans, 1999). Försöken har blivit för dyra, för tids- och kunskapskrävande samt alltför komplicerade för att göra SGML lämpligt att användas för webben. Därför används HTML som är ett enklare format för att beskriva och skapa struktur för textinnehållet som ska presenteras på en hemsida. En av orsakerna till att webben har blivit så populär är att det i HTML-språket går förhållandevis snabbt och enkelt att skapa hemsidor.

HTML, liksom SGML, innehåller dock brister som skapat behov av ytterligare ett markeringsspråk. Exempel på brister är:

- HTML blandar innehåll och presentation. Nästan all information som visas på nätet är uppblandad med information om hur informationen ska presenteras, som exempelvis vilket typsnitt som ska användas, storleken på bokstäver, bakgrundsfärg etc. Detta gör att koden blir svår att återanvända, eftersom den är specifikt skriven för att informationen ska presenteras på ett visst sätt (North & Hermans, 1999; Åström, 1999). Det gör det svårt att till exempel återanvända ett dokument för att presentera informationen på en annan plattform, exempelvis mot en mobiltelefons begränsade skärm, utan hela dokumentet måste skrivas om och anpassas för att passa den nya plattformen.
- HTML passar inte för datautbyte eftersom dess taggar i liten utsträckning används för att identifiera innehållet i dokumentet (North & Hermans, 1999). Exempelvis informerar *Paragraph*-taggen endast att det rör sig om ett textstycke. Den säger ingenting om innehållet i textstycket, till exempel om det är ett recept, en recension eller en roman. Detta faktum gör att HTML inte passar för att representera strukturerad data som exempelvis hämtas från en databas eftersom informationen om den information som hämtats är allt för bristfällig (Elmasri & Navathe, 2004).
- HTML har en fast uppsättning taggar som är standardiserade och klara både till namn och funktion (Åkerström, 1999). Finns inte den tagg eller kombination av taggar som uppfyller det du har tänkt dig göra måste du kanske kompromissa eller helt avstå från det du tänkt dig göra.
- I HTML finns det ingen möjlighet att återanvända information flera gånger på samma sida. Vill du exempelvis använda huvudrubrikens text även som titel för sidan måste texten skrivas två gånger på två olika platser i dokumentet.

```
Exempel:
<HTML>
<HEAD>
     <TITLE>Huvudrubrikens text</TITLE>
</HEAD>
<BODY>
     <H1>Huvudrubrikens text</H1>
</BODY>
</HTML>
```

Eftersom både HTML och SGML har brister, skapades XML som ett mellanting mellan SGML och HTML. XML förenar enkelheten hos HTML med flexibiliteten och styrkan hos SGML (Åström, 1999).

Översatt till svenska betyder *extensible* utbyggbart. Med detta menas att det går att hitta på egna taggar. Utbyggbarheten löser två av problemen med HTML:

- Till skillnad från HTML, som har ett fast antal taggar, går det att hitta på nya taggar för att passa behov som gamla taggar inte täcker (Åström, 1999).
- I och med att du själv kan hitta på egna taggar kan du också ge dina taggar beskrivande namn för att ge en fingervisning om innehållet i taggen. På så sätt går det att göra XML-dokument självbeskrivande eftersom namnen på taggarna beskriver innehållet.

I XML går det att återanvända data på flera olika ställen och i olika sammanhang på en sida utan att för den skull behöva skriva ut informationen upprepade gånger. Detta går inte att göra i HTML. Vill du visa samma information på flera olika ställen måste du även skriva ut informationen flera gånger, vilket exemplifierades tidigare.

XML skiljer också på innehåll och presentation. Ett XML-dokument innehåller endast strukturerad information. Hur informationen ska presenteras beskrivs i en för XML-dokumentet tillhörande stilmall. En stilmall är ett dokument som bestämmer hur XML-dokumentet kommer att se ut på exempelvis en hemsida, på papper eller i en mobiltelefon.

2.2.1 XML-dokument

Allting i XML betraktas som dokument. Begreppet används logiskt och ska inte tolkas riktigt på samma sätt som i vardagligt tal. Med dokument menas en uppsättning information i form av element, minst ett, och inte nödvändigtvis något som kan läsas av människor eller skrivas ut på papper (Azelia, 2002).

Ett XML-dokument består av ett antal noder, *Nodes*. Element, attribut, kommentarer och text är alla exempel på noder.

Det finns ett antal regler för hur ett XML-dokument måste vara uppbyggt. När dessa regler är uppfyllda, det vill säga dokumentet är syntaktiskt korrekt, sägs dokumentet vara ”well-formed” eller välformat. Några av dessa regler som ett dokument måste uppfylla är:

- Ett XML-dokumentet måste inledas av en XML-deklaration som bland annat talar om vilken version av XML som används. En deklaration kan se ut på följande sätt:

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

- Ett XML-dokument måste bestå av ett, och *endast* ett, så kallat rot-element som kan innehålla ett eller flera andra element. På det här viset skapas en

hierarkisk trädstruktur (Elmasri & Navathe, 2004). Hierarkin som skapas påminner om abstraktionen för hur filer organiseras i ett filsystem i operativsystem som exempelvis Windows och UNIX, det vill säga att mappar eller kataloger kan innehålla filer men också ytterligare mappar som på så sätt bildar trädstrukturer.

- Alla element måste avslutas. Det går att avsluta taggar på två olika sätt. Antingen genom det vanliga sättet med en sluttagg eller, om elementet är tomt går det att istället avsluta det med ett ”/”-tecken sist i taggen.

```
Exempel:
Start och sluttagg:
<element>info</element>
Exempel på hur det går att avsluta en tom tagg:
<tomt_element värde=”1” />
```

- Element får inte överlappa varandra. Ett element får inte avslutas förrän alla element det innesluter också har avslutats, det vill säga elementen måste i så fall omsluta varandra. Hade element tillåtits överlappa varandra skulle det inte vara möjligt att skapa hierarkiska trädstrukturer.

```
Exempel:
Tillåtet eftersom elementen inte överlappar:
<parent>
      <child></child>
</parent>

Inte tillåtet! Elementen överlappar varandra.
<parent>
      <child>
</parent>
      </child>
```

Ett element som omsluter andra element kallas för komplexa, innehåller det bara data och inga andra element kallas elementet för enkelt (Elmasri & Navathe, 2004).

2.2.2 DTD och XML-schema

XML kan vara självbeskrivande på så sätt att elementens namn kan ge en fingervisning om vilken typ av information som elementet innehåller och detta kan räcka ganska långt (Azelia, 2002). I vissa fall kan det dock uppstå problem om taggar tillåts alltför godtyckligt. I applikationer som använder XML kan det till exempel vara smart att låsa de XML-dokument som används till att endast kunna ha ett visst innehåll och struktur (Azelia, 2002). En applikation vars beteende ändras beroende på vilka element den stöter på skulle få problem om det plötsligt skulle dyka upp element som den inte känner igen och vet den inte hur den ska hantera.

DTD och XML-schema är så kallade dokumentdefinitioner och dessa används för att bestämma vilka element som får finnas i ett XML-dokument. En dokumentdefinition dikterar också hur ett dokument får byggas upp, till exempel vilka element som får innehålla vilka och i vilken ordning de får förekomma etc.

DTD står för Document Type Definition och är ett arv från SGML (Azelia, 2002). DTD används också för HTML där varje version av HTML har en egen DTD i vilken det bl.a. definieras vilka taggar och attribut som får användas för just den versionen.

XML-schema är också en standard för att bestämma strukturen i XML-dokument. XML-schema använder samma syntaxregler som XML (Elmasri & Navathe, 2004). XML-schema har lånat mycket funktionalitet från databasvärlden såsom nycklar, referenser och identifierare (Elmasri & Navathe, 2004).

Med XML-schema skapas ett schemadokument som består av ett antal element i vilka det går att definiera begränsningar, betydelser, användningsområden för elementen men också relationer mellan element och deras innehåll (Statskontoret, 2000b). Något som finns i XML-schema men inte i DTD är möjligheten att skapa datatyper, till exempel element som endast får innehålla nummer eller strängar formaterade efter vissa regler (Statskontoret, 2000b).

2.2.3 XML-Länkar

Redan i HTML finns det sätt att skapa länkar mellan dokument, det vill säga att peka ut andra dokument eller en viss del i ett dokument. Det finns dock problem med hur det fungerar i HTML. Ett dokument kan pekas ut av flera andra dokument. Om sedan det utpekade dokumentet flyttas eller tas bort måste varje länk i de utpekande dokumenten ändras vilket kan bli mödosamt om det rör sig om många olika dokument (Statskontoret, 2000c). Dessutom är målet i en HTML-länk ett helt dokument, även om vissa delar av ett dokument kan pekas ut måste hela dokumentet läsas in trots att endast en viss del är intressant (Statskontoret, 2000c). HTML-länken är också oflexibel på så sätt att den pekar på en ”fysisk” plats i ett dokument.

Med XML-länkning går det att komma ifrån de problem som finns med HTML-länkning (Statskontoret, 2000c). Underhåll av länkar underlättas genom att alla länkar kan samlas i en länkdatabas, länkar behöver inte finnas i det aktuella dokumentet (Statskontoret, 2000c).

XML-länkning tillhandahåller också ny funktionalitet som inte fanns i HTML som bland annat dubbelriktade länkar där båda ändar av länken kan användas både som källa och mål samt att en och samma länk kan innehålla mer än ett mål, det vill säga en länk kan bestå av flera länkar (Statskontoret, 2000c).

XML-länkning är uppdelad i två delar; XLink och XPointer (Statskontoret, 2000c).

XLink används för att länka samman olika dokument. I XML kan en länk vara antingen en enkel länk som leder från en källa till ett mål, eller en utökade länk där ett element kan leda till flera mål.

XPointer används för att adressera olika delar i ett XML-dokument. Istället för att peka på ”fysiska” platser i dokumentet som HTML-länkar gör pekar XML-länkar till platser i informationsstrukturen (Statskontoret, 2000c). Det som kan pekas ut med XML-länkar är information i form av element, attribut eller innehållet i element (Statskontoret, 2000c).

IDREF används för att peka ut ett unikt element i ett XML-dokument (Grose m.fl., 2002). Idref är ett attribut vars värde pekar på ett annat element någonstans i dokumentet. Det andra elementet identifieras genom att det har ett ID-attribut med motsvarande värde (W3C, 2004). Id-attributets värde måste vara unikt för hela dokumentet.

2.2.4 Representation av data i XML

I XML går det att representera samma data på olika sätt. Information kan uttryckas antingen omsluten av ett element eller som ett attributvärde. Det spelar egentligen

ingen roll, informationen är densamma. Nedan visas fyra exempel på hur samma data om en bok av den påhittade författaren ”Lars Larsson” med titeln ”Mina memoarer” kan representeras på olika sätt i XML-format.

```
Exempel 1:
<bok>
     <titel>Mina memoarer</titel>
     <författare>Lars Larsson</författare>
</bok>

Exempel 2:
<bok titel=”Mina memoarer”>
     <författare>Lars Larsson</författare>
</bok>

Exempel 3:
<bok titel=”Mina memoarer” författare=”Lars Larsson”/>

Exempel 4:
<bok>
     <titel värde=”Mina memoarer”/>
     <författare värde=”Lars Larsson”/>
</bok>
```

Det är viktigt att applikationer som utbyter data mellan varandra via XML att båda ”förstår” hur den andra applikationen representerar den data som ska användas. Om dessa representerar samma data på olika sätt gör detta att applikationerna får svårt att förstå varandra (Grose m.fl, 2002). Med hjälp av dokumentmallar som DTD eller XML-schema går det att styra hur informationen i XML-dokument måste vara uppbyggd, det vill säga i vilken ordning element måste komma och vilka attribut de får eller måste innehålla (Grose m.fl., 2002).

Med XML:s länktekniker, XLink och XPointer, behöver inte information som hör till ett element ”fysiskt” finnas hierarkiskt under elementet i ett XML-dokument. En länk kan peka ut informationen någon helt annanstans i samma dokument eller i ett helt annat dokument.

2.2.5 XML i applikationer

För att ett program ska kunna läsa XML krävs att det har en XML-tolk, en så kallad *parser* (Åström, 1999). En parser läser in XML-dokumentet och kontrollerar att denna är skriven på rätt sätt, till exempel att dokumentet följer de grundläggande regler som finns för utformningen av XML-dokument eller regler som har satts upp i DTD eller XML-schema. Ibland måste även XML:en göras om till ett format som programmet förstår så att programmet kan handskas med informationen på det sätt som det behöver (Åström, 1999).

SAX står för Simple API for XML parsers (Azelia, 2002) och är ett API[2] för att hantera XML-dokument. SAX hanterar trädstrukturer genom att det exekverande programmet uppmärksammas när en start- eller sluttagg stöts på (Elmasri & Navathe, 2004).

[2] API står för *Application Programming Interface*, och är en samling av de funktioner som ett program kan utföra.

DOM står för Document Object Model och är ett API som genom ett plattforms- och språkneutralt gränssnitt tillåter programmeringsspråk att dynamiskt komma åt och uppdatera innehållet, strukturen och utseendet av dokument (W3C, 2005).

DOM innehåller funktioner för att manipulera XML-dokumentets trädstruktur på olika sätt, till exempel ta bort, ändra eller flytta (Elmasri & Navathe, 2004). DOM ser alla elementen som objekt som relaterar till varandra som i ett släktträd. Elementet närmast över i hierarkin kallas för förälder (parent), element på samma nivå kallas för syskon (siblings) och element på närmast lägre nivå kallas för barn (child) (Statskontoret, 2000a).

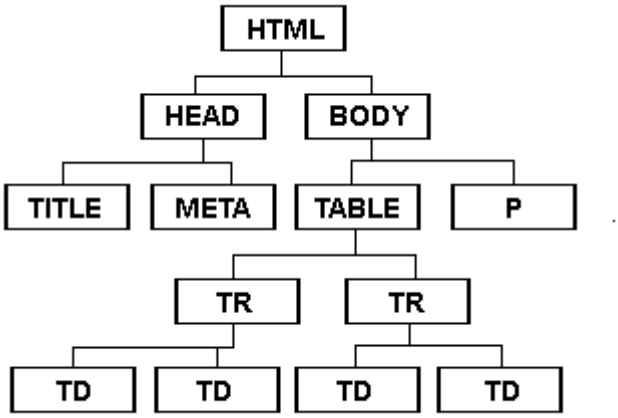


Fig. 2.4. Exempel på ett DOM-träd

Den här (familje)trädstrukturen gör enligt Armstrong (2001) DOM API:et enklare att använda jämfört med SAX. Tyvärr behöver applikationen läsa in hela trädstrukturen från XML-dokumentet i minne, vilket gör att DOM kräver mer datorresurser.

2.3 XMI

När UML först dök upp fanns det ingen standard för att utbyta UML-modeller mellan olika modelleringsverktyg (Stevens, 2003). De flesta verktyg använde sig av egna format som andra verktyg, om det verkligen behövdes, kunde omvandla till ett format som det egna verktyget kunde läsa (Stevens, 2003). Resultatet kunde dock bli opålitligt eftersom det format som översatts inte var utformat för ett sådant syfte (Stevens, 2003).

XMI står för XML Metadata Interchange och är en standard för att utbyta metadata-information om modeller. XMI baseras på tre olika industristandarder; Meta-Object Facility (MOF) och UML från OMG samt XML från W3C (Jiang & Systä, 2002; Statskontoret, 2001).

Fig. 2.5. XMI grundar sig på XML, UML och MOF (Statskontoret, 2001)

MOF är en standard för att representera metamodeller. XMI:s vanligaste användningsområdet är för utbyte av UML-modeller men XMI kan också användas för alla metamodeller som är baserade på MOF (Jiang & Systä, 2002; Stevens, 2003). UML:s

metamodell använder MOF som metamodell och därför går det att använda XMI för att utbyta UML (OMG, 2003a).

XML-delen av XMI består av två delar; en DTD och ett XML-dokument (Jiang och Systä, 2002). XML-dokumentet är uppbyggt av taggar som innehåller information om den sparade modellen. DTD:n i sin tur förklarar vilka taggar som får användas och på vilket sätt (Jiang & Systä, 2002). DTD:n är anpassad för att följa den MOF-metamodell som används. XML-dokumentet inleds med en så kallad ”XMI-header” där det bland annat kan finnas information om vilken XMI-version som används, vilken metamodell som version av metamodellen som används (till exempel UML version 1.3) samt vilket verktyg som använts för att exportera modellen.

Även små modeller kan, när de översätts, bli stora XMI-filer (Stevens, 2003). Enligt Stevens (2003) är den stora fördelen med att XMI är baserad på XML, att informationen i dokumenten kan hanteras och manipuleras genom lättillgängliga XML-tekniker. Exempelvis finns XML-parsers tillgängliga för en rad olika programmeringsspråk.

2.3.1 Versioner

OMG presenterade version 1.0 av XMI, den första officiella versionen, 1999 (OMG, 2000a). Sedan dess har standarden kommit i ytterligare tre officiella versioner; 1.1, 1.2 och 2.0 (OMG, 2000b, 2002, 2003b). De nuvarande officiella versionerna är 1.2 och 2.0. Den första versionen 1.0 är relativt styrd och oflexibel jämfört med version 2.0 som är mer abstrakt och anpassningsbar (Persson, 2004). Version 1.2 är mer flexibel än 1.0, men den senaste versionen 2.0 är den hittills mest flexibla (Persson, 2004). I och med version 2.0 kan även XML-schema användas som dokumentmall för XMI (OMG, 2003b).

OMG har specificerat olika standardiserade DTD:er för olika versioner av UML till varje version av XMI (Persson, 2004). För UML version 1.0 finns det en DTD för XMI version 1.0, en för 1.1, 1.2 och 2.0, samma sak för UML version 1.1 och så vidare. Detta har gjorts för att underlätta utbyte av UML-modeller mellan olika modelleringsverktyg (Persson, 2004).

Tillverkare av modelleringsprogram väljer ibland att bygga ut sin version av XMI för att den ska kunna tillhandahålla en viss funktionalitet som saknas och som tillverkaren eftersträvar (Jiang & Systä, 2002).

För modelleringsverktyg som stödjer olika versioner av XMI krävs att modellerna konverteras till den version som verktyget stödjer för att det ska kunna använda modellen (Persson, 2004). Olika versioner av XMI är alltså inte kompatibla med varandra.

3 Problem

I detta kapitel kommer en övergripande beskrivning av det problem den här undersökningen behandlar. I 3.1 beskrivs det aktuella problemområdet. 3.2 innehåller en kortfattad problemprecisering över den problemställning rapporten grundar sig på. Kapitlet avslutas med 3.3 där de avgränsningar som har gjorts för undersökningen förklaras.

3.1 Problemområde

Som företag eller som enskild utvecklare är det önskvärt att kunna välja den utvecklingsmiljö som av olika orsaker passar bäst för ett utvecklingsprojekt. Helst är inte utvecklarna låsta till att endast använda ett modelleringsverktyg (Persson, 2004; Jiang & Systä, 2002). Licenskostnader, olika verktygs lämplighet för olika typer av projektet eller utvecklarnas kunskaper och erfarenhet av ett visst verktyg kan vara faktorer som spelar in för vilken miljö som företaget eller utvecklaren önskar använda. Jobbar dessutom utvecklare inom samma utvecklingsgrupp på olika platser och/eller har olika plattformar är det tänkbart att det inte finns möjlighet för alla att ha samma utvecklingsmiljö. För att de olika projektdeltagarna ska kunna samarbeta och kunna kommunicera kring modeller över det tänkta systemet måste deras verktyg kunna öppna och hantera de modeller som används. Problem uppstår om modellerna inte kan hanteras av alla deltagare på grund av att deras olika verktyg inte kan tolka dem.

Som tidigare nämnts skapades XMI som en industristandard för utbyte av bland annat UML-modeller mellan olika verktyg. Avsikten med en standardisering är bland annat att säkerställa utbytbarhet och kompabilitet (Ollner, 2005). En modell sparad i ett standardiserat format ska således fungera i alla modelleringsverktyg som stödjer den aktuella standarden. Persson (2004) och Persson, Gustavsson, Lings, Lundell, Mattson och Ärlig (2005b) visar i sina undersökningar på katastrofala resultat vid utbyte av modeller sparade i XMI-format mellan olika modelleringsverktyg med öppen källkod[3]. Varje verktyg som undersöktes fick exportera modeller till XMI och därefter importerades modellen till de andra verktygen. Undersökningen visar att överföringen endast i enstaka fall fungerar korrekt. Allt som oftast överförs modeller felaktigt eller inte alls eftersom de olika verktygen inte förstår varandras XMI-format.

Det finns enligt Persson (2004) olika orsaker till varför XMI i dagsläget har problem som överförings format:

- Genom att tillverkare modifierar XMI-standarden efter sina egna konstruktioner blir XMI-dokument från olika verktyg inkompatibla (Persson, 2002).
- XMI specifikationen är komplex och svårförståelig och den senaste versionen av XMI, version 2.0, är ännu mer komplex än tidigare versioner (Persson, 2004). Om standarden tolkas på olika sätt kan det leda till att de olika verktygens implementationer inte är kompatibla.
- Verktyg som stödjer olika versioner av XMI är inte kompatibla (Persson, 2004). Det krävs en tolk som översätter för att verktygen ska förstå varandras modeller.

[3] Program med öppen källkod ger användaren tillgång till källkoden och ofta tillåtelse för användaren att själv ändra och distribuera källkoden och programmet (Persson, 2004).

Förutom de problem som Persson (2004) nämner finns det faktum, som nämnts tidigare, att det i XML går att representera samma information på olika sätt. En tillverkare av ett modelleringsprogram kanske väljer att representera viss information som underliggande element när en annan tillverkare valt att representera den som attribut. När sedan modeller överförs riskerar de att tolkas felaktigt.

XMI standarden stödjer olika sätt att länka eller referera till olika platser i XMI-dokumentet eller till helt andra XMI-filer (Grose m.fl., 2002). XML-attributen xmi.id, xmi.label och xmi.uuid används för att unikt kunna identifiera element (OMG, 2000a). Tanken med dessa attribut är att tillåta XML-element att sammankopplas till andra XML-element genom XML:s länktekniker IDREF, XLink och XPointer (OMG, 2000a). Detta försvårar ytterligare förståelsen mellan applikationer som utbyter information via XML eftersom information som är relevant för ett visst element kan ”gömma” sig på flera olika ställen i ett XML-dokument. Det här är ett problemområde i vilket det pågår mycket forskning och där problemet ännu inte är löst (Christophides, Plexousakis, Scholl, & Tourtounis, 2003; Milo & Suciu 1999; Kaushik, Bohannon, Naughton & Korth, 2002; Li & Moon, 2001).

XMI-dokument som exporterats från modelleringsverktyg uppfyller inte alltid de krav som OMG:s dokumentmallar för XMI ställer. Vid tester med oberoende valideringsverktyg har det visat sig att strukturen på filer kan skilja sig från gällande XMI-standard (Persson, Gustavsson, Lings, Lundell, Mattson & Ärlig, 2005a). En översättare av XMI-dokumentet riskerar att misslyckas med att tolka informationen om strukturen på dokumentet avviker från hur strukturen egentligen, enligt dokumentmallen, ska se ut. Detta gör att dokumentmallen för XMI inte längre ger någon tolkningsfördel vid import av en modell eftersom de regler som dokumentmallen ställt upp ändå inte följs.

Oavsett anledning till varför ett XMI-dokument inte går att importera till ett verktyg så innehåller dokumentet information om modellen. Information som kan vara väldigt betydelsefull. Ponera att ett företag har använt olika UML-diagram för att på olika sätt åskådliggöra och dokumentera ett mycket omfattande informationssystem. Det modelleringsprogram som diagrammen ursprungligen gjordes i finns av olika anledningar inte längre på företaget – giltiga licenser saknas, programmet tillverkas inte längre (går inte att får tag på) eller programmet finns inte för den plattform som företaget idag använder. Ansvariga för dokumenteringen har dock varit förutseende och sparat modellerna i XMI-format för att göra det möjligt att använda dem vid ett senare tillfälle i ett annat verktyg. När den dagen kommer och modellerna behövs igen fungerar inte importen av XMI-dokumenten till det nya verktyget! Tusentals mantimmar har använts för att skapa de modeller som nu inte går att använda. Den information som finns i modellerna är nödvändig för det jobb som nu har planerats. Att återskapa modellerna genom att gå igenom systemets källkod eller på andra sätt samla in information om systemet skulle kräva mycket tid och på så sätt bli kostsamt.

Lösningen på ovanstående problem skulle vara ett program som oavsett hur information har representerats (element i element, attribut i element eller XML-länkar) och utan hjälp från DTD eller XML-schema klarar av att utvinna informationen från XMI-dokument.

3.2 Problemprecisering

Problemet är formulerat som en hypotes, vilken lyder:

Det är möjligt att med en enkel och flexibel extraktor utvinna och lista relevant information om ett UML-diagram som har sparats i XMI-format, utan hjälp från DTD eller XML-schema.

Med *relevant information* avses på förhand åsyftad information om det UML-diagram som finns sparat i XMI-format.

Anledningen till varför DTD eller XML-schema inte ska behövas för extraktionen är för att XMI-dokument inte alltid följer XMI-standardens dokumentmallar (Persson m.fl., 2005a). En dokumentmall skulle i de fall då den inte följs, inte ge någon tolkningsfördel, eftersom strukturen på XMI-dokumentet skiljer sig från den struktur som avses i dokumentmallen. Extraktorn skulle dessutom gå att använda som ”livboj” för att extrahera information om modellen i fall då dokumentmallen för XMI-dokumentet någon anledning saknas.

Med *flexibel* avses att extraktorn ska gå att anpassa för att stödja olika XMI- och UML-versioner samt eventuella felaktigheter i XMI-användningen i det dokument som används.

Målet med hypotesen är att skapa en extraktor som klarar att extrahera information om modeller utan dokumentmall, vilket skulle kunna fungera som en livboj för att rädda information om sparade modeller i de fall XMI-dokumenten är felaktiga eller dokumentmallen saknas.

För att hypotesprövningen ska lyckas, det vill säga för att hypotesen ska verifieras, krävs att, på förhand åsyftad information om det UML-diagram som finns sparat i de XMI-dokument som används för hypotesprövningen, går att extrahera.

Hypotesen kräver att ett program för att hämta ut informationen ur XMI-dokument implementeras, en så kallad *extraktor*. För att enklare kunna uppnå slutmålet har problemet delats upp i fyra olika delmål. Dessa är:

1. Rangordna de element som används i klassdiagram och avgöra i vilken ordning informationen i dessa ska hämtas. De element som är lättast att identifiera ska hämtas först. Eventuellt spelar även beroenden, element emellan, in i avgörandet av vilken ordning som är lämpligast.
2. Välja en lämplig implementationsplattform.
3. Avgöra hur extraktorn kan göras konfigurerbar för att stödja olika versioner av XMI, både nuvarande och kommande. Skapa en idé om hur detta ska göras så generellt som möjligt.
4. Implementera och testa extraktorn.

3.3 Avgränsningar

- Implementationen kommer att begränsas till att klara av att extrahera information från UML:s klassdiagram. Klassdiagram är en populär och ofta förekommande diagramtyp (Fowler, 2000). Informationsextraktion från denna diagramtyp antas därför behövas i fler fall jämfört med andra diagramtyper.
- Undersökningen har inte för avsikt att utvinna all information om klassdiagrammen från XMI-dokumenten. Endast utvald, och för klassdiagram

karaktäristisk, information kommer att extraheras. Denna avgränsning görs främst på grund av rapportens omfattning och givna tidsramar. Det finns även en möjlighet att olika modelleringsverktyg värderar olika typer av information om klassdiagram på olika sätt. Exempelvis finns det en risk att olika verktyg som använder samma modell inte sparar samma information om modellen. Genom att endast utvinna karaktäristisk information om klassdiagram, som därför antas finnas i alla XMI-dokument, minimeras risken att extraktorn fungerar olika beroende på vilket modelleringsverktyg som har genererat ett XMI-dokument.

4 Metod

I detta kapitel beskrivs den metod som har valts för undersökningen utifrån de delmål som har preciserats i föregående kapitel.

4.1 Metodval

Den kvalitativa undersökningen kommer att genomföras i form av ett praktiskt projekt där hypotesprövning, litteraturstudie, experiment och implementation kommer att användas. Att genomföra undersökningen som ett praktiskt projekt är användbart när syftet, som i det här fallet, är att genomföra experiment och implementationer för att påvisa principer och koncept samt att från dessa samla erfarenheter (Berndtsson, Hansson, Olsson & Lundell, 2002).

4.1.1 Hypotesprövning

Problemet har formulerats som en hypotes. Därför kommer strävan med undersökningen att vara att validera hypotesen, det vill säga att i det här fallet bevisa att antagandet stämmer. Målet med prövningen är med andra ord att verifiera hypotesen. Resultatet kan dock innebära att hypotesen måste förkastas, till exempel om det visar sig att implementationen inte kan extrahera eller lista den önskade informationen.

4.1.2 Litteraturstudie

Det är viktigt att ett praktiskt projekt grundar sig på relaterad teori för att bättre belysa den teoretiska innebörden av problemet (Berndtsson m.fl., 2002). Därför kommer en litteraturstudie att göras. Litteraturstudien är betydelsefull vid hypotesprövningen, eftersom en sådan prövning med fördel genomförs när det finns tillräcklig kunskap inom området (Patel & Davidson, 2003). Målet med litteraturstudien är att få grundläggande kunskaper och förståelse för problemområdet, men också att få idéer om hur problemet kan lösas.

Den litteratur som bedöms som allra mest relevant är främst de specifikationer från OMG som finns för de olika versionerna av XMI. XMI-standarden är det mest väsentliga för undersökningen och det är därför viktigt att få en god insikt i hur standarden är tänkt att tillämpas och vad som skiljer de olika versionerna åt.

Litteraturen om XMI kan tillsammans med litteratur om UML även ge stöd för det första delmålet; problemet med i vilken ordning de olika XMI-objekten ska identifieras. Den här litteraturen kan eventuellt ge vägledning om huruvida UML:s notation eller strukturen på XMI-dokument har någon inverkan på i vilken ordning objekten bör extraheras. Exempelvis kanske objekten relaterar till varandra på ett sätt som gör att objekten måste extraheras i en viss ordning.

Även för det andra delmålet, det vill säga valet av lämplig implementationsplattform, kan antagligen litteratur kring mjukvaruplattformar, utvecklingsverktyg och liknande implementationer visa sig ge stöd.

4.1.3 Experiment och implementation

Experiment kan användas för att verifiera eller falsifiera en hypotes. Inom datavetenskapen är det ofta vanligt med implementationer för att genomföra experiment (Berndtsson m.fl., 2002). En implementation bör ha ett tydligt syfte som exempelvis att bevisa att något går att utföra (Berndtsson m.fl., 2002). Av dessa anledningar har

en implementation valts att användas för att pröva hypotesen i denna undersökning. Erfarenheter från experiment kommer att ligga till grund för implementationen. Syftet med experimenten och implementationen är att visa att problemet går att lösa och därmed verifiera hypotesen.

Genom att praktiskt undersöka i vilken ordning det är möjligt eller enklast att extrahera de olika objekten kan erfarenheter från experimenten underlätta lösningen av det första delmålet.

En utvärdering och analys av de resultat som framkommer från experimenten kommer att behövas för att undersöka delmål tre; att göra extraktorn konfigurerbar för att stödja olika versioner av XMI. Experimenten kan ge idéer om vilka funktioner som kan generaliseras och hur konfigurering av extraktorn ska göras på bästa sätt.

Implementationen kan också visa sig ge fördelar för presentationen av och förståelsen för undersökningens resultat. Enligt Dawson (2000) är en av fördelarna med en implementation att det enklare går att presentera de resultat som har uppnåtts, till skillnad från att visa resultaten med pseudokod eller genom en teoretisk beskrivning av hur programmet ska fungera. En implementation hjälper dessutom till att framhäva och tydliggöra de egenskaper eller det beteende som lösningen har (Berndtsson m.fl., 2002).

5 Genomförande

I detta kapitel beskrivs undersökningen utifrån hur den kronologiskt har genomförts samt hur de olika delmålen har angripits.

5.1 Litteratur- och teoristudier

Undersökningen inleddes av litteraturstudier för att ge en grundläggande förståelse för problemet och problemområdet. Till att börja med söktes litteratur om de bakomliggande teknikerna UML, XML och XMI. Den informationen hittades främst i böcker på bibliotek, men även i publicerade rapporter på Internet. Den inledande litteratursökningen genererade mycket material om teknikerna UML och XML men något mindre information om XMI. För att komplettera detta gjordes även sökningar i tidskrifts- och forskningsdatabaser, där ytterligare information om XMI hittades. Även ännu ej publicerat material där problem vid utbyte av modeller mellan modelleringsverktyg med hjälp av XMI påträffades.

XML, UML och XMI:s respektive standardiseringsorgan visade sig också ha publicerat sina standarders specifikationer på sina hemsidor, vilket gjorde dem lättåtkomliga.

Den inledande litteraturstudien användes som stöd för det första delmålet; rangordning av UML-elementen. Studien bidrog till förståelse för hur UML-objekten ska användas och hur de inbördes relaterar till varandra samt hur dessa sedan representeras i XMI. Specifikationer för de olika versionerna av XMI kom att användas i stor utsträckning under detta delmål.

För delmål två söktes och studerades litteratur om olika mjukvaruplattformar och hur dessa plattformar kan användas för att hantera XML. Dessa källor hittades både i tryckt form och på Internet.

Referenslitteratur kring den valda plattformen användes för att söka svar på specifika frågor och problem som uppkom under utvecklingen av extraktorn, det vill säga det sista delmålet. Även olika API-specifikationer för den valda plattformen och specifikationer för de olika versionerna av XMI användes mycket under detta delmål.

5.2 Experiment

Som grund för experimenten användes XMI-dokument som exporterats från olika modelleringsverktyg. Dessa XMI-dokument har använts i två tidigare undersökningar där problem kring utbytandet av modeller mellan modelleringsverktyg har undersökts (Persson, 2004; Persson m.fl., 2005b). XMI-dokumenten baseras på en och samma UML-modell som har skapats i olika modelleringsverktyg, varefter de ifrån verktygen har exporterats till XMI. De olika XMI-dokumenten innehåller alltså samma information även om de olika verktygen använder sig av sitt eget sätt att representera informationen i XMI.

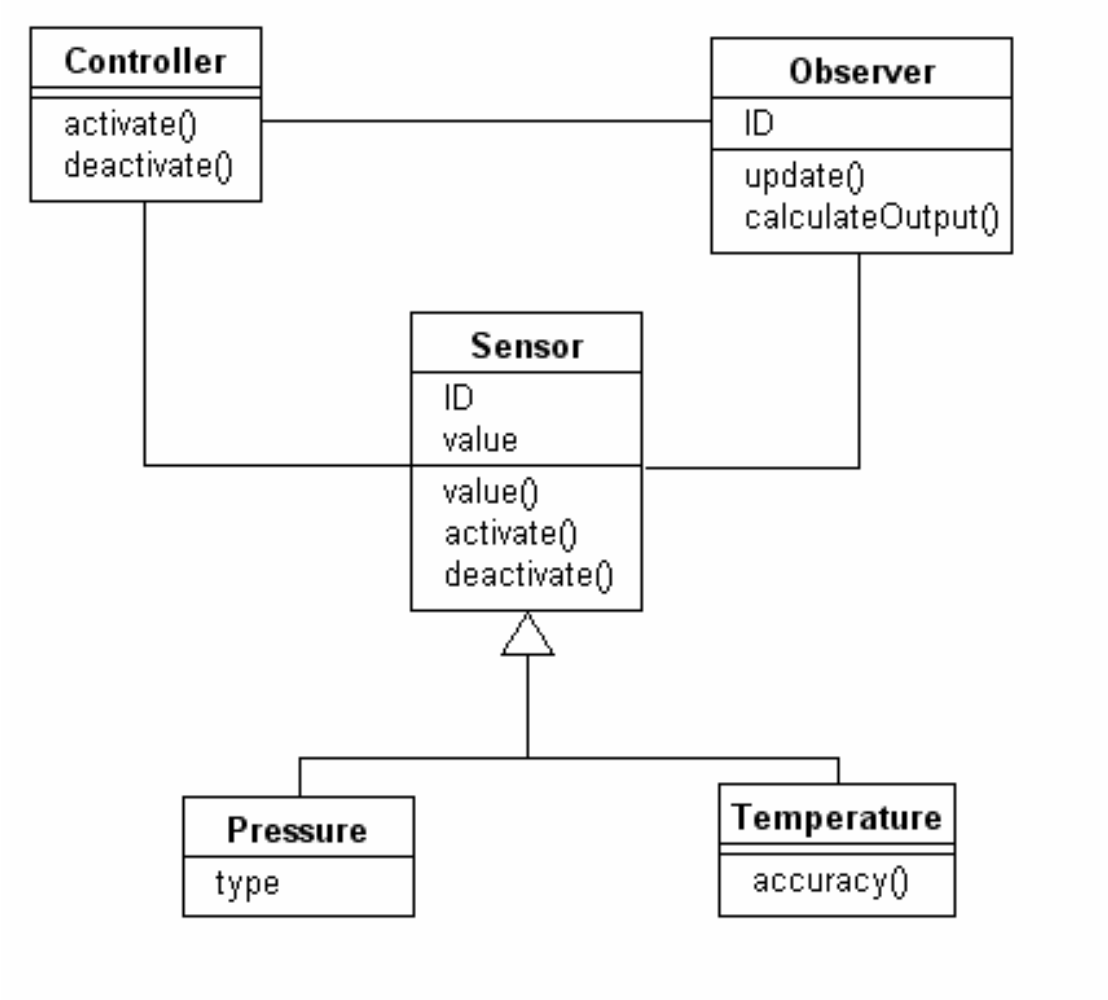


Fig. 5.1. Den UML-modell som har exporterats till XMI.

Experimentet började med att XMI-dokumenten delades in i två olika grupper. Orsaken till detta är att dokumenten följer olika versioner av XMI, vilket gör att de skiljer sig mycket från varandra. Den ena gruppen innehöll XMI-dokument exporterade till XMI version 1.0 eller 1.1 (*experimentgrupp 1*) och den andra gruppen innehöll XMI-dokument exporterade till version 1.2 eller 2.0 (*experimentgrupp 2*).

De verktyg som har använts för att exportera modellerna samt vilken version av XMI och UML som använts är:

Experimentgrupp 1			
Verktyg	**XMI-version**	**UML-version**	**Filnamn**
Agro	1.0	1.3	ArgoXMI.xmi
Rhapsody	1.0	1.3	RhapsodyXMI.xmi
Rose	1.0	1.3	RoseXMI10.xmi
Visio	1.0	1.1	Visio.xmi

Tabell 5.1. Experimentgrupp 1

Experimentgrupp 2			
Verktyg	**XMI-version**	**UML-version**	**Filnamn**
Artisan	1.3	1.3 eller 1.4	Artisan.xmi
Fujaba	Uppgift saknas[4]	1.2 eller 1.3	FujabaXMI.xmi
Poseidon	1.2	Uppgift saknas	PoseidonXMI.xmi
Rose	1.1[5]	1.3	RoseXMI11.xmi

Tabell 5.2. Experimentgrupp 2

Till att börja med gicks alla XMI-dokumenten igenom manuellt för att avgöra i vilken ordning de olika UML-elementen skulle urskiljas. Även de övriga experimenten hade kunnat genomföras manuellt genom att steg för steg gå igenom varje XMI-dokument för att analysera hur information relaterar till varandra. Problemet är att XMI-dokument ofta är stora, komplext uppbyggda och innehåller mycket information vilket kan bli övermäktigt för det mänskliga ögat att hantera (Persson m.fl., 2005a). Risken hade varit stor att experimenten inte utförts korrekt, till exempel att information förbisetts. Viktigt var också att experimenten alltid genomfördes metodiskt för varje fil så att alla filer lästes på samma sätt varje gång. Detta hade varit svårt att göra manuellt. Därför genomfördes huvuddelen av experimenten genom att mindre program skapades som skötte informationsutvinningen.

I undersökningens alla experiment med testprogram används så kallade *NodeFilter* (Apache XML Project, 2004) för att filtrera bort element som inte uppfyller uppsatta kriterier. Genom att använda ett anpassat NodeFilter går det att via *Xerces* välja ut alla element i ett XML-träd som uppfyller vissa kriterier. Xerces är den XML-parser som valts för undersökningen. Ett exempel på användning av NodeFilter kan vara att om vi endast är intresserade av element vars taggnamn slutar på ”Class” går det att skriva följande kod i ett NodeFilter:

```
Exempel i Javakod:
//kontrollerar om noden är ett element
if (node.getNodeType() == Node.ELEMENT_NODE)
{
     //kontrollerar att nodenamnet slutar på ”Class”
     if (node.getNodeName().endsWith("Class"))
     {
           //filtret accepterar elementet
           return NodeFilter.FILTER_ACCEPT;
     }
}
//filtrerar bort noden om den inte accepterats ovan
return NodeFilter.FILTER_REJECT;
```

På liknande sätt går det att skapa olika filter för att passa kriterier för olika typer av XMI-objekt och specifik information som dessa innehåller.

[4] XMI-version anges inte, dock liknar strukturen på dokumentet de dokument som är av version 1.2 eller 2.0. Versionen antas alltså vara av version 1.2 eller 2.0.

[5] XMI-dokumentet uppges vara av XMI-version 1.1. Strukturen liknar dock inte de övriga dokumenten i experimentgrupp 1 utan mer de i experimentgrupp 2. Versionen antas därför vara 1.2 eller 2.0, inte 1.1 som anges.

I experimentfasen skapades olika testprogram för varje XMI-objekt, det vill säga klasser, associationer och arv. Strukturen programmen emellan är snarlik. Det som skiljer dem åt är att de använder olika typer av NodeFilter, vilka är anpassade för att hitta olika typer av information om den aktuella objekttypen. Samma program användes för att hitta samma typ av objekt i båda experimentgrupperna, men med skillnaden att kriterierna för de filter som användes ändrades. Fyra konstruktioner är gemensamma för alla testprogram:

- **Extractor:** Den del av programmet som sköter öppnandet av olika XMI-dokument och presenterar informationen på datorskärmen.
- **XMIDocument:** Den del av programmet som läser in XMI-dokumenten och sorterar ut eftersökta XMI-objekt.
- **XMIObject:** Den del av programmet som representerar ett eftersökt XMI-objekt; klass, association eller arv. Ett XMIObject hanterar den information som är specifik för just den typen av objekt, exempelvis attribut och operationer för klasser.
- **NodeFilter:** Ett NodeFilter filtrerar bort element som inte uppfyller vissa kriterier. Olika typer av NodeFilter filtrerar efter olika kriterier. NodeFilter används av XMIDocument och XMI-objekt.

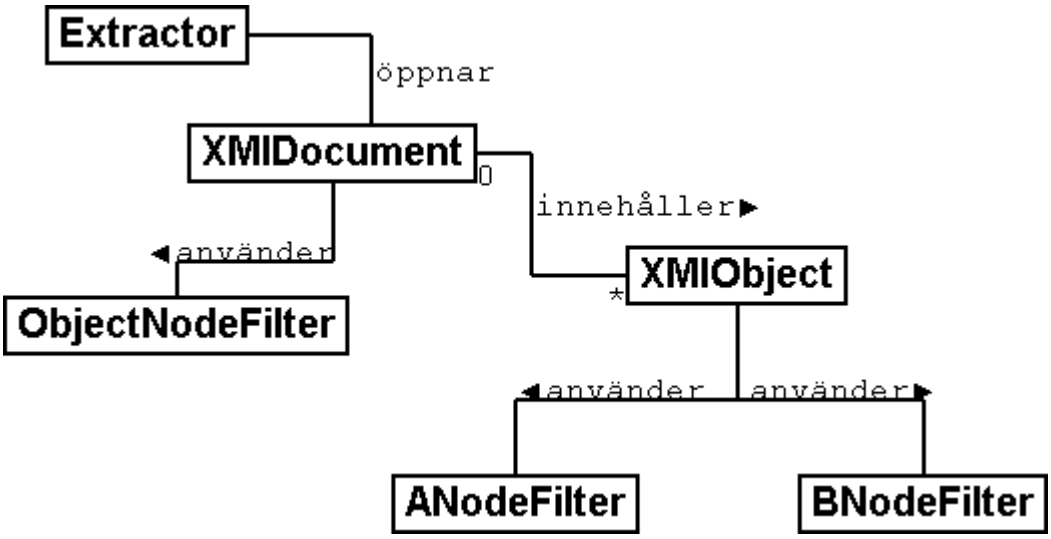


Fig. 5.2. Exempeldiagram på strukturen för ett testprogram.

Experimenten med testprogrammen genomfördes genom att först välja ut en typ av XMI-objekt från vilken information skulle hämtas. Därefter gjordes tester i två steg för varje XMI-objekt:

1. Urskilja alla förekomster av den utvalda objekttypen i de olika XMI-dokumenten.
2. Hitta specifik information i de urskiljda objekten, till exempel namn, vilka attribut och operationer som en klass innehåller.

Först genomfördes tester i experimentgrupp 1. De olika typerna av XMI-objekt gicks igenom en efter en. När all information om en vald objekttyp hade hittats i alla dokument och informationen visade sig vara korrekt gjordes samma test för nästa objekttyp. När alla typer i experimentgruppen testats upprepades samma procedur i nästa experimentgrupp. På så sätt identifierades de skillnader som finns mellan XMI-

dokumenten inom samma experimentgrupp och hur programmet måste fungera för att hitta informationen i samtliga filer i gruppen. Dessutom förvärvades erfarenheter om skillnader mellan de olika experimentgrupperna. Dessa erfarenheter visade sig vara betydelsefulla vid avgörandet av hur implementationen skulle göras konfigurerbar.

Alla experiment har genomförts i syfte att ligga till grund för delmål 3, vars avsikt är att avgöra hur extraktorn kan göras konfigurerbar för att stödja olika versioner av XMI. Dessutom underlättar experimenten till att i förlängningen avgöra den slutgiltiga designen av implementationen, det vill säga delmål 4.

5.3 Implementation

Erfarenheter från de genomförda experimenten användes som underlag för en systemanalys. En systemanalys är ett moment då förutsättningarna för ett nytt system undersöks grundligt (Avison & Shah, 1997). Resultatet av denna analys låg sedan till grund för de krav som ställs i delmål 4, det vill säga designen av implementationen.

Resultatet av systemanalysen användes för att skapa en strukturerad design över implementationen. Designen visualiserades genom att skapa ett klassdiagram i UML. Diagrammet beskriver kraven på funktionen för de olika delarna av programmet och hur dessa delar relaterar till varandra.

Med systemdesignen som bakgrund skapades implementationen, det vill säga det program som är det huvudsakliga resultatet av denna undersökning. Den ursprungliga designen följdes i huvudsak men vissa avsteg gjordes i situationer där designen inte visade sig fungera.

5.4 Hypotesprövning

När väl implementationen var färdig och resultatet gick att utvärdera genomfördes hypotesprövningen. För att verifiera hypotesen krävdes att implementationen skulle kunna extrahera all åsyftad information från alla XMI-dokument som använts i undersökningen. Detta skulle visserligen inte innebära att implementationen skulle kunna extrahera information även från XMI-dokument som inte använts i undersökningen, men hypotesen skulle dock verifieras inom ramen för denna undersökning.

Samtliga XMI-dokument som använts i undersökningen grundar sig på samma modell (se figur 5.1) och ska därför innehålla samma information. För att hypotesprövningen skulle verifieras var samma information tvungen hittas i samtliga XMI-dokument.

6 Resultat och analys

I detta kapitel presenteras undersökningens resultat för varje enskilt delmål. Kapitlet avslutas med en slutsats.

6.1 Delmål 1 - Rangordning av UML-element

Litteraturen om UML och XMI samt analyserna av XMI-dokumenten gjorde klart att klasser i det närmaste är självskrivna som första objekt att identifiera ur XMI-dokument. De viktigaste byggstenarna i klassdiagram är klasserna med sina attribut och operationer, association och arv. Den modell som undersökningen använde sig av innehöll också endast nämna byggstenar (*se figur 5.1*). Utan klasser blir klassdiagrammen tämligen meningslösa. Genom att först identifiera klasser blir det möjligt att sedan fortsätta med att identifiera ytterligare objekt som är sammankopplade med dessa klasser. Associationer och arv har mycket gemensamt. Båda är relationer mellan två eller fler klasser. Dessutom representeras de på ett liknande sätt i XMI. Efter att ha analyserat vilket av dessa element som skulle vara enklast att extrahera togs beslutet att associationer är den enklare av de båda.

Den slutgiltiga rangordningen av UML-element blev alltså:

1. Klasser
2. Associationer
3. Arvsrelationer

6.2 Delmål 2 - Val av lämplig implementationsplattform

Objektorienterade språk som Java och C++ och UML delar det objektorienterade tankesättet (Skansholm, 2000). UML används för att modellera objektorienterade system, system som sedan kan förverkligas i program skapade i ett objektorienterat språk. Många av de konstruktioner som används i UML, som exempelvis klassers attribut och operationer, har en direkt motsvarighet i objektorienterade språk. Dessa har därför logiskt mycket gemensamt.

Andra programmeringsspråk, som exempelvis Perl, är bra på att hantera reguljära uttryck, det vill säga att med specialkommandon leta efter mönster i en textmassa. Att använda reguljära uttryck visade sig mycket användbart för experimenten eftersom mycket av problematiken handlar om att hitta mönster i en informationsmängd. Stöd för reguljära uttryck finns även i ett flertal andra programmeringsspråk som Java, C och C++.

En egen inläsningsfunktion för XML går att skapa från grunden men då finns risk att buggar oupptäckt smyger sig in i programmet. En redan befintlig parser, där inläsningsfunktion är en del, är sedan tidigare testad och använd och innehåller förmodligen färre ”buggar”. Enligt Berndtsson m.fl. (2002) skulle buggar i programmet inkräkta på undersökningens validitet, eftersom de kan påverka resultatet. En parser använder sig dessutom ofta av ett standardiserat parser-API som till exempel DOM och SAX, vilka också stöds av de flesta utvecklingsplattformar (Homer, Sussman & Fussell, 2004).

Den plattform som slutligen valdes var Java med en XML-parser; *Xerces2 Java Parser 2.6.2*. De främsta orsakerna till varför Java valdes var att:

- Program som utvecklas i Java är plattformsoberoende. Med plattformsoberoende menas att samma program går att köra på flera olika datorer oavsett vilket operativsystem och processor som används. Detta gör att implementationen kan användas på olika plattformar utan speciell anpassning.
- Java har ett väl utvecklat API för XML.
- Java och XML passar enligt Azelia (2002) mycket bra ihop rent tekniskt. Båda fokuserar på flyttbarhet, där Java tillhandahåller flyttbara applikationer och XML ger flyttbar data.
- Java har stöd för reguljära uttryck.

I stort sett alla programmeringsspråk som har möjlighet att läsa in och behandla textfiler hade kunna användas för att hantera XMI-dokumenten på olika sätt. Valet av implementationsplattform hade därför kunnat ske förhållandevis godtyckligt. Litteraturstudien gav det underlag för resonemanget som ledde fram till det slutgiltiga valet av Java som implementationsplattform.

Huruvida en parser skulle användas för att hantera XML-information i implementationen var också ett ställningstagande. En XML-parser valdes för att hantera XMI-dokumenten, för att inte eventuella buggar i hanteringen av XML ska inkräkta på undersökningens validitet. Den XML-parser som valdes var *Xerces2 Java Parser 2.6.2*. Orsaken till varför *Xerces2* valdes var framförallt för att den stödjer Javas API för XML-hantering. Eftersom Xerces2 är skapad i Java är också den plattformsoberoende och kan därför används tillsammans med implementationen på olika plattformar. Parserns funktioner och egenskaper är väl dokumenterade och finns lätt tillgängliga via webbsidor på Internet (The Apache Software Foundation, 2004). Dessutom kostar den ingenting att använda (The Apache Software Foundation, 1999).

6.3 Delmål 3 – Konfigurationsmöjligheter

Vid den inledande granskningen av undersökningsmaterialet, det vill säga XMI-dokumenten, undersöktes vilka uppgifter som fanns angående XMI- och UML-version. Detta gjordes för att eventuellt kunna använda denna information vid extraheringen. För filerna i experimentgrupp 1 visade sig alla versionsuppgifter finnas med och vara korrekta (*se tabell 5.1*). I experimentgrupp 2 däremot är versionsuppgifterna för både XMI och UML inkonsekventa (*se tabell 5.2*), vilket gör det omöjligt att med hjälp från dessa uppgifter avgöra hur informationen ska extraheras.

De felaktigheter som upptäckts i dokumenten i experimentgrupp 2 var:

- Artisan refererar i XML-namespacelänken till UML 1.4 men i metamodellelementet till UML version 1.3.
- I Poseidon saknas uppgift om vilken metamodell (vilken version av UML) som används.
- Fujaba uppger i attributet för XML-namespace att UML 1.3 används men UML 1.2 i metamodelltaggen. Eventuellt uppges metamodellen felaktigt som version 1.2 när avsikten egentligen var att uppge XMI-versionen. Detta antagande bygger på att dokumentet saknar uppgift om XMI-version och att attributet som används för att uppge metamodellversion heter ”xmi.version”.

- Rose uppger XMI version 1.1 men förmodligen är det är version 1.2 eller 2.0 eftersom strukturen på dokumentet är mycket lik de övriga i experimentgrupp 2.

På grund av att versionsuppgifterna för både XMI och UML är felaktiga eller saknas för samtliga filer i experimentgrupp 2 har experimenten inte kunnat använda dessa uppgifter i något avseende.

Efter granskningen av undersökningsmaterialet var avklarad vidtog experimentfasen. Experimenten är indelade klass-, associations- och arvsexperiment. För att genomföra dessa experiment skapades allra först testprogram, som skulle underlätta extraheringen av information från XMI-dokumenten.

De testprogram som togs fram för att extrahera information från de olika XMI-dokumenten använde sig av olika NodeFilter. Samma filterkriterier gick ofta att använda i båda experimentgrupperna, men ibland behövde de ändras. Däremot behövde programmen anpassas mellan de olika XMI-dokumenten för att hitta informationen. I dokumenten i experimentgrupp 1 hittades ofta informationen som textnoder någonstans i hierarkin under det eftersökta XMI-objektet. I dokumenten i experimentgrupp 2 hittades samma information ofta i attribut i XMI-objektets startelement.

6.3.1 Klassexperimenten

Klassexperimenten använde sig av tre olika NodeFilter; *ClassNodeFilter*, *AttributeNodeFilter* och *OperationNodeFilter*. Det första filtret används för att urskilja klasselement och de övriga två för att hitta klassens attribut respektive operationer. Nedan beskrivs stegvis de problem som uppkom, hur dessa löstes och de vilka skillnader som upptäcktes mellan XMI-dokumenten inom samma experimentgrupp.

Experimentgrupp 1

1. *Hitta klasselementen.* För att sortera bort alla ointressanta element krävdes att filtret endast accepterar element som slutar på ”.Class” och ej innehåller länkattributet ”xmi.idref”. Visiodokumentet innehåller en extra klassdefinition som inte finns med i den ursprungliga UML-modellen och egentligen inte borde vara med. Klassdefinitionen har namnet ”<*unspecified*>” och saknar både attribut och operationer.
2. *Hitta övrig information ur klasselementen*. Den övriga informationen som hämtas är klassnamn, attribut och operationer. Namnet hittas i samtliga XMI-dokument genom att elementnamn som slutar på ”ModelElement.name” söks direkt under klasselementet. Klassens attribut och operationer hittas genom att element som slutar på ”Attribute” respektive ”Operation” eftersöks var som helst i hierarkin under klasselementet.

NodeFilter	**Accepterar**
ClassNodeFilter	Alla element som slutar på ”.Class” och saknar attributet ”xmi.idref”
AttributeNodeFilter	Alla element under ett klasselement som slutar på ”Attribute”
OperationNodeFilter	Alla element under ett klasselement som slutar på ”Operation”

Tabell 6.1. Klasstestets filterinställningar för experimentgrupp 1.

Experimentgrupp 2

1. *Hitta klasselementen.* Genom att ändra filtret till att acceptera element som slutar på ”Class” och saknar länkattribut hittas alla klasselement i gruppens alla XMI-dokument.
2. *Hitta övrig information ur klasselementen*. Klassnamnet hittas för alla dokument som ett attribut med namnet ”name” i klasselementet. Attribut- och operations-elementen hittas på samma som i experimentgrupp 1, genom att söka efter element som slutar på ”Attribute” respektive ”Operation”. Informationen som eftersöks, deras namn, finns för de båda som ett attribut med namnet ”name” i de element som hittats.

NodeFilter	**Accepterar**
ClassNodeFilter	Alla element som slutar på ”Class” och saknar attributet ”xmi.idref”
AttributeNodeFilter	Alla element under ett klasselement som slutar på ”Attribute”
OperationNodeFilter	Alla element under ett klasselement som slutar på ”Operation”

Tabell 6.2. Klasstestets filterinställningar för experimentgrupp 2.

6.3.2 Associationsexperiment

I associationsexperimenten användes två olika NodeFilter; *AssociationNodeFilter* och *AssociationEndsNodeFilter*. Det första används för att urskilja associationselement det andra för att hitta associationens ändpunkter.

Experimentgrupp 1

1. *Hitta associationselementen.* Genom att låta filtret acceptera element som slutar på ”Association” och saknar länkattribut hittas alla associationselement i gruppens alla XMI-dokument.
2. *Hitta övrig information ur associationselementen*. Övrig information för associationer är dess namn och ändpunkter, det vill säga de klasser som de är associerade till. Namnet hittades genom att element som slutar på ”ModelElement.name” söks direkt under associationselementet. Ändarna hittades genom att element som slutade på ”AssociationEnd.type” hämtades. I alla dokument använder associationer länkar för att peka ut de klasser de associerar till. De olika dokumenten använder olika taggar för dessa länkar. Dessa taggar har snarlika namn men är ändå inte samma. Genom att endast söka efter värdet i länk-attributet, oavsett taggnamn, hittas identiteten för klasserna.

NodeFilter	**Accepteras**
AssociationNodeFilter	Alla element som slutar på ”Association” och saknar attributet ”xmi.idref”
AssociatioEndsFilter	Alla element vars förälder slutar på ”type”

Tabell 6.3. Associationstestets filterinställningar för experimentgrupp 1.

Experimentgrupp 2

1. *Hitta associationselementen.* Samma filter används för att hitta associations-elementen i båda experimentgrupperna.
2. *Hitta övrig information ur associationselementen*. Namn hittas i de fall då de används i attributet ”name” i associationselementet. Ändarna hittas i element som slutar på ”AssociationEnd”. Värdet för associationsändarna kan hittas på olika ställen i olika dokument. Ibland som länkar i andra fall som attribut.

NodeFilter	**Element som accepteras**
AssociationNodeFilter	Alla element som slutar på ”Association” och saknar attributet ”xmi.idref”
AssociatioEndsFilter	Alla element som slutar på ”AssociationEnd”.

Tabell 6.4. Associationstestets filterinställningar för experimentgrupp 2.

6.3.3 Arvsexperimenten

I arvsexperimenten användes tre olika NodeFilter; *GeneralizationNodeFilter, ParentNodeFilter* och *ChildNodeFilter*. Det första används för att urskilja arvs-element de andra för att hitta supertypen (förälder) respektive subtypen (barn) för arvet.

Experimentgrupp 1

1. *Hämta arvselementen*. Elementet Arvselementet hittas genom att söka på element som slutar på ”Generalization” och saknar länkattribut.
2. *Hitta övrig information ur arvselementen.* Olika taggnamn används för både förälder- och barnelementen och även för det länkelement som dessa innehåller. De olika taggar för föräldrar och barn som förekommer accepteras i programmet. Länkelement hittas genom att endast värdet i länkattributet hämtas oavsett elementets namn.

NodeFilter	**Element som accepteras**
GeneralizationNodeFilter	Alla element som slutar på ”Generalization” och saknar attributet ”xmi.idref”
ParentNodeFilter	Alla element som slutar på ”parent”
ChildNodeFilter	Alla element som slutar på ”child”

Tabell 6.5. Arvstestets filterinställningar för experimentgrupp 1.

Experimentgrupp 2

1. *Hämta arvselementen*. Elementen i de olika dokumenten hittas genom att söka efter element som slutar på "*Generalization*" och saknar länkattribut.
2. *Hitta övrig information ur arvselementen.* I en del dokument finns informationen om förälder och barn i attribut, i andra i länkelement under arvselementet.

NodeFilter	**Element som accepteras**
GeneralizationNodeFilter	Alla element som slutar på ”Generalization” och saknar attributet ”xmi.idref”
ParentNodeFilter	Alla element som slutar på ”parent”
ChildNodeFilter	Alla element som slutar på ”child”

Tabell 6.6. Arvstestets filterinställningar för experimentgrupp 2.

6.3.4 Sammanfattning delmål 3

De klass-, associations- och arvsexperiment som genomfördes gav en mycket god grund till hur konfigurationsfunktionen skulle byggas upp och hur den skulle fungera i implementationen, det vill säga själva syftet med delmål 3. Experimenten bidrog dessutom till kunskaper som var nödvändiga för att kunna lösa delmål 4, det vill säga implementationen.

En viktig erfarenhet som experimenten gav var kunskapen om att söka igenom trädstrukturer i XMI-dokumenten för att hitta önskad information. Dessutom gav experimenten en inblick i hur implementationen skulle gå att göra generell, såtillvida att det genom inställningar går att anpassa den för olika typer av information.

6.4 Delmål 4 - Implementation

De experiment som genomfördes gav många idéer om hur den slutgiltiga implementationen skulle utformas. En analys av experimentens resultat resulterade i en uppsättning krav som implementationen måste uppfylla.

- **Generell design** – Implementationen ska inte vara låst till att enbart stödja klassdiagram eller klass-, associations- och arvsobjekt. Designen ska vara så generell att det på ett enkelt sätt går att utöka programmet till att även hantera exempelvis ett nytt UML-objekt.
- **Konfigurerbarhet** – Inställningar för hur programmet uppför sig ska kunna ändras för att:
    - Genom ändringar i konfigurationen även stödja framtida förändringar av XMI-standarden.
    - Stödja kravet på generell design och möjliggöra ett utbyggbart program. Möjlighet att klara av andra diagramtyper, andra UML-objekt (inte bara klasser, associationer och arv).
    - Ge möjlighet att ange olika sätt att leta efter samma information. Informationen kan finnas på olika ställen i XMI-dokumenten och representeras på olika sätt, till exempel under element med olika namn och ibland som textnoder eller som attribut.

- **Sökträd** – Möjlighet att specificera på vilken nivå i ett XML-dokument som efterfrågad information finns och i vilken ordning elementen ska uppträda, till exempel först ett element med namnet ”klass” sedan ett element med ”namn”. Ibland har det visat sig vara avgörande att leta efter informationen på en särskild plats i XML-hierarkin.
- **Reguljära uttryck** – Flera av experimenten visade att snarlika taggnamn användes. Med hjälp av reguljära uttryck går det att hitta dem genom att söka efter minsta gemensamma nämnaren.
- **Presentation** – Möjlighet att presentera informationen på ett lättförståligt, lättläst och tydligt sätt.

Kraven resulterade i en strukturerad design för implementationen. I designen bestämdes funktionerna för programmets olika delar och deras inbördes relationer. Figur 6.1 visar de mest grundläggande delarna av implementationen i ett klassdiagram (se bilaga 1 för ett mer detaljerat klassdiagram).

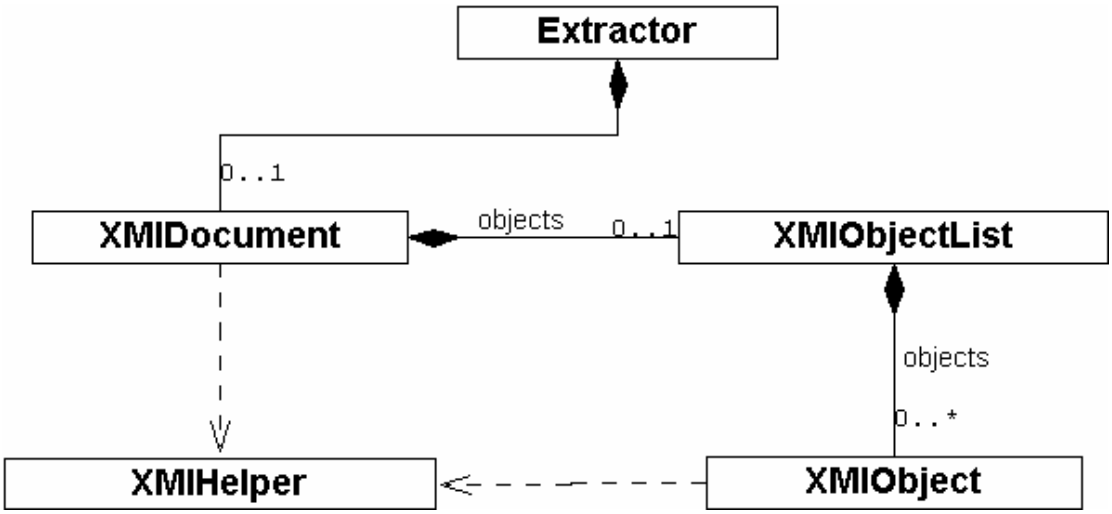


Fig. 6.1. Klassdiagram över den slutgiltiga implementationen.

Extractor Extractor tillhandahåller ett grafiskt gränssnitt mot användaren. Via detta går det att öppna XMI-dokument varefter den information som hittas presenteras på datorskärmen (se bilaga 2).

XMIDocument XMIDocument är den del av programmet som läser in XMI-dokumentet och sorterar ut XMI-objekt efter informationen från konfigurationsfilen och lägger dessa i XMIObjectList (se bilaga 3).

XMIObjectList Hanterar samlingar av XMI-objekt och innehåller funktioner för att sköta dessa listor på olika sätt (se bilaga 4).

XMIObject Representerar ett XMI-objekt; till exempel klass, association eller arv. Ett XMIObject hanterar den information som är specifik för just den typen av objekt, exempelvis attribut och operationer för klasser. Vilken information objekten innehåller bestäms av konfigurationsfilen (se bilaga 5).

XMIHelper Innehåller användbara funktioner för att underlätta hanteringen av XMI-dokument. Bland annat innehåller XMIHelper funktioner för att söka i XML-träd på olika sätt (se bilaga 6).

Grundläggande för implementationen är möjligheten till konfiguration. Detta har möjliggjorts med hjälp av en konfigurationsfil. Konfigurationsfilen är ett XML-dokument. I det finns ett antal element som beskriver hur information kan hittas i XMI-dokument. Konfigurationsfilen är indelad i tre olika delar; objekt, objektmedlemmar samt sökvägar till information om dessa.

- **Objekt** är ett XMI-objekt, till exempel klass, association eller arv.
- **Objektmedlemmar** är egenskaper som ett objekt kan ha, till exempel namn, attribut och operationer.
- **Sökvägarna** beskriver var i XMI-dokumenten informationen om objektmedlemmarna kan hittas. Mer än en sökväg ska kunnas uppges ifall informationen kan finnas på olika ställen och definieras på olika sätt. Programmet söker alltså på olika sökvägar för att hitta den önskade informationen.

Eftersom implementationen endast hanterar XMI-objekt och dess medlemmar har en generell design uppnåtts. I konfigurationsfilen går det att styra vilka XMI-objekt som implementationen ska hitta och var i XMI-dokumenten information objekten kan hittas. Det går alltså att lägga till och ta bort XMI-objekt, egenskaper för objekt eller sökvägar till information genom att ändra i konfigurationsfilen och på så sätt ändra vad implementationen letar efter.

Utöver konfigurationsfilen är programmets sökfunktion en viktig del. Sökfunktionen använder sig av innehållet i konfigurationsfilen för att hitta informationen i XMI-dokumenten. Inledningsvis var tanken att använda NodeFilter för sökfunktionen. Konceptet att använda NodeFilter var tänkt att sortera bort oönskade element ur XMI-dokumenten. Detta fungerade bra i experimenten eftersom det mellan experimenten gick lätt att ändra filterkriterier. Dock behövde filtret i den slutgiltiga implementationen vara generellt och kunna anpassas för att hantera olika typer av objekt och inte enbart dem som angivits från början. Därför övergavs idén med NodeFilter till förmån för en ny och smidigare sökfunktion. Denna sökfunktion tar hänsyn till var i XML-trädet element befinner sig samt möjliggör sökning på elementnamn via reguljära uttryck. Sökfunktionen fungerar enligt följande princip förklarad i pseudokod (se även rad 142 till 212 i bilaga 6):

```
search ($xMLTree, $searchPath) returns @matches
begin
     initialize @matches;

     while ($xMLTree has more Nodes)
     do
           $Node = next $xMLTree node;

            if ($Node = (first part of $searchPath))
            then
                 if ($searchPath is at the end)
                 then
                       add $Node to @matches;
                 else if (not at the end of $searchPath)
                 then
```

```
                    add search($Node, (next part of $searchPath))
                        to @matches;
                end
            end
        end
        return @matches;
end
```

Funktionen jämför stegvis en del av ett XML-träd med en sökväg till ett element. Stämmer hela sökvägen överens med trädets struktur läggs elementet till i en lista där samtliga funna element sparas. När väl hela XML-trädet har sökts igenom, skickas de funna elementen tillbaka till programmet. Jämförelsen av sökvägen görs med hjälp av ett reguljärt uttryck.

Implementationen som har skapats är ett Javaprogram med ett grafiskt gränssnitt. Från det grafiska gränssnittet är det möjligt för en användare att välja vilket XMI-dokument som ska öppnas. Informationen i det valda XMI-dokumentet samlas sedan in med hjälp av uppgifter från konfigurationsfilen. Därefter presenteras den insamlade informationen på datorskärmen.

Fig. 6.2. Användaren väljer XMI-dokument.

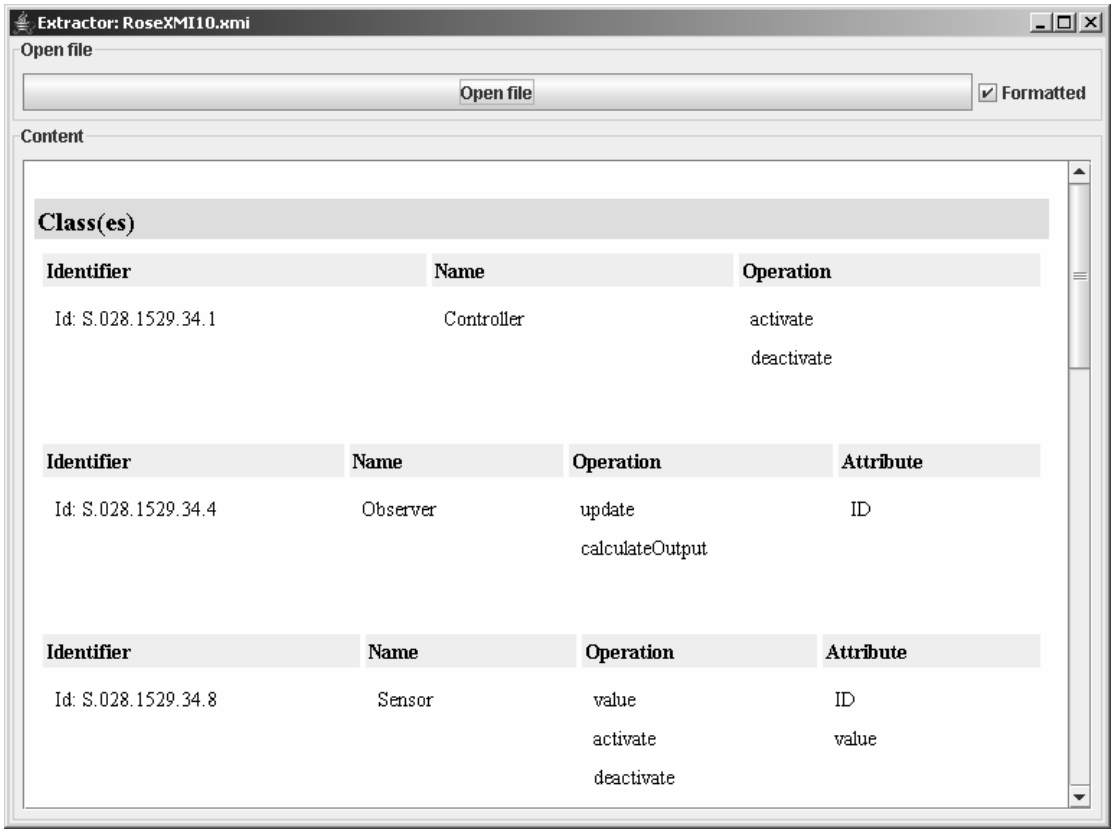


Fig. 6.3. Extraherad information presenteras på datorskärmen.

6.5 Hypotesprövning

Den hypotes som undersökningen har haft för avsikt att verifiera lyder:

Det är möjligt att med en enkel och flexibel extraktor utvinna och lista relevant information om ett UML-diagram som har sparats i XMI-format, utan hjälp från DTD eller XML-schema.

Underlaget för hypotesprövningen har varit samtliga XMI-dokument som använts i undersökningen (*se tabell 5.1 och 5.2*). Hypotesen har prövats genom att öppna XMI-dokumenten i den slutgiltiga implementationen och därefter kontrollera den information som hittas.

Resultatet av hypotesprövningen är att implementationen, det vill säga extraktorn, lyckas hitta samma information från samtliga XMI-dokument. Därmed är hypotesen verifierad inom ramen för den här undersökningen. Inga anspråk görs på att extraktorn även ska klara att extrahera information från XMI-dokument som inte använts i undersökningen.

Extraktorn hittar informationen med hjälp av en konfigurationsfil som innehåller sökvägar till den relevanta informationen. Dessa sökvägar är resultatet av tidigare analyser och tester av dokumenten. Av den anledningen kan extraktorn endast hitta specifik information på ställen i XMI-dokument där samma information hittats vid tidigare tillfällen. Extraktorn klarar alltså att hitta information i XMI-dokument som följer samma struktur som de dokument som använts i den här undersökningen. I andra XMI-dokument, som inte följer en redan känd struktur, hittas inte nödvändigtvis den eftersökta informationen. Konfigurationsfilen kan dock anpassas för att stödja ytterligare strukturer, när de påträffas, och på så sätt göra extraktorn användbar för ett

större antal XMI-dokument. Tack vare detta uppfylls även det i hypotesen uppställda kravet på att extraktorn ska vara flexibel.

6.6 Slutsats

En slutsats som kan dras utifrån den undersökning som gjorts är att:

Sättet på vilken XMI-standarden används skiljer sig åt mellan modelleringsverktyg. Verktygen är ändå konsekventa i sin egen användning, det vill säga dokumenten följer en genomgående struktur för på vilket sätt olika typer av information är representerad. Förutsatt att sättet på vilket verktyg använder XMI-standarden är känt sedan tidigare går det att extrahera informationen.

Den slutliga implementationen, det vill säga extraktorn, lyckas hitta, på förhand påvisad, information om den modell som sparats i de XMI-dokument som använts i undersökningen. Detta främst tackvare extraktorns flexibla konfigureringsmöjligheter. Därmed förklaras den uppställda hypotesen verifierad.

7 Diskussion

I detta kapitel diskuteras undersökningens metod och resultat utifrån positivt och negativt utfall. Diskussionen avslutas med uppslag för fortsatt arbete inom området.

7.1 Kritisk granskning av arbetet

Inledningsvis i undersökningen gjordes en litteraturstudie. Studien gav en bra introduktion och överblick över problemområdet. Dock visade sig den stora mängden litteratur och information om UML och XML vara näst intill oöverskådlig. Detta medförde att jag endast kunde studera en bråkdel av det material som finns inom området. Trots detta har jag fått en bra inblick i såväl UML som XML. Utöver UML och XML har litteraturstudien dessutom kretsat kring XMI. Inom detta område var det dessvärre svårare att hitta adekvat litteratur. Från början var tanken att tidigare forskning kring XMI skulle ge en bra vägledning för den här undersökningen, men det har varit svårt att finna forskningsresultat som specifikt berör problemområdet i den här undersökningen. Det material som påträffats har i de allra flesta fall endast gett vag vägledning till undersökningen.

Efter litteraturstudien påbörjades experimenten, med vilka avsikten var att studera och tolka de XMI-dokument som använts i undersökningen. Målet var att identifiera olika verktygs skillnader i användningen av XMI-standarden. Genomförandet av experimenten bidrog till kunskaper om XMI som litteraturstudien inte kunde ge, vilket var en mycket positiv aspekt av experimenten. Experimenten gav med andra ord en konkret förståelse för XMI och de problem som skulle undersökas. De faktiska resultaten av experimenten, det vill säga att användandet av XMI skiljer sig mellan verktygen, var dock inte speciellt förvånande. Detta hade nämligen litteraturstudien förvarnat om.

Implementationen var det sista steget i undersökningen. De erfarenheter som experimenten bidrog till gav en god grund till kunskapen om vilka krav som behövde ställas på implementationen. Således visade sig resultaten av experimenten vara användbara och därmed lyckade. Även valet av plattform visade sig vara lyckat för implementationen, därför att arbetet fortlöpte utan några större problem. En anledning till detta är att Java och Xerces2 fungerade mycket bra tillsammans, till och med över förväntan. De kompabilitetsproblem som befarades visade sig vara grundlösa. Det krävdes inga krångliga installationer eller specifika inställningar för att de skulle vara kompatibla.

7.2 Diskussion runt resultatet

Trots att hypotesprövningen föll ut med förväntat resultat går det att diskutera vissa aspekter av resultatet. Först och främst var tanken från början att användaren helst inte skulle behöva omfattande kunskaper om XMI eller XML för att kunna ändra i konfigurationsfilen. Tyvärr kräver den slutliga konfigureringslösningen inte enbart kunskap om XMI och XML utan även grundläggande kunskap om reguljära uttryck. Dessutom måste användaren ändra inställningar manuellt, vilket inte är särskilt användarvänligt. Ett tillägg till programmet från vilket det via ett grafiskt gränssnitt skulle gå att ändra inställningar i konfigurationsfilen hade varit önskvärt. Omfattningen för undersökningen tillät dock inte att ett sådant gränssnitt kunde utvecklas.

Det krångliga och svårbegripliga konfigurationsförfarandet vägs upp av programmets flexibilitet. Programmets flexibilitet är också det allra mest positiva resultatet av undersökningen och går ut på att ändra konfigurationsfilen för att styra programmets

beteende. Genom att lägga till inställningar kan programmet även extrahera information från XMI-dokument som har skapats i andra verktyg än dem som undersökningen baserats på. Dessutom är det möjligt att ställa in konfigurationsfilen för att hitta nya typer av XMI-objekt.

Implementationen fungerar bra med tanke på att den modell som har används för experimenten är förhållandevis enkel. Det är inte säkert att implementationen skulle kunna hantera en mer komplex modell på samma sätt.

7.3 Fortsatt arbete

Ett givet område för fortsatt arbete är givetvis att utveckla implementationen för att utöka användbarheten. Följande är exempel på områden där ett fortsatt arbete skulle förbättra implementationen.

- Förståelsen för de modeller som extraheras skulle höjas om resultatet visualiserades. Nu är exempelvis relationerna för associationer och arv endast förklarade med ett unikt id-nummer. Detta gör att det kan vara svårt att förstå vilka objekt som relaterar till varandra. Genom att visualisera detta i form av ett diagram skulle förståelsen betydligt förbättras. Dessutom skulle det vara möjligt att skriva ut modellen på ett överskådligt sätt.
- En uppenbar brist med den befintliga implementationen är att den inte klarar av objektmedlemmar som har mer än en egenskap. I förlängningen vore det önskvärt om implementationen klarade just detta. Ett eventuellt resultat av detta kunde vara att implementationen då skulle kunna hantera mer komplexa modeller.
- Implementationens användningsområden skulle utökas om det vore möjligt att exportera den extraherade informationen till ett format anpassat för en viss typ av modelleringsverktyg. Denna funktion skulle även kunna användas för att korrigera felaktigheter i XMI-dokument eller översätta ett sparat XMI-dokument till en annan XMI- eller UML-version.
- För tillfället stöds endast extrahering av information från klassdiagram. Andra typer av diagram kan inte hanteras. Ett antagande är att det bör gå att applicera andra typer av diagram utan några (större) modifikationer i programkoden. Det som krävs är antagligen endast att konfigurationsfilen anpassas för diagramtypen.

Ytterligare ett område där det finns utrymme för fortsatt forskning är hur det kommer sig att versionsuppgifterna i XMI-dokumenten ofta är felaktiga eller bristfälliga. Oavsett det beror på standarden medvetet inte följs eller om standarden helt enkelt inte förstås av dem som skapar modelleringsverktyg så är det ett intressant forskningsområde.

Referenser

Alhir, S. S. (2003) *Learning UML*. Sebastopol, CA: O'Reilley & Associates, Inc.

Alihr, S. S. (1998) *UML in a nutshell.* Sebastopol, CA: O'Reilley & Associates, Inc.

Apache XML Project. (2004) *NodeFilter*. I: XML Standard API. Tillgänglig på Internet: http://xml.apache.org/xerces2-j/javadocs/api/org/w3c/dom/traversal/NodeFilter.html [Hämtad 2005-05-17]

Armstrong, E. (2001) *Working with XML - The Java API for Xml Processing (JAXP) Tutorial*. [Elektronisk]. Tillgänglig på Internet: http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/index.html [Hämtad 2005-04-18].

Avison, D. & Shah, H. (1997) *The Information Systems Development Life Cycle: A First Course in Information Systems.* Maidenhead: McGraw-Hill Book Company Europe.

Azelia. (2002) *XML – En skonsam men effektiv introduktion.* [Elektronisk]. Göteborg: Azelia AB. Tillgänglig på Internet: http://www.azelia.se/xml.pdf [Hämtad 2005-04-18].

Berndtsson, M., Hansson, J., Olsson & Lundell, B. (2002) *Planning and Implementing your Final Year Project with Success!* London: Springer-Verlag.

Christophides, V., Plexousakis, D., Scholl, M. & Tourtounis, S. (2003) On labeling schemes for the semantic web. Presenterat vid *WWW2003- The 12th International World Wide Web Conference,* Budapest, Ungern, 20-24 Maj, 2003, s. 544-555.

Elmasri, R. & Navathe, B. (2004) *Fundamentals of Database Systems.* Boston, MA: Pearson Education, Inc

Fowler, M. (2000) *UML Distilled – A Brief Guide to the Standard Object Modelling Language*. Harlow, England: Addison-Wesley.

Grose, J.G., Doney, C.D. & Brodsky, A.B. (2002) *Mastering XMI.* New York, NY: John Wiley & Sons Inc.

Homer, A., Sussman, D. & Fussel, M. (2004) *A First Look at ADO.NET and System.Xml v. 2.0.* Boston, MA: Pearson Education, Inc.

Jiang, J.J. och Systä, T., (2002) *UML model exchange using XMI*. [Elektronisk]. Projektrapport, Institute of Software Systems, Tampere University of Technology, Finland. Tillgänglig på Internet: http://www.cs.tut.fi/~xmlohj/linkit/XMIReport.pdf [Hämtad 2005-04-03]

Kaushik, R., Bohannon, P., Naughton, J. F, & Korth, H. F. (2002) Covering indexes for branching path queries. Presenterat vid *SIGMOD 2002*, Madison, Wisconsin, USA, 2-3 Juni, 2002. s. 133-144.

Li, Q. & Moon, B. (2001) Indexing and querying xml data for regular path expressions. Presenterat vid *VLDB 2001 - 27th international conference on Very Large Databases*, Rom, Italien, 11-14 September, 2001, s. 315-326.

Milo, T. & Suciu, D. (1999) Index structures for path expressions. Presenterat vid *ICDT – 7th International Conference on Database Theory*, Jerusalem, Israel, 10-12 januari, 1999, s. 277-295.

Nader, J. C. (1998). *Prentice Hall's illustrated dictionary of computing, third edition.* Sydney: Prentice Hall of Australia Pty Ltd.

Nationalencyklopedin. (2005) *Grafisk framställning.* [Elektronisk]. Tillgänglig på Internet: http://www.ne.se.persefone.his.se/jsp/search/article.jsp?i_art_id=184747 [Hämtad 2005-06-06].

North, S. & Hermans, P. (1999) *Lär dig XML på 3 veckor.* Sundbyberg: Pagina förlags AB

Ollner, J. (2005). Standardisering. I: *Nationalencyklopedin*. [Elektronisk]. Tillgänglig på Internet: http://ne.se.persefone.his.se/jsp/search/article.jsp?i_art_id=314053 [Hämtad 2005-04-23]

OMG. (2000a) *OMG XML Metadata Interchange (XMI) Specification – Version 1.0.* [Elektronisk]. Object Management Group. Tillgänglig på Internet: http://www.omg.org/docs/formal/00-06-01.pdf [Hämtad 2005-04-03]

OMG. (2000b) *OMG XML Metadata Interchange (XMI) Specification – Version 1.1*. [Elektronisk]. Object Management Group. Tillgänglig på Internet: http://www.omg.org/docs/formal/00-11-02.pdf

OMG. (2002) *OMG XML Metadata Interchange (XMI) Specification – Version 1.2*. [Elektronisk]. Object Management Group. Tillgänglig på Internet: http://www.omg.org/docs/formal/02-01-01.pdf [Hämtad 2005-04-03]

OMG. (2003a) *OMG Unified Modeling Language Specification – Version 1.5* [Elektronisk]. Object Management Group. Tillgänglig på Internet: http://www.omg.org/docs/formal/03-03-01.pdf [Hämtad 2005-04-03]

OMG. (2003b) *XML Metadata Interchange (XMI) Specification – Version 2.0*. [Elektronisk]. Object Management Group. Tillgänglig på Internet: http://www.omg.org/docs/formal/03-05-02.pdf [Hämtad 2005-04-03]

OMG. (2003c) *UML 2.0 Infrastructure Specification.* [Elektronisk]. Object Management Group. Tillgänglig på Internet: [Elektronisk]. http://www.omg.org/docs/ptc/03-09-15.pdf [Hämtad 2005-05-16].

OMG. (2004) *UML 2.0 Superstructure Specification.* [Elektronisk]. Object Management Group. Tillgänglig på Internet: [Elektronisk]. http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.zip [Hämtad 2005-05-16].

OMG. (2005) *About the Object Management Group™ (OMG™)*. [Elektronisk]. Object Management Group. Tillgänglig på Internet: http://www.omg.org/gettingstarted/gettingstartedindex.htm [Hämtad 2005-06-04].

Patel, R. & Davidson, B. (2003) *Forskningsmetodikens grunder.* Lund: Studentlitteratur.

Persson, A. (2004) *Integrationsmöjligheter för modelleringsverktyg baserade på öppen källkod.* [Elektronisk]. Magisteruppsats, Institutionen för kommunikation och information, Högskolan i Skövde. Tillgänglig på Internet: http://www.ida.his.se/ida/htbin/exjobb/2004/HS-IKI-MD-04-306 [Hämtad 2005-04-03]

Persson, A., Gustavsson, H., Linds, B., Lundell, B., Mattson, A. & Ärlig, U. (2005a) Adopting Open Source development tools in a commercial production environment – are we locked-in? Prestenteras vid *EMMSAD'05 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design* Porto, Portugal, 13-14 Juni, 2005.

Persson, A., Gustavsson, H., Linds, B., Lundell, B., Mattson, A. & Ärlig, U. (2005b) OSS tools in a heterogenous environment for embedded systems modelling: an analysis of adoptons of XMI. Presenteras vid *5th Workshop on open source software engineering,* St. Louis, MO, USA, 17 maj, 2005.

Skansholm, J. (2000) *C++ direkt.* Lund: Studentlitteratur.

Statskontoret. (2000a) *XML-familjen - Vad är Document Object Model (DOM)?* [Elektronisk]. Tillgänglig på Internet: http://www.statskontoret.se/upload/Publikationer/2000/200035.pdf [Hämtad 2005-04-03]

Statskontoret. (2000b) *XML-familjen - Vad är XML-Schema och XML-DTD?* [Elektronisk]. Tillgänglig på Internet: http://www.statskontoret.se/upload/Publikationer/2000/200032.pdf [Hämtad 2005-04-03]

Statskontoret. (2000c) *XML-familjen - Vad är XML-Länkar?* [Elektronisk]. Tillgänglig på Internet: http://www.statskontoret.se/upload/Publikationer/2000/200034.pdf [Hämtad 2005-04-03]

Statskontoret. (2001) *Standarder inom XML-världen*. [Elektronisk]. Tillgänglig på Internet: http://www.statskontoret.se/upload/Publikationer/2001/2001312.pdf [Hämtad 2005-04-03]

Stevens, P. (2003) Small-Scale XMI Programming: A Revolution in UML Tool Use? *Automated Software Engineering, 10*(1), 7–21

The Apache Software Foundation. (1999) *The Apache Software License, Version 1.1*. [Elektronisk]. Tillgänglig på Internet: http://xml.apache.org/LICENSE [Hämtad 2005-05-22]

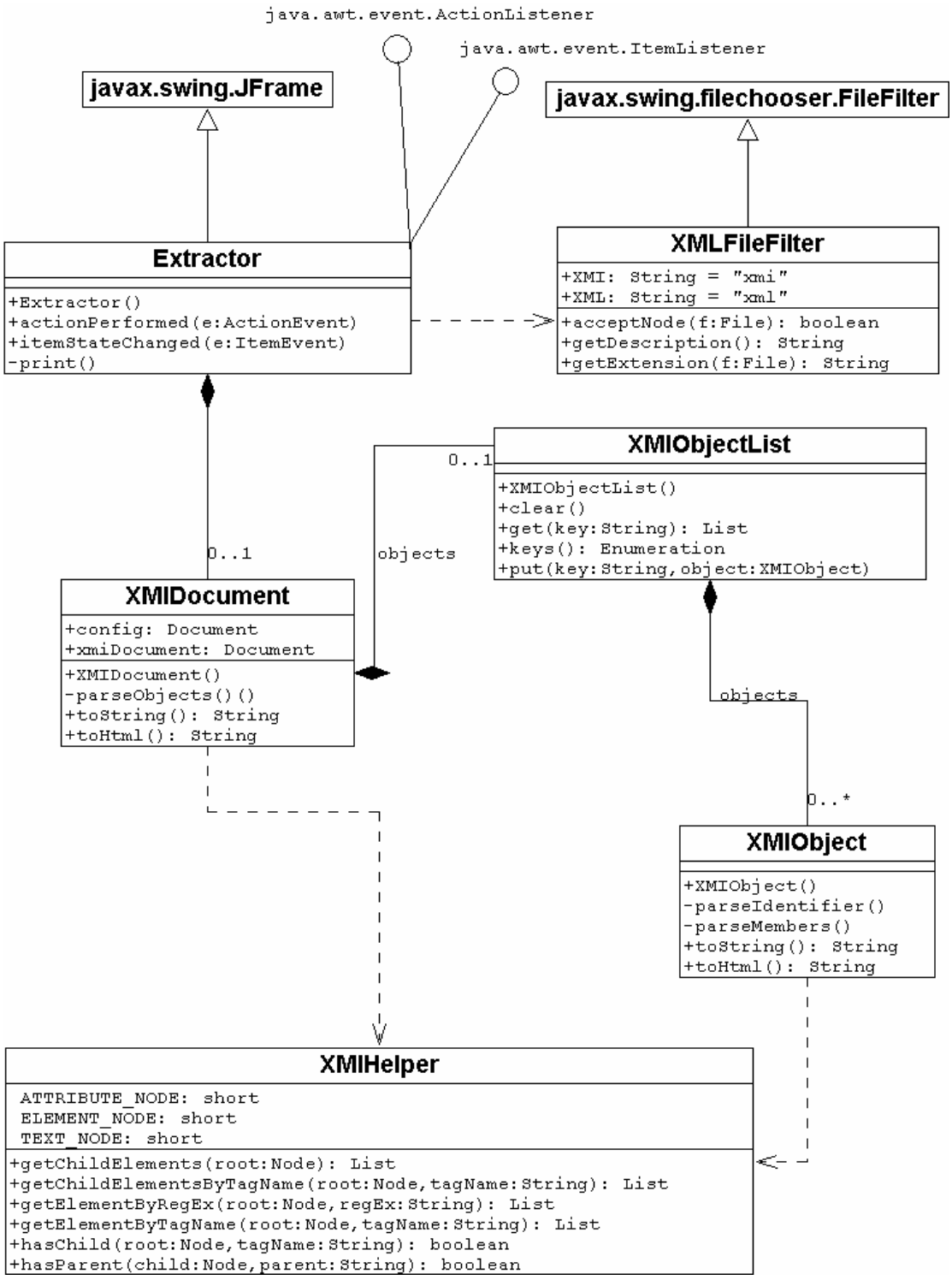
The Apache Software Foundation. (2004) *API JavaDoc*. [Elektronisk]. Tillgänglig på Internet: http://xml.apache.org/xerces2-j/api.html [Hämtad 2005-05-22]

W3C. (2004) *Extensible Markup Language (XML) 1.0 (Third Edition)* [Elektronisk]. Tillgänglig på Internet: http://www.w3.org/TR/2004/REC-xml-20040204/ [Hämtad 2005-06-06]

W3C. (2005) *Document Object Model (DOM)* [Elektronisk]. Tillgänglig på Internet: http://www.w3.org/DOM/ [Hämtad 2005-06-06]

Åström, P. (1999) *XML – Extensive Markup Language.* Stockholm: Docendo Läromedel, AB.

Bilaga 1 Klassdiagram för implementationen



Bilaga 2 Extractor

Extractor.java

```
1  import java.io.*;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  /** The GUI for the extractor application */
7  public class Extractor extends JFrame implements ActionListener, ItemListener
8  {
9
10     /** Name of application */
11     private static final String TITLE = "Extractor";
12     /** button to file opener */
13     private JButton openButton;
14     /** file chooser */
15     private JFileChooser fileChooser;
16     /** The chosen file */
17     private String file = "";
18     /** where the info gets printed */
19     private JEditorPane text;
20     /** checkbox for user do get info in plain text or in HTML */
21     private JCheckBox isHTML;
22     /** the XMI-document with the information about the UML-model */
23     private XMIDocument xmiDoc;
24
25     /** Constructor */
26     public Extractor()
27     {
28         super(TITLE);
29
30         JPanel infoPanel = new JPanel();
31         infoPanel.setLayout(new BorderLayout());
32         infoPanel.setBorder(BorderFactory.createCompoundBorder(
33             BorderFactory.createTitledBorder("Content"),
34             BorderFactory.createEmptyBorder(4, 4, 4, 4)
35             ));
36
37         text = new JEditorPane();
38         text.setEditable(false);
39         JScrollPane scrollPane = new JScrollPane(text);
40         scrollPane.setBounds( 10, 10, 280, 180 );
41         infoPanel.add( scrollPane, BorderLayout.CENTER );
42
43
44         JPanel filePanel = new JPanel();
45         filePanel.setLayout(new BorderLayout());
46         filePanel.setBorder(BorderFactory.createCompoundBorder(
47             BorderFactory.createTitledBorder("Open file"),
48             BorderFactory.createEmptyBorder(4, 4, 4, 4)
49             ));
50
51         openButton = new JButton("Open file");
52         openButton.addActionListener(this);
53
54         isHTML = new JCheckBox("Formatted", true);
55         isHTML.addItemListener(this);
56
57         filePanel.add(openButton, BorderLayout.CENTER);
58         filePanel.add(isHTML, BorderLayout.EAST);
59
60         JPanel mainPanel = new JPanel();
61         mainPanel.setLayout(new BorderLayout());
62         mainPanel.add(infoPanel, BorderLayout.CENTER);
63         mainPanel.add(filePanel, BorderLayout.NORTH);
64
65         getContentPane().add(mainPanel);
66     }
67
68     /** Handels actions generated from opening a file */
69     public void actionPerformed(ActionEvent e)
70     {   //Handle open button action.
71         if (e.getSource() == openButton)
72         {
73             fileChooser = new JFileChooser();
74             fileChooser.setAcceptAllFileFilterUsed(false);
75             fileChooser.setFileFilter(new XMLFileFilter());
76             int returnVal = fileChooser.showOpenDialog(openButton);
77
78             if(returnVal == JFileChooser.APPROVE_OPTION)
79             {
80                 file = fileChooser.getSelectedFile().getPath();
81                 print();
82             }
83         }
84     }
85
```

Bilaga 2 Extractor

Extractor.java

```
86      /** Handles checkbox selection */
87      public void itemStateChanged(ItemEvent e)
88      {
89          Object source = e.getItemSelectable();
90          if (source == isHTML)
91          {
92              print();
93          }
94      }
95
96      /** Prints fetched XMI document on screen */
97      private void print()
98      {
99          xmiDoc = null;
100
101         if (file == null || file.equals(""))
102         {
103             return;
104         }
105
106         try
107         {
108             xmiDoc = new XMIDocument(file);
109             String fileName = file.substring(file.lastIndexOf(File.separator)+1);
110
111             //Add filename to titlebar.
112             setTitle(TITLE + ": " + fileName);
113
114             if (isHTML.isSelected())
115             {
116                 text.setContentType("text/html");
117                 text.setText(xmiDoc.toHtml());
118             }
119             else
120             {
121                 text.setContentType("text/plain");
122                 text.setText(xmiDoc.toString());
123             }
124
125             //Scroll to begining of page
126             text.setCaretPosition(0);
127         }
128         //Catch all exceptions and report.
129         catch (Throwable exception)
130         {
131             exception.printStackTrace(System.err);
132             text.setText("Error:\n" + exception.getMessage());
133         }
134         //repaint
135         update(getGraphics());
136     }
137
138     public static void main(String[] args) throws Exception {
139
140         Extractor frame = new Extractor();
141         frame.addWindowListener(new java.awt.event.WindowAdapter()
142             {
143                 public void windowClosing(java.awt.event.WindowEvent e)
144                 {
145                     System.exit(0);
146                 }
147             });
148         frame.setSize(720, 480);
149         frame.setVisible(true);
150     }
151 }
```

Bilaga 3 XMIDocument

XMIDocument.java

```
1  import org.apache.xerces.parsers.*;
2  import org.w3c.dom.*;
3  import org.xml.sax.*;
4  import java.util.*;
5  import java.io.*;
6
7  /** Object containing the information from a XMI document.
8      Provides a way to specify how this informations should
9      be retrived with the help from a configuration file.
10 */
11 class XMIDocument
12 {
13     /** The XMI document */
14     private Document document;
15     /** The Configuration file */
16     private Document config;
17
18     /** constant path to configuration file */
19     private static final String CONF_FILE = "xmi_conf.xml";
20
21     /** object containding all the parsed xmi objects */
22     XMIObjectList objects = new XMIObjectList();
23
24     /** Constuctor. Starts the parsing of the XMI document */
25     public XMIDocument(String xmiFile) throws Exception
26     {
27
28         DOMParser parser = new DOMParser();
29
30         /*
31         this is a workaround to prevent the parser from
32         trying to read a dtd even if one does't exist.
33         DTD's are not interesting for this application but
34         obviously parsers always reads the dtd even if it is set to
35         be non-validating, because there might be default values declared in
36         the dtd.
37         */
38         parser.setEntityResolver(
39             new EntityResolver()
40             {
41                 //always returns an empty inputstream
42                 //witch is the same as an empty dtd-file
43                 public InputSource resolveEntity(String publicId, String systemId)
44                     throws SAXException, IOException
45                 {
46                     return new InputSource(new ByteArrayInputStream("".getBytes()));
47                 }
48             }
49         );
50
51         //parse xmi document
52         parser.parse(xmiFile);
53         document = parser.getDocument();
54
55         //parse config file
56         parser.parse(CONF_FILE);
57         config = parser.getDocument();
58
59         parseObjects();
60     }
61
62
63     /** Parse the information from the XMI document */
64     private void parseObjects()
65     {
66         Element root = config.getDocumentElement();
67         //get a list of all XMI objects from the config file
68         List objectConfigList = XMIHelper.getElementsByTagName(root, "object");
69
70         //parse the objects with help from the info from the config file
71         for (int i=0;i<objectConfigList.size();i++)
72         {
73             Element objectConfig = (Element)objectConfigList.get(i);
74             String key = objectConfig.getAttribute("name");
75
76             List list = XMIHelper.getElementsByRegEx(document.getDocumentElement(),objectConfig.getAttribu
77
78             if (list != null)
79             {
80                 Iterator it = list.iterator();
81
82                 while (it.hasNext())
83                 {
84                     Element n = (Element)it.next();
85                     //OBS! hardcoded (hårdkodat), should be retrived from the config file instead.
```

XMIDocument.java

```
                    if (!n.hasAttribute("xmi.idref"))
                    {
                        objects.put(key, new XMIObject(n, objectConfig));
                    }
                }
            }
        }
    }

    /** Returns the parsed information from the XMI document */
    public String toString()
    {
        String out = "";

        Enumeration keys = objects.keys();

        while (keys.hasMoreElements())
        {
            String key = (String)keys.nextElement();
            List xmiObjects = (List)objects.get(key);

            Iterator it = xmiObjects.iterator();

            out += key + "\n-------------\n";
            while (it.hasNext())
            {
                out += it.next().toString();
            }
        }
        return out;
    }

    /** Returns the parsed information from the XMI document in HTML */
    public String toHtml()
    {
        String out = "<html><body>\r\n";
        out += "<a name=\"DOCUMENT_START\">\r\n";
        out += "<table cellspacing=4 cellpadding=2 border=0>";
        Enumeration keys = objects.keys();

        while (keys.hasMoreElements())
        {
            String key = (String)keys.nextElement();
            List xmiObjects = (List)objects.get(key);

            Iterator it = xmiObjects.iterator();

            if (key.endsWith("s"))
                key += "(es)";
            else
                key += "(s)";

            out += "<tr><td bgcolor=\"#dddddd\"><font size=\"+1\"><b>" + key + "</font></b></td></tr>";

            out += "<tr><td valign=\"top\">";
            while (it.hasNext())
            {
                out += ((XMIObject)it.next()).toHtml();
                out += "<tr><td> </td></tr>";
            }
            out += "</td></tr><tr><td> </td></tr>";
        }
        out += "</table></body></html>";

        return out;
    }
}
```

Bilaga 4 XMIObjectList

XMIObjectList.java

```
 1  import java.util.*;
 2
 3  /**
 4  Class used to hold all XMIObjects parsed from the XMI file.
 5  Provides functionallity to add and retrive XMI objects from underlying collection.
 6  */
 7  class XMIObjectList
 8  {
 9      /** Map containing all xmi objects */
10      private Hashtable objects;
11
12      /** Default constructor. */
13      public XMIObjectList()
14      {
15          objects = new Hashtable();
16      }
17
18      /** Adds an object to the collection by specifying a key and the object to add.
19          The key represents the type of xmi object. For exampel should all UML classes be
20          added under the key "Class" an so on...
21      */
22      public void put(String key, XMIObject object)
23      {
24          if (objects.containsKey(key))
25          {
26              ((ArrayList)objects.get(key)).add(object);
27          }
28          else
29          {
30              ArrayList list = new ArrayList();
31              list.add(object);
32              objects.put(key, list);
33          }
34      }
35
36      /** Gets xmi objects by the specified key from the memeber.
37          Intended to be used for retriving all instances of one specified type of xmi object.
38          For exampel by providing the key "Class", all instances of UML classes will be returned.
39      */
40      public List get(String key)
41      {
42          return (List)objects.get(key);
43      }
44
45      /** Returns the all keys for the stored xmi objects */
46      public Enumeration keys()
47      {
48          return objects.keys();
49      }
50
51      /** Clears the map from xmi objects */
52      public void clear()
53      {
54          objects.clear();
55      }
56  }
57
```

Bilaga 5 XMIObject

XMIObject.java

```
import org.w3c.dom.*;
import java.util.*;

/** Class acts as a representation of a XMI object with
    the properties (members) read from the configuration file
*/
class XMIObject
{
    /** The part of the XMI file representing this object*/
    private Element root;
    /** Configuration for this type of xmi object*/
    private Element config;
    /** identifier */
    private String id = "";
    /** identifier */
    private String uuid = "";
    /** identifier */
    private String label = "";
    /** Map containing properties for this object */
    private Hashtable members = new Hashtable();

    /** Constructor */
    public XMIObject(Node xmi, Node conf)
    {
        root    = (Element)xmi;
        config  = (Element)conf;
        parseIdentifier();
        parseMembers();
    }

    /** Sets the identifier properties for this instance.
        Any of these identifiers can later be used to uniqely
        identify this paricular XMI object instance.
    */
    private void parseIdentifier()
    {
        NamedNodeMap attribs = ((Element)root).getAttributes();

        for (int i = 0; i < attribs.getLength(); ++i)
        {
            if (attribs.item(i).getNodeName().equals("xmi.id"))
                id = attribs.item(i).getNodeValue();
            if (attribs.item(i).getNodeName().equals("xmi.uuid"))
                uuid = attribs.item(i).getNodeValue();
            if (attribs.item(i).getNodeName().equals("xmi.label"))
                label = attribs.item(i).getNodeValue();
        }
    }

    /** Reads properties from the config file and fetches the information from the XMI file. */
    private void parseMembers()
    {
        //The members for this xmi object
        List memberConfigNodes =    XMIHelper.getChildElementsByTagName(config, "member");
        Iterator confIt = memberConfigNodes.iterator();

        //Collect the info for each member
        while (confIt.hasNext())
        {
            Element memberConfigNode = (Element)confIt.next();
            String memberName = memberConfigNode.getAttribute("name");
            //get all the different options for retriving info about the member
            List memberOptions = XMIHelper.getChildElementsByTagName(memberConfigNode, "option");

            Iterator optIt = memberOptions.iterator();
            //go throug all options
            while (optIt.hasNext())
            {
                Element option = (Element)optIt.next();
                String path = option.getAttribute("path");
                String type = option.getAttribute("type");
                //get the member value(s)
                List tmpList = XMIHelper.getElementsByRegEx(root, path);

                Iterator valueNodes = tmpList.iterator();

                ArrayList values = new ArrayList();
                //gets the value for the member and stores it.
                while (valueNodes.hasNext())
                {
                    Element e = (Element)valueNodes.next();
                    if (type.equals("attribute"))
                    {
                        String attribName = option.getAttribute("attribute");
                        if (e.hasAttribute(attribName))
```

Bilaga 5 XMIObject

XMIObject.java

```
                        {
                            values.add(e.getAttribute(attribName));
                        }
                    }
                    else if (type.equals("textnode"))
                    {
                        values.add(XMIHelper.getTextNode(e));
                    }
                }

                //if info was found add it to the map.
                if (values != null && !values.isEmpty())
                {
                    members.put(memberName, values.clone());
                }

                values.clear();


            }
        }
    }

    /** prints the info about this object */
    public String toString()
    {
        String out = "";

        if (id != null && !id.equals(""))
            out += "Id: " + id;
        if (uuid != null && !uuid.equals(""))
            out += "Uuid: " + uuid;
        if (label != null && !label.equals(""))
            out += "Label: " + label;

        out += "\n";

        Enumeration keys = members.keys();
        while (keys.hasMoreElements())
        {
            String key = (String)keys.nextElement();
            List xmiMembers = (List)members.get(key);

            Iterator it = xmiMembers.iterator();

            out += key + ":\n";
            while (it.hasNext())
            {
                try
                {
                    out += "  " + it.next().toString() +"\r\n";
                }
                catch (NullPointerException npe)
                {
                    System.out.println("NullPointer trying to print : " + key);
                }
            }
        }
        return out + "\r\n";
    }


    /** prints the info about this object in HTML */
    public String toHtml()
    {
        String out = "<!-- ##### XMI OBJECT START ##### -->\r\n";
        out += "<table width=700 cellpadding=2 border=0>\r\n<tr>";

        String info = "";

        String identifiers = "<table>";
        boolean isId = false;

        if (id != null && !id.equals(""))
        {
            identifiers += "\r\n\t<tr><td valign=top>Id: " + id +"</td></tr>";
            isId = true;
        }
        if (uuid != null && !uuid.equals(""))
        {
            identifiers += "\r\n\t<tr><td valign=top>Uuid: " + uuid +"</td></tr>";
            isId = true;
        }
        if (label != null && !label.equals(""))
        {
```

Bilaga 5 XMIObject

XMIObject.java

```
171             identifiers += "\r\n\t<tr><td valign=top>Label: " + label +"</td></tr>";
172             isId = true;
173         }
174
175         if (isId)
176         {
177             identifiers += "</table>\r\n";
178             info += "\r\n<td valign=top>" + identifiers + "</td>\r\n";
179             out += "\r\n\t<td align=\"left\" bgcolor=\"#eeeeee\"><b>Identifier</b></td>";
180         }
181
182         Enumeration keys = members.keys();
183
184         while (keys.hasMoreElements())
185         {
186             String key =  (String)keys.nextElement();
187             out += "\r\n\t<td align=\"left\" bgcolor=\"#eeeeee\"><b>" + key + "</b></td>";
188
189             List xmiMembers =  (List)members.get(key);
190
191             Iterator it = xmiMembers.iterator();
192
193             info += "\r\n\t<td valign=\"top\" align=\"left\">\r\n\t\t<table>\n";
194             while (it.hasNext())
195             {
196                 try
197                 {
198                     String value = it.next().toString();
199                     //HTML Encode
200                     value = value.replaceAll("<","&lt;");
201                     value = value.replaceAll(">","&gt;");
202                     info += "\r\n\t\t<tr>\r\n\t\t\t<td valign=\"top\" align=\"top\">" + value +"</td></tr>
203                 }
204                 catch (NullPointerException npe)
205                 {
206                     System.out.println("NullPointer trying to print : " + key);
207                 }
208             }
209             info += "\r\n\t\t</table>\r\n\t</td>";
210         }
211
212         out += "\r\n</tr>\r\n<tr>" + info + "\r\n</tr>\r\n</table>\r\n";
213         out += "<!-- ##### XMI OBJECT END ##### -->\r\n";
214
215         return out;
216     }
217 }
```

Bilaga 6 XMIHelper

XMIHelper.java

```
import org.w3c.dom.traversal.*;
import org.w3c.dom.*;
import java.util.*;
import java.util.regex.*;

/**
Class containing useful methods for dealing with XMI documents.
*/
public final class XMIHelper
{
    public static final short ELEMENT_NODE      = Node.ELEMENT_NODE;
    public static final short TEXT_NODE         = Node.TEXT_NODE;
    public static final short ATTRIBUTE_NODE    = Node.ATTRIBUTE_NODE;

    /** Returns all child elements for specified node */
    public static List getChildElements(Node root)
    {
        List l = new ArrayList();
        NodeList nl = root.getChildNodes();

        Node n = null;

        for (int i = 0; i < nl.getLength(); i++)
        {
            n = nl.item(i);

            if (n.getNodeType() == Node.ELEMENT_NODE)
            {
                l.add(n);
            }
        }

        return l;
    }

    /** Returns all child elements for specified node with the specified tag name */
    public static List getChildElementsByTagName(Node root, String tagName)
    {
        List l = getChildElements(root);
        Iterator it = l.iterator();
        while ( it.hasNext() )
        {
            Element e = (Element)it.next();
            if ( !e.getTagName().equals(tagName) )
            {
                it.remove();
            }
        }

        return l;
    }

    /** Returns the text information from a textnode */
    public static String getTextNode(Node n)
    {
        n.normalize();

        Node text = n.getFirstChild();
        if (text != null && text.getNodeType() == TEXT_NODE)
        {
            return text.getNodeValue();
        }
        return null;
    }

    /** Returns ALL elements with a particular tagname anywhere in the xml-tree */
    public static List getElementsByTagName(Node root, String tagName)
    {
        if (root.getNodeType() == ELEMENT_NODE)
        {
            DocumentTraversal dt = (DocumentTraversal) root.getOwnerDocument();

            NodeIterator it = dt.createNodeIterator(root,
                                                    NodeFilter.SHOW_ELEMENT,
                                                    null,
                                                    false);
            Node n =  it.nextNode();
            ArrayList matches = new ArrayList();
            while (n != null)
            {
                if (n.getNodeName().equals(tagName))
                {
                    matches.add(n);
                }

```

XMIHelper.java

```
86                  n = it.nextNode();
87              }
88
89              if (!matches.isEmpty())
90                  return matches;
91          }
92
93          return null;
94      }
95
96      /** Test if element has a child with a specified tagname */
97      public static boolean hasChild(Element root, String tagName)
98      {
99          NodeList nl = root.getChildNodes();
100
101         Node n = null;
102
103         for (int i = 0; i < nl.getLength(); i++)
104         {
105             n = nl.item(i);
106
107             if (n.getNodeName().equals(tagName))
108             {
109                 return true;
110             }
111         }
112
113         return false;
114     }
115
116     public static boolean hasParent(Node child, String parent)
117     {
118         return child.getParentNode().getNodeName().equals(parent);
119     }
120
121
122
123 /**
124 Method for retrieving nodes in a "regular expression-like" type of way.
125 Also, there is a possibillity to specify where in the XMI-tree the wanted node should be located.
126 <br>
127 <i>Examples:</i>
128 <pre>
129     getElementsByRegEx(root_node, "node");
130             returns all matches anywhere under root_node
131
132     getElementsByRegEx(root_node, "./node");
133             returns all matches directly under root_node
134
135     getElementsByRegEx(root_node, "./../node");
136             returns all matches two "levels" under root_node
137
138     getElementsByRegEx(root_node, "node/another_node");
139             returns all matches anywhere under the root node but with the parent node
140 </pre>
141 */
142     public static List getElementsByRegEx(Node root, String regEx)
143     {
144         List matches = new ArrayList();
145
146         if (regEx.equals("."))
147         {
148             matches.add(root);
149             return matches;
150         }
151
152         //starts with "./"
153         boolean isChild = false;
154         //isParent == regex has '/' somewhere in string
155         boolean isParent = false;
156         //if more than one node parted by '/'
157         String restOfRegEx = "";
158
159         //### prepare path ###
160
161         if (regEx.startsWith("./"))
162         {
163             isChild = true;
164             regEx = regEx.substring(2);
165         }
166
167         int slashStart = regEx.indexOf("/");
168         //contains a '/'
169         if (slashStart > -1)
170         {
```

Bilaga 6 XMIHelper

XMIHelper.java

```
171             isParent = true;
172             //copy path after first slash an add "./" so that we know it's a child
173             restOfRegEx = "./" + regEx.substring(slashStart+1);
174             regEx = regEx.substring(0,slashStart);
175         }
176 
177         //### start matching part ##
178 
179         DocumentTraversal dt = (DocumentTraversal) root.getOwnerDocument();
180         NodeIterator it = dt.createNodeIterator(root,
181                                                 NodeFilter.SHOW_ELEMENT,
182                                                 null,
183                                                 false);
184         Node n = it.nextNode();
185         while (n != null)
186         {   //
187             Pattern p = Pattern.compile(regEx);
188             Matcher m = p.matcher(n.getNodeName());
189 
190             if (m.matches())
191             {
192                 if (((isChild && n.getParentNode() == root) && !isParent) //last subnode
193                     || (!isChild && !isParent)) //any matches
194                 {
195                     matches.add(n);
196                 }
197 
198                 if (((isChild && n.getParentNode() == root) && isParent) // subnode but has children
199                     || (!isChild && isParent)) //not subnode but has children
200                 {
201                     //recursive call
202                     matches.addAll(getElementsByRegEx(n, restOfRegEx));
203                 }
204             }
205             n = it.nextNode();
206         }
207 
208         //### end matching part ###
209 
210         return matches;
211     }
212 }
213 
```

Bilaga 7 XMLFileFilter

XMLFileFilter.java

```
1  import java.io.File;
2  import javax.swing.filechooser.*;
3
4  /**
5  Class used to filter out files that does not have the file ending xml or xmi.
6  */
7  public class XMLFileFilter extends FileFilter
8  {
9      private final static String XMI = "xmi", XML = "xml";
10
11     //Accept all directories and all xml or xmi files.
12     public boolean accept(File f)
13     {
14         if (f.isDirectory())
15         {
16             return true;
17         }
18
19         String extension = this.getExtension(f);
20         if (extension != null)
21         {
22             if (extension.equals(XMI) || extension.equals(XML))
23                 return true;
24             else
25                 return false;
26
27         }
28             return false;
29     }
30
31     /** The description of this filter */
32     public String getDescription()
33     {
34         return "XML document (*.xml, *.xmi)";
35     }
36
37     /** Get the extension of the file. */
38     public static String getExtension(File f)
39     {
40         String ext = null;
41         String s = f.getName();
42         int i = s.lastIndexOf('.');
43         if (i > 0 && i < s.length() - 1)
44         {
45             ext = s.substring(i+1).toLowerCase();
46         }
47         return ext;
48     }
49 }
```