Avdelning för datavetenskap

Afrah Al-abuhalje

och

Sanaa Al-abuhalje

Översikt och tillämpning av XML

Overview and application of XML

Examensarbete 10 poäng

DAI

Datum/Termin: 07-06-05

Handledare: Robin Staxhammar

xaminator: Martin Blom

Ev. löpnummer: C2007:07

Karlstads universitet 651 88 Karlstad

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

Information@kau.se www.kau.se

Översikt och tillämpning av XML

Afrah Al-abuhalje och Sanaa Al-abuhalje

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

Afrah Al-abuhalje

Sanaa Al-abuhalje

Godkänd, 2007-06-05

Handledare: Robin Staxhammar

Examinator: Martin Blom

Sammanfattning

Allt eftersom kraven ökat på mer avancerade applikationer över Internet har det förekommit kritik mot att HTML inte räcker till, vilket är en av anledningarna till att ett nytt märkningsspråk som heter XML växt fram. I det här arbetet redogör vi för, alltifrån grunderna i XML till mer avancerade delar som DTD, XML-schema och XSLT.

XML kombinerar HTML:s enkelhet med SGML:s möjligheter. En av XML:s främsta styrkor är att det kan användas för att lagra all typ av data, utan att man behöver ta någon hänsyn till hur den senare ska presenteras. Innehåll och presentation är helt separerade. En annan viktig egenskap är att XML-dokument lagras som vanliga textfiler, vilket innebär att XML är system- och plattformsoberoende.

I denna uppsats utgår vi från två mål, dels att tillämpa XML för att skapa ett lämpligt lagringsformat för konfigurationsdata till en nätverksemulator, och dels att redogöra för XML. För att emulera förhållanden som finns i ett riktigt nätverk kan t ex bitfel, paketförluster, bandbreddsbegränsning och fördröjning emuleras. Inställningar av önskad frekvens av paketförluster, bitfel o s v är exempel på konfigurationsdata. Då vi redogör för XML återkopplar vi till, och beskriver tillämpningen stegvis. På så sätt får läsaren en god inblick i hur XML fungerar.

Abstract

HTML is not powerful enough to handle the increasing demands on Internet applications of today, which is one of the reasons to why a new markup language called XML has been introduced. In this report we describe the basics as well as more advanced parts of XML like DTD, XML Schema and XSLT.

XML combines the simplicity of HTML and the possibilties of SGML. One of the premier strengths of XML is that it may be used to store any type of data without any considerations to how it will be presented later. The content and the presentation are separated completely. Another important property is that XML documents are stored as ordinary text files, which means that XML is system and platform independent.

In this report our aim consists of two goals. One goal is to create a suitable format for storing configuration data using XML. The configuration data is to be used by a network emulator. In order to emulate the conditions present in a real network the emulator is capable of emulating things as bit errors, packet losses, bandwidth limitation and delay, all of which are examples of configuration data. The other goal is to describe the basics as well as more advanced parts of XML in general. As we describe XML, we continuously show how we apply this knowledge to our application. In this way, the reader achieves a good insight into how XML works.

Innehållsförteckning

1 Inledning

1.1 bakgrund

Vid datavetenskap på karlstads universitet utvecklades ett verktyg för emulering av nätverk som möjliggör att man genom att låta nätverkstrafik passera en dator kan emulera förhållanden som finns i det riktiga nätet eller över specifika länkar för att undersöka hur transportprotokoll och applikationer fungerar när det uppkommer fördröjning och paketförluster. Emulatorn kan hantera följande aspekter: bitfel, paketförluster, bandbreddsbegränsning och fördröjning. Den inkommande trafiken styrs av mönsterfiler, där det finns en mönsterfil för varje aspekt. Beroende på vilket scenario som skall emuleras så finns det behov av att flera aspekter skall emuleras samtidigt.

För att gruppera de mönsterfiler som hör ihop ska vi skapa scenariofiler, som är sammanslagna mönsterfiler. För att hålla reda på scenariofilerna är det tänkt att en central punkt ska spara själva scenariofilerna samt information om dem. Denna scenariofilsinformation ska hanteras med XML för att möjliggöra flexibel bearbetning av informationen och enkel presentation på webbsidor.

Själva iden till vår uppsats, att skriva om XML, fick vi när vi började läsa en kurs i datakommunikation, där läraren pratade om XML. Eftersom vi inte har kommit i kontakt med XML förut blev vi intresserade av ämnet och ville därför ta reda på mer om XML. När vi började forska i ämnet såg vi snabbt att det är ganska stort område och det som läraren tog upp vid föreläsningarna var bara toppen av ett isberg.

1.2 Mål

Uppsatsens mål är tvådelat. Dels ska vi utföra en tillämpningsuppgift och utifrån den förklarar vi syntaxen och semantiken av XML-kod, och dels ska vi presentera och exemplifiera den terminologi som förekommer i samband med XML för att klargöra centrala begrepp.

Det huvudsakliga målet med vår uppsats att färdigställa tillämpningsuppgiften som har följande krav:

1. Att ta fram ett förslag på ett XML-schema för dataformatet vi fick i uppgiften.
2. Att ta fram ett förslag på en XML-fil innehållande exempelinformation som följer begränsningarna man har i XML-schemat.
3. Att skapa en XSLT-fil som ger en översiktlig presentation av informationen i XML-filen på en webb-sida.
4. Att skapa en XSLT-fil som ger en detaljerad presentation av informationen i XML-filen på en webb-sida.

1.3 Avgränsningar

Den målgrupp som vi vänder oss till med denna uppsats är våra kollegor i dataingenjörsprogrammet.

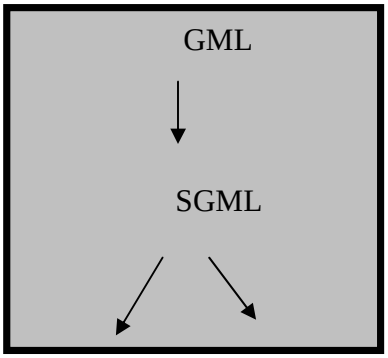
Vi gör inte några anspråk på att framställa en komplett lärobok i XML, utan vi vill ge en översiktlig bild av XML och dess beståndsdelar. Eftersom XML är så pass omfattande kan vi inte gå in i varje del på djupet.
Vi beskriver SGML och HTML endast översiktligt, eftersom de utgör endast en bakgrund till XML och ligger utanför syftet med denna uppsats.

1.4 Uppsatens upplägg

I kapitel 2 tar vi upp markeringsspråkens historia, från SGML på 80-talet, HTML på 90-talet och till XML som också började utvecklas i slutet av 90-talet .
Kapitel 3 beskriver tillämpningsuppgiften som är tänkt att ingå som en viktig del i uppsatsen. För att kunna tillämpa uppgiften har vi i kapitel 4, 5 och 6 tagit upp reglerna för och teknikerna med vilka man skapar XML-dokument. Vi beskriver de olika XML-teknikerna bland annat med hjälp av kodexempel, eftersom vi anser att det är ett bra sätt att tillgodogöra sig kunskap. Kapitel 4 visar hur man tar fram välutformade XML-dokument, det vill säga dokument som överensstämmer med XML:s grundläggande syntaktiska regler. Kapitel 5 avslöjar hur man skriver giltiga XML-dokument: dokument som inte enbart överensstämmer med de grundläggande syntaktiska reglerna utan också motsvarar en specifik dokumentstruktur som antingen definieras i själva dokumentet eller i en separat fil. I detta kapitel beskriver vi översiktligt vad en DTD är och hur man knyter en DTD till ett XML-dokument, vad XML-schema är och hur det fungerar och lite kortfattat vad RELAX-NG är. I kapitel 6 förklarar vi hur man visar ett XML-dokument i en webbläsare. Detta kapitel tar upp XSLT-tekniken som används för att inte enbart kunna formatera dokumentets innehåll utan även välja ut och ändra dess innehåll, vilket ger fullständig kontroll över vad som visas.
I kapitel 7 och 8 anger vi resultatet, rekommendationer och summeringen av uppsatsen.
I slutet har vi även en ordlista, vilken är ett stöd under läsningen av uppsatsen.

2 Översikt av XML

2.1 Bakgrunden till XML



Figur 2.1 XML:s evolution

2.1.1 SGML

Den utveckling som ledde fram till XML (eXtensible Markup Language) började redan på 1960-talet med uppkomsten av deskriptiv uppmärkning. Uppmärkning är ett sätt att berika en text med förklaringar och instruktioner. Man skiljer på processinriktad och deskriptiv uppmärkning. Processinriktad uppmärkning går ut på att beskriva vad som ska göras, medan deskriptiv uppmärkning handlar om att ange vad det är. [6]

En forskningsgrupp inom IBM, som leddes av Charles Goldfarb, introducerade möjligheten att definiera egna dokumenttyper. Resultatet kallade man Generalized Markup Language (GML). Mot slutet av 70-talet inbjöds Charles Goldfarb att arbeta med en ny standard för deskriptiv uppmärkning som ANSI (American National Standards Institute) höll på med. Arbetet kallades Standard Generalized Markup Language (SGML) och byggde i många avseenden på GML. Efter några år flyttades SGML över till ISO (International Standardization Organization), och blev en internationell ISO-standard (8879-1986) [6]. SGML är ett så kallat markup språk, det vill säga att det används för att definiera ett dokuments struktur med hjälp av märkord. Dessa märkord styr inte själva innehållet i ett dokument utan definierar dess struktur och hur det ska bearbetas.

SGML har utvecklats med intentionen att lösa ett antal konkreta problem, framförallt underlätta utbytet av information mellan olika datorer, olika plattformar, underlätta återanvändning av information, skapa någon form av mallar för att strukturera information samt ge förutsättningar för att skapa mer intelligenta informationslösningar. Återanvändning innebär inte enbart att redan skriven text återanvänds, det handlar även om att information skall kunna publiceras på olika medier utan alltför mycket arbete [11].

SGML användes först och främst i Unix-baserade system och var inte tänkt att utnyttjas för vanligt hemmabruk. Idag används SGML i större omfattning än vad som är allmänt känt. De främsta användningsområdena är stora tillämpningar med komplex information med lång livslängd, exempelvis lagring av information i vårdsektorn. SGML används bland annat inom telekomindustrin, fordonsindustrin, läkemedelsindustrin, försvarsindustrin och nyhetsbranschen med mera. Användandet av SGML innebär ofta att man anammar ett mer ingenjörsmässigt synsätt på informationen, det vill säga författaren av dokumenten behöver inte ägna sig åt layout utan koncentrera sig på innehållet [11].

Trots att SGML är en mycket omfattande och generell standard som klarar att uppfylla de flesta önskemål vad det gäller modularisering, strukturering och länkning av information, blev SGML aldrig den framgång man hoppades på, på grund av dess omfång och komplexitet. Standarden var omöjlig att implementera, på grund av de många valmöjligheterna och alternativa skrivsätten i syntaxen. Det i sin tur skapade dyra program och dyra konsulter som ledde till att bara ett fåtal mycket stora organisationer kunde kosta på sig att använda SGML [6].

2.1.2 HTML

Eftersom SGML inte är framtaget för att användas på Internet, utvecklades HTML (HyperText Markup Language), för att på ett enkelt sätt kunna göra informationen allmänt tillgänglig via Internet och oberoende av någon enskild leverantörs programvaror. Formellt sett är HTML en tillämpning av SGML. Eller med andra ord, HTML är en uppsättning taggar som följer SGML:s regler. Den uppsättning taggar som definieras av HTML är anpassad efter strukturen för hypertextdokument [7]. De flesta dokumenten på webben är kodade i HTML. För att läsa dessa använder man en webbläsare, t.ex. Netscape eller Internet Explorer [11]. HTML är det filformat som används för att lagra webbsidor. HTML har en fix uppsättning fördefinierade element med

vilka man kan hantera komponenterna som ingår i en normal webbsida. Rubriker, stycken, listor, tabeller, bilder och hyperlänkar är exempel på element [8].
Användningen av HTML har lett till att man som läsare och användare kan ta del av en mängd information oavsett var den publiceras i världen, vilken webbläsare man använder och vilken dator man har [11]. Trots att HTML ännu är det vanligaste språket för att skapa webbsidor har det begränsad kapacitet att lagra information. HTML har bara tillgång till en begränsad mängd fördefinierade märkord. Dessa märkord räcker inte alltid till för att skapa mer avancerade presentationer och tillämpningar.
Att utöka antalet märkord i HTML är inte möjligt eftersom varje verksamhet behöver sina egna märkord.
Många leverantörer av HTML-verktyg har utökat HTML med sina egna märkord för att överbrygga begränsningar och skapa nya möjligheter att presentera information. Dessa ansträngningar har lett till viss inkompatibilitet mellan olika HTML-verktyg. Resultatet av detta syns på HTML-sidor med ikoner innehållande texter som ”Denna sida visas bäst i Internet Explorer” eller ”Denna sida visas bäst i Netscape”. Skaparen av en sådan HTML-sida har valt att använda märkord som endast stöds av en viss leverantör [11].

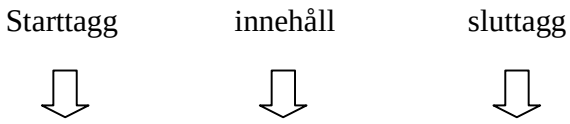
Exempel på de dokument som inte går att beskriva särskilt väl med HTML:

1 Ett dokument som inte innehåller de normala komponenterna (rubriker, stycken, listor, tabeller osv.). HTML saknar exempelvis märkord som behövs för att koda ett partitur eller en uppsättning matematiska ekvationer.

2 Ett dokument som man vill ordna i en trädliknande hierarkisk struktur. Exempelvis om man skriver en bok och vill avbalka den i delar, kapitel, avsnitt A, B, C och så vidare.

3 En databas, till exempel ett bibliotek. Man skulle kunna använda en HTML-sida för att lagra och visa oföränderlig databasinformation (till exempel en lista med beskrivning av böcker), men om man vill sortera, filtrera, söka rätt på och arbeta med informationen på andra sätt måste alla data ettikeras. HTML saknar de nödvändiga märkorden för att åstadkomma detta. [8]

Ett annat problem med HTML var att man kunde slarva med koden. Det kunde vara element som inte avslutades korrekt, element vars attribut inte skrevs till fullo osv. Med element menar vi den grundläggande byggstenen som används för att bygga XML- eller HTML-dokument. Det består vanligtvis av kombinationen starttagg, innehåll och sluttagg. Innehållet i ett element är allt som finns mellan start- och sluttaggen. Så här kan ett komplett element med startagg, sluttagg och innehåll se ut:



<titel> skriv med XML eller HTML </titel>

HTML:s struktur är mycket bristfällig. Det innebär att man mycket väl kan skriva syntaktiskt ”korrekt” HTML som är strukturellt sett är ganska egendomlig. Innehållet i HTML-märkordet <BODY> har definierats så att alla märkord som får användas kan placeras i vilken ordning som helst. Ett exempel på syntaktiskt korrekt HTML är:

```
<BODY>

<P> Denna uppsats beskriver XML </P>
<H3> Inledning </H3>
<H1> Avslutning </H1>

</BODY>
```

Exempel 2.1 Syntaktiskt korrekt HTML

En paragraf <P> kan mycket väl förekomma utan någon inledande rubrik i HTML. Likaväl kan en rubrik på nivå 3 <H3> uppträde före en rubrik på nivå 1 <H1>. HTML:s avsaknad av struktur leder också till att användaren inte kan få stöd i en HTML-editor angående vilka märkord som får infogas, HTML-editorn kan inte avgöra (utifrån HTML-specifikationen) vilka element som får förekomma i vilken ordning [11]. Problemet har inte med editorn att göra utan snarare att allt är tillåtet.

Under slutet av 90-talet insåg man också att webbsidor inte för alltid skulle kunna gå att göra med HTML. Man hade även ett önskemål att kunna göra webbsidor både för bildskärmar på datorn, handdatorn, mobiltelefonen och andra ställen [5].

2.1.3 XML, lösningen på problemen!

Uppenbarligen finns ett glapp mellan den omfattande och komplexa SGML-standarden och den enkla och fattiga HTML-standarden. XML utvecklades för att fylla detta tomrum, och är en lösning som försöker förena det bästa från dessa två världar.
XML utvecklades som ett universellt standardiserat filformat. Ett format som är helt oberoende av hård- och mjukvara, oberoende av skriftspråk, ja till och med oberoende av själva informationen. Ett format för alla slags branschspecifika data, webbsidor, sekundsnabba transaktioner (t ex banktransaktioner), bilder och långa textdokument. Framförallt handlar det inte enbart om att presentera data, utan om att göra den tillgänglig för automatisk bearbetning oavsett var den finns [5]. XML är en delmängd av SGML som särskilt utvidgats för användning på Internet. Man har uteslutit det som i SGML varit komplext och svårt att förstå och att bygga programvaror för. Det viktigaste hos SGML, flexibiliteten, det vill säga att själva kunna bestämma vilken uppsättning märkord man vill använda och hur man vill strukturera information finns kvar i XML [11]. När man skriver ett XML-dokument kan man istället för att begagna sig av en begränsad uppsättning fördefinierade element, som man gör med HTML, skapa egna och ge dem godtyckliga namn. Tack vare detta, som ordet extensible (utbyggbar) ingår i Extensible Markup Language [8].

XML började utvecklas redan under 1996 av XML working group (ursprungligen känd som the SGML Editorial Review Board) som bildades under överinseende av World Wide Web Consortium (W3C), och släpptes också i ett antal betaversioner. Ordförande var Jon Basak, Sun microsystems, med aktivt deltagande av en XML Special Interest Group (tidigare känd som the SGML Working Group) som också hade bildats av W3C. Gruppen beskriver språket så här: [8] citat:

”XML (extensible markup language) är en delmängd av SGML…Målet med språket är att göra det möjligt att erbjuda, ta emot och bearbeta generisk SGML på Internet på samma sätt som det idag är möjligt att arbeta med HTML. XML har utformats så att det ska vara lätt att implementera och fungera ihop både med SGML och HTML”. [8] Slut på citat

Den första standarden, 1.0 antogs i februari 1998 [12]. Trots namnet är XML inget märkspråk utan ett regelverk för att skapa märkspråk, ett så kallat metaspråk.
XML är väldigt flexibelt, det kan användas för att kommunicera med ett existerande datorsystem eller program. Detta innebär att företag inte behöver välja mellan olika system. XML kan inte bara integrera system, utan i vissa fall till och med ersätta dem. Med XML blir det relativt enkelt för ett företag att byta miljö, eftersom samma XML-dokument kan skapas, sparas och läsas på olika operativsystem med olika programvaror från olika tillverkare [10].

Istället för att använda ASCII-kod, kan XML använda ISO-standaren ISO 10646 unicode, vilken är en internationell teckenstandard som använder 31 binära siffror och som gör det möjligt att representera alla världens språk, t.ex. kinesiska och arabiska.
XML:s syntax är inte lika rik på funktioner och alternativ som SGML, vilket innebär att det är enkelt både för människa och maskin att läsa och skriva XML-dokument [8]. En maskin kräver en väl definierad struktur för att kunna hantera informationen vi skapar och det är just det som XML ger [11].

2.2 De officiella målen med XML enligt W3C

1. ”XML skall vara lätt att använda över Internet”
   XML utformades huvudsakligen för att lagra och överföra information på Internet, och att ge stöd åt distribuerade program på Internet.

2. ”XML skall stödja en stor mängd av applikationer”
   XML utformades även så att program som inte kommunicerar via Internet kan utnyttja tekniken, till exempel program med vars hjälp man skapar dokument och filtrerar, översätter eller formaterar information.

3. ”XML skall vara kompatibelt med SGML”

Mycket av drivkraften bakom XML har varit att ge den stora marknaden av SGML-användare ett format som lämpar sig för överföring av högt strukturerade dokument via webben samtidigt som det skall vara kompatibelt med befintliga format .Bland annat var tanken att befintliga programvaror för SGML skall kunna läsa och skriva XML.

4. ”Det skall vara enkelt att skriva program som behandlar XML-dokument”

Om ett markeringsspråk avsett för webben ska vara praktiskt och godtas av alla måste det vara lätt att utveckla webbläsare och andra program som bearbetar dokumenten.

5. ”Antalet valfria inslag i XML skall hållas till ett absolut minimum, helst ska det inte finnas några alls”

SGML har hundratals valfria inslag, vilket är en stor bidragande orsak till språkets komplexitet. Exempel på valfria inslag i SGML är bland annat att det är möjligt att omdefiniera de avgränsade tecknen, normalt < och >, och att det är tillåtet att utlämna sluttaggen i de fall tolken kan räkna ut var elementet slutar. I XML finns inga egentliga valfria inslag.

6. ”XML-dokument skall vara läsbara och rimligt lättbegripliga”

Det skall vara möjligt att tyda ett XML-dokument - även om man inte har tillgång till en viss webbläsare - bara genom att öppna filen i en vanlig text-editor.

Det är lätt för en vanlig människa att läsa ett XML-dokument, eftersom det är skrivet i klartext och har en logisk, trädliknande struktur. Koden kan göras ännu mer lättläst genom att författaren väljer begripliga namn på dokumentets element, attribut och entiteter (vi förklarar vad element, attribut, entiteter är i kapitel 4).

7. ”XML:s utformning bör gå fort”

XML-standarden kan naturligvis överleva endast om programmerarsamfundet och användarna accepterar den. Därför behövde standarden bli klar innan båda dessa grupper började göra bruk av alternativa standarder, som mjukvaruföretag har en tendens att framställa med hög frekvens.

8. ”XML:s design ska vara formell och kortfattad ”

XML-specifikationen är, liksom HTML, skriven i något som kallas Extended Backus-Naur Form (EBNF). Det här formatet kan vara svårläst men löser oklarheter och gör det i slutändan lättare att skriva XML-dokument och i synnerhet XML-verktyg. Man behöver dock inte kunna EBNF för att kunna XML.

9. ”XML-kod skall vara lätt att skapa”

Det enda man behöver för att skriva XML-dokument är en vanlig text-editor.

10. ”XML-kod behöver absolut inte vara kortfattad”

Enligt mål 6 är tydlighet viktigare än koncishet. I HTML och SGML har man använt sig av förenklingar för att minska antalet tecken som man var tvungen att mata in för hand. Detta har lagt större tyngd på det program som ska behandla koden samtidigt som den blir svårare att tyda för det mänskliga ögat [8].

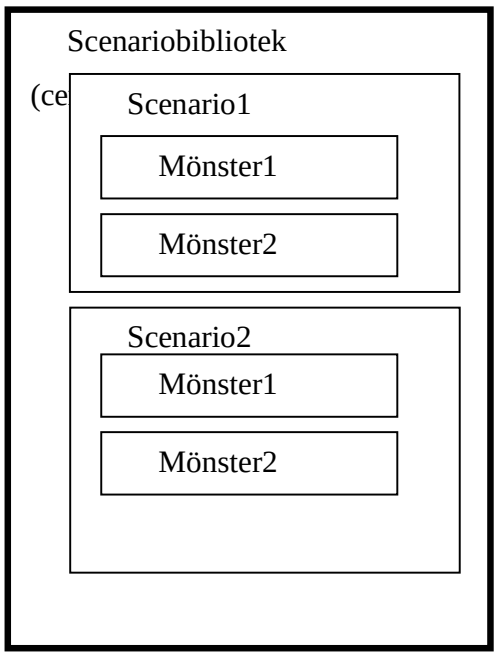
3 Tillämpningsuppgiften

Detta kapitel beskriver tillämpningsuppgiften som är tänkt att ingå som en huvuddel i vårt examensarbete. Uppgiften illustrerar hur XML kan användas för att lagra information på ett strukturerat sätt så att informationen senare kan bearbetas på olika sätt och även presenteras på webbsidor.

3.1 Bakgrunden till uppgiften

Vid datavetenskap utvecklades ett verktyg för emulering av nätverk som möjliggör att man genom att låta nätverkstrafik passera en dator kan emulera förhållanden som finns i det riktiga nätet eller över specifika länkar för att undersöka hur transportprotokoll och applikationer fungerar när det uppkommer fördröjning och paketförluster. Emulatorn kan hantera följande aspekter: bitfel, paketförluster, bandbreddsbegränsning och fördröjning. Den inkommande trafiken styrs av mönsterfiler, där det finns en mönsterfil för varje aspekt. Beroende på vilket scenario som skall emuleras så finns det behov av flera aspekter skall emuleras samtidigt.

För att hålla ihop de mönsterfiler som hör ihop skall vi skapa scenariofiler, som är sammanslagna mönsterfiler. För att hålla reda på scenariofilerna är det tänkt att en central punkt skall spara själva scenariofilerna samt information om scenariofilerns. Denna scenariofilsinformation skall hanteras med XML för att möjliggöra flexibel bearbetning av informationen och enkel presentation på webbsidor.



Figur 3.1 Central punktens mönster

3.2 Dataformat

Tabellen nedan ger en beskrivning av det data (information) som skall kunna lagras i XML-filen.

Information	**Typ**	**Beskrivning**
Scenario id	Heltal	Scenarioidentifikation som får endast innehålla siffror.
Descriptive name	Textsträng	Namn som beskriver scenariot. Får endast innehålla 60 tecken.
Description	Textsträng	Ger detaljerad beskrivning om vad scenariot emulerar. Får endast innehålla 1000 tecken.
Source	Textsträng	Ger källan för scenariofilerna och de ingående mönsterfilerna. Innehåller endast en av tre värden: trace, analytical eller simulation.

Providing organization	Textsträng	Ger namnet på organisationen som Tillhandahåller filerna. Får endast innehålla 60 tecken.
Providing person	Textsträng	Namnet på personen som tillhandahåller filerna. Maximalt 60 tecken.
Contact email	Textsträng	Email-adressen för den person som får kontaktas. Maximalt 60 tecken.
Number of patterns	Heltal	Antal mönsterfiler som ingår i scenariofilen. Maximalt 99 stycken
Pattern type	Textsträng	Ger mönstertyp.
Pattern mode	Textsträng	Ger metoden av mönstret.
Pattern size	Textsträng	Ger storleken på den valda ”Pattern mode” enheten.
Pattern description	Textsträng	Ger kort beskrivning av mönstret.

Tabell 3.1 De information som ska finnas med i XML-filen

3.3 Uppgiften i olika steg

Uppgiften kan delas upp i ett antal steg där de tre första är nödvändiga för att få en grundläggande funktionalitet.

1 Ta fram ett förslag på ett XML-schema för dataformatet enligt ovan.

För att kunna lösa denna del av uppgiften har vi i kapitel 5 beskrivit XML-schema. Beskrivningen i tabellen ovan och det vi går igenom i kapitel 5 hjälper oss att ge ett förslag på ett XML-schema som är lämpligt för just denna uppgift. I sista avsnittet i kapitel 5 finns ett förslag på hur ett XML-schema kan se ut.

2 Ta fram en giltig XML-fil innehållande exempelinformation.

Med giltig menas att XML-filen skall följa XML-schemat vi har skapat ovan. Följande är ett exempel på information som kan finnas med i XML-filen:

Scenario ID: 12312349

Descriptive Name: WLAN Handover - 1

Description:
This scenario emulates the IP -level conditions seen by a single WLAN user as he moves between two access points located in at ENSICA, Toulose. 802.11b is used, and there were no other active traffic during the measurements. This scenario is part of a larger collection with scenario files for the same conditions.

Source: Trace

Providing org: ENSICA

Providing person: Tanguy Perennou

Contact Email: tanguy.perennou@gmail.com

Number of patterns: 2

Pattern type: Loss
Pattern mode: Time
Pattern size: 200 seconds
Pattern info: 5 losses

Pattern type: Bandwidth change
Pattern mode: Time
Pattern size: 200 seconds
Pattern info: 154 bandwidth changes

Denna del av uppgiften löses efter att vi beskriver XML-syntaxen i kapitel 4 och efter att vi har gjort ett XML-schema. I sista avsnittet i kapitel 5 skall vi ger vi ett exempel på hur XML-filen kan se ut.

3 Skapa en XSLT-fil som ger en översiktlig presentation av informationen i XML-filen i en webbläsare.

Exempel på hur det kan se ut i webbläsaren:

Senario-id:1123456 **Descriptive name**: ......
Source: Trace **Nr of patterns**:3
Description: ....
Providing organization: ....
Providing person: ....
Contact email: ....

4 Skapa en XSLT-fil som ger en detaljerad presentation av informationen i XML-filen i en webbläsare.

Utseendet i webbläsaren är samma som ovan plus patternbeskrivningarna:

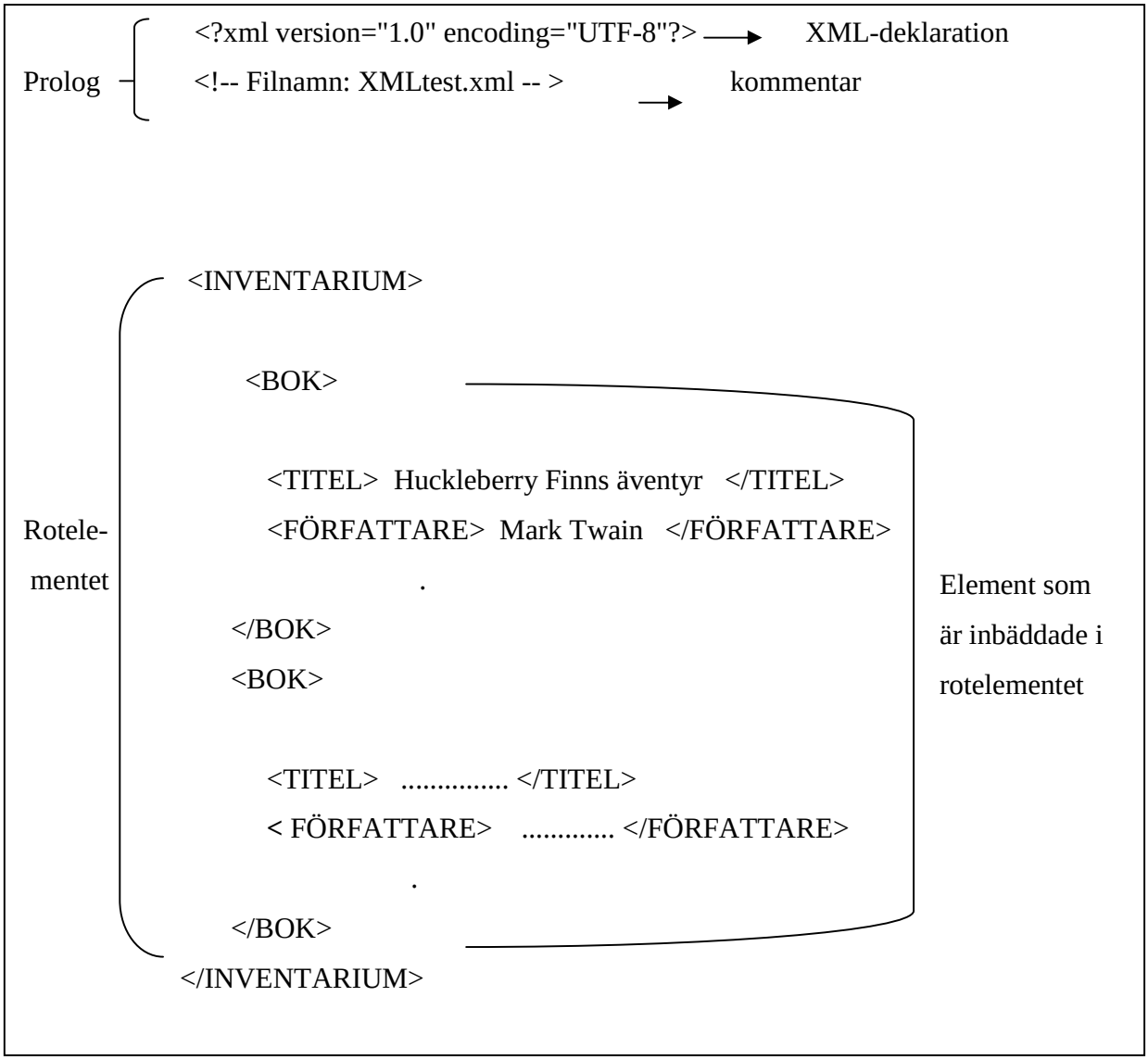
Pattern type: ….
Pattern mode: ….
Pattern size: ….
Pattern description: ….

För att kunna lösa punkt 3 och 4 förklarar vi i kapitel 6 hur man skapar en XSLT-fil. XSLT-filen kopplas till XML-dokumentet och instruerar webbläsaren hur XML-informationen ska formateras. Med hjälp av den kan XML-dokumentet öppnas direkt i webbläsaren. Utifrån förklaringen i kapitel 6 kommer vi att ge ett förslag på hur XSLT-filerna för punkt 3 och 4 kan se ut. Detta kan man se i sista avsnittet i kapitel 6.

4 XML-syntax

4.1 XML-dokuments uppbyggnad

Ett XML-dokument består av två huvudsakliga delar: prologen och dokumentelementet eller som det ofta kallas rotelementet. [8]



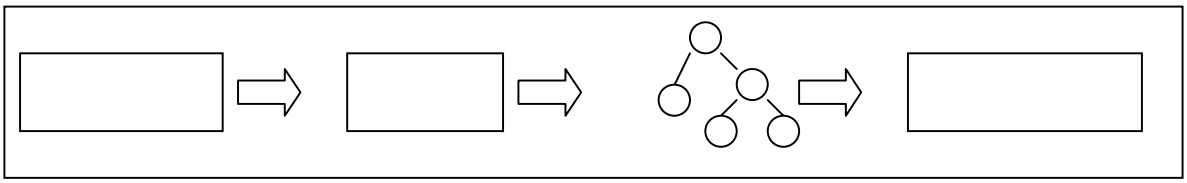
Figur 4.1 XML-dokuments översikt

Prologen

Först i prologen (se figur 4.1) hittar man XML-deklarationen:

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML-deklarationen förklarar att det här är ett XML-dokument och berättar för tolken vilken version av XML man använder. [6]. En tolk (eng. parser) är ett program som kontrollerar och översätter XML-dokument, så att innehållet kan hanteras av applikationen, det program som står för databehandlingen. Alla applikationer som använder XML har en inbyggd tolk som kontrollerar syntaxen.



Figur 4.2 Tolken kontrollerar och översätter dokumentet, så att innehållet kan hanteras av applikationen

Tolken kan även kontrollera att XML-dokument överensstämmer mot schemat (om det finns något). Det kan kontrollera

- att alla element och attribut som används är deklarerade
- att alla element och attribut som inte används får utelämnas
- att element som används flera gånger får användas flera gånger
- att element bara innehåller de element och attribut de får innehålla
- att elementen används i rätt ordning

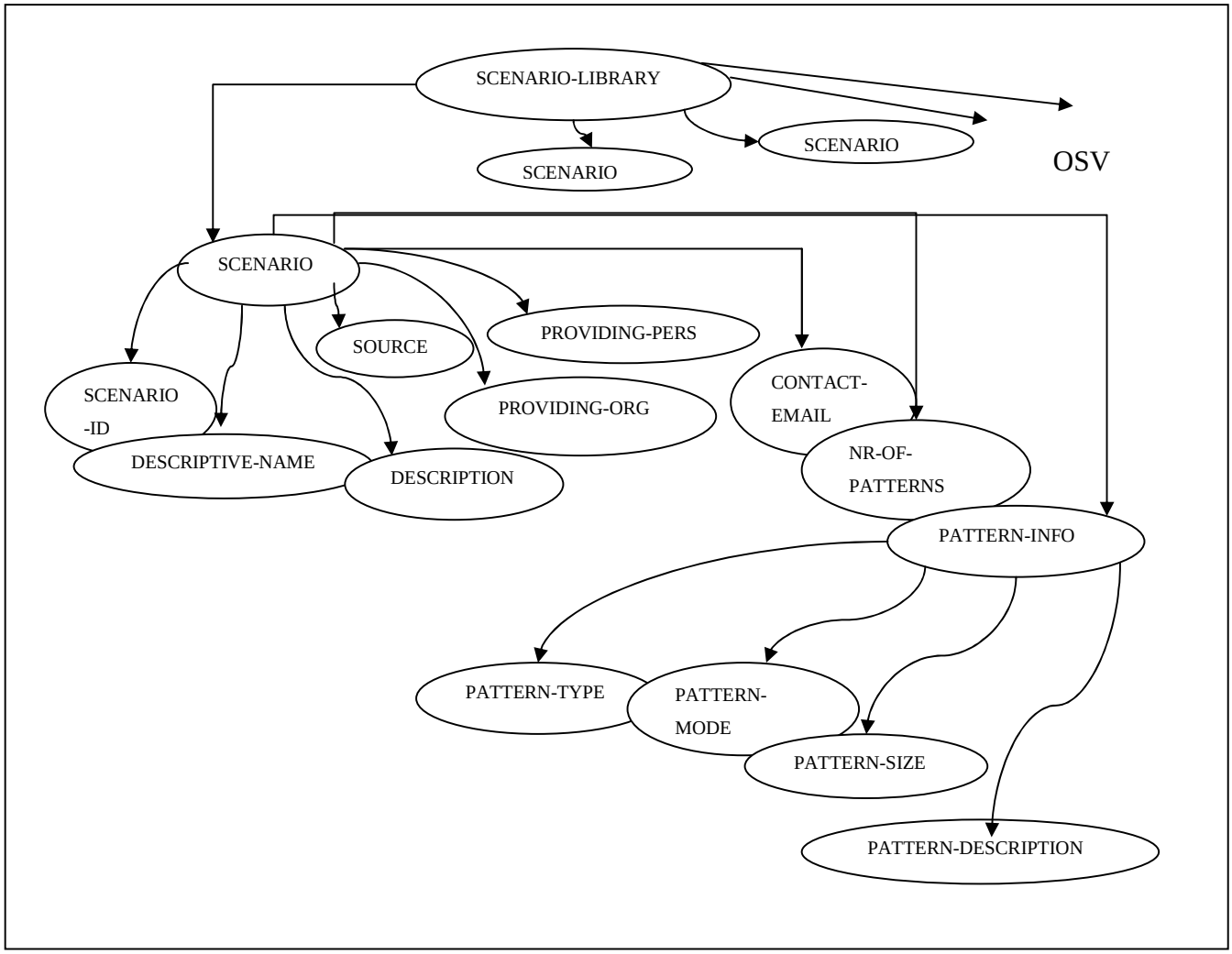
Att kontrollera att ett dokument följer schemat kallas för validering, och man skiljer därför på icke-validerande och validerande tolkar. [6]

Versionnumret i XML-deklarationen ska omges av antingen apostrofer eller citationstecken. Normaluppsättningen är UTF-8, som är komprimerad unicode med variabel bitlängd som tar betydligt mindre plats än ren Unicode. ISO-88591-1 rekommenderas om man vill använda svenska eller andra västereuropiska tecken. [4]. XML-deklarationen är valfri, men

specifikationen anger att den bör vara med. Finns det en XML-deklaration måste den stå först i dokumentet.
Prologen kan även innehålla en referens till schemat (schemat beskriver strukturen på XML-dokument och används för att validera dokument mot deras modeller, se kapitel 5 för mer information), så att en validerande tolk kan hitta det, och en eller flera processinstruktioner som innehåller information som tolken skickar vidare till applikationer (program som behandlar XML-dokument), [8]. Processinstruktioner liknar i många avseenden kommentarer.
Syntaxmässigt börjar en processinstruktion med <? och slutar med?>. Däremellan finns det alltid ett namn, som kallas mål (eng. target), eftersom den identifierar den applikation eller del av applikation som ska utföra något. Kommentarer i XML har samma syfte som kommentarer i andra programspråk, här kan människor som läser eller skriver koden förmedla information till varandra. [6]. Kommentarer börjar med tecknen <!-- och slutar med kombinationen -->. Det enda som inte får förekomma mellan dessa ändpunkter är --, det vill säga två bindstreck i följd. Tolken bryr sig inte om kommentarerna, men de kan vara värdefull information för den som läser eller skriver koden [7].

4.1.1 Element

Elementen är grunden i ett XML-dokument. Ett element består endera av en ensam tomelementstagg , det vill säga ett element som saknar innehåll, eller av kombinationen startagg, innehåll och sluttagg. Starttaggen består av namnet inom vinkelparenteser, <title>, och sluttaggen består av samma namn med vinkelparenteser men med inledande snedstreck, </title>. Innehållet i ett element är allt som finns mellan start- och sluttaggen. [15]. Det vanligaste innehållet i ett element är teckendata, det vill säga text som uttrycker elementets informationsinnehåll, och andra element, så kallade barn. Element kan också innehålla kommentarer, processinstruktioner, entiteter och CDATA-sektioner [6]. Entiteter och CDATA-sektioner förklaras lite senare i kapitlet.
Elementen är ordnade i en trädliknande hierarki med element som är inbäddade i andra element. Se figuren på nästa sida.



Figur 4.3 Elementen i trädliknande hieararki

Figuren visar ett exempel på ett sådant träd, namnen på alla förekommande element som vi har använt i trädet har vi tagit från vår tillämpningsuppgift. Av strukturen framgår att elementen SCENARIO-LIBARARY, SCENARIO och PATTERN-INFO, vardera representerar eller pekar på en understruktur. Var och en av dessa understrukturer beskrivs med sin uppsättning av ingående element.

XML kräver att element är korrekt nästlade. Om definitionen av ett element inleds i ett visst element måste det också avslutas i samma element. Elementen får aldrig överlappa varandra, vilket ibland är möjligt i HTML. Följande kod är ogiltig i XML-dokument men är tillåten i HTML: [8]

<h1> Här är min **** Rubrik **</h1>** **<p>** med **** **(FEL! Borde finnas innan man**

avslutar h1) en paragraf ** **

Detta skulle istället ha skrivits som:
<h1> Här är min Rubrik </h1>   med   en paragraf

4.1.2 Rotelementet

Alla XML-dokument måste ha exakt ett rotelement som måste stå först efter prologen och som innehåller alla andra element som kan finnas med i dokumentet. Exempel på ett felaktigt XML-dokument:

```
<? xml version="1.0" encoding="UTF-8?>
<!-- saknar ett rotelement  som kan vara t.ex. INVENTARIUM   -->
<BOK>     ……            </ BOK>
```

Exempel 4.1 Felaktig XML-dokument

Rotelemenetet i ett XML-dokument liknar elementet <BODY> i HTML-dokument, förutom att det kan heta vad som helst, bara namnet är giltigt. Här följer regler på giltiga namn [6]:

- Namn får inte börja med en siffra, det måste inledas med en bokstav eller ett understrykningstecken följt av noll eller flera bokstäver, siffror, punkter eller bindstreck. Bokstäver och siffror på alla språk är tillåtna.
- Namnens längd är obegränsad.
- Tomrumstecken och övriga skiljetecken är förbjudna.
- Undvik namn som inleds med XML, eftersom alla namn som börjar på XML är reserverade för själva XML-specifikationen.

Några exempel på giltiga elementnamn:

- Födelseår
- Партія
- Xsl:call-template
- dateTime.iso8601

Exempel på några ogiltiga elementnamn:

- 123
- för namn
- tom&jerry

XML skiljer på versaler och gemener, så födelseår, FödelseÅr och FÖDELSEÅR är tre olika namn.

4.1.3 Tomma element

Element som saknar innehåll kallas för tomma element. Tomma element har sluttaggen direkt efter starttaggen, som i det här exemplet:

<BOK> </BOK>
Istället för att använda en sluttagg kan man använda sig av en tomelementtagg. Detta görs genom att lägga till ett snedstreck innan den stängande vinkelparentesen i en starttagg:

<BOK/>

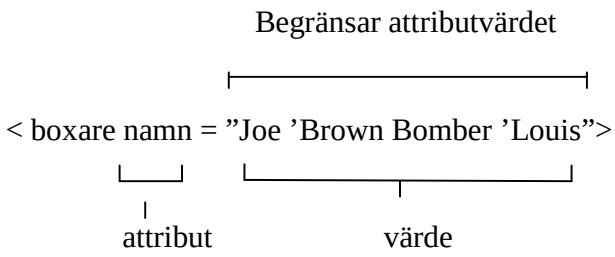
Bägge notationerna betyder samma sak. [6]

Det finns två anledningar till att använda tomma element:

- För att tala om för XML-programmet att det ska utföra en åtgärd eller visa ett objekt. Ett exempel från HTML är det tomma elementet BR, som instruerar webbläsaren att infoga en radbrytning.
- För att ha ett attribut i det. Ett HTML-exempel är det tomma elementet IMG, som har attribut som talar om för tolken var bilden kan hittas och hur den ska visas. [8]

4.1.4 Attribut

Förutom innehållet i ett element kan ett element även ha attribut. Attribut är en alternativ metod att inkludera information i ett element [15]. Man kan lägga en eller flera attributspecifikationer i ett elements starttagg eller i ett tomt element. Attributspecifikationen är ett namn/värdpar som förknippas med elementet. Attribut namnges efter samma regler som element. Namnet på attributet skiljs från värdet med ett likhetstecken. [7]
Attributvärden avgränsas med enkla eller dubbla citationstecken, vilket man använder spelar ingen roll, förutom att det måste matcha varandra. Det finns särskilda regler om man vill sätta in enkla eller dubbla citationstecken i ett attributvärde. Det enkla sättet är att avgränsa attributvärdet med det tecken som inte förekommer i värdet, det vill säga om man använder ett enkelt citationstecken i värdet då bör ett dubbelt citationstecken användas till attributvärdet:



< boxare namn = ’Joe ”Brown Bomber” Louis’>
[6]
I vissa situationer är det olämpligt att använda attribut eftersom det har flera begränsningar: [6]

- Attribut kan inte innehålla element eller andra attribut. Man bör använda element om informationen är strukturerad. Till exempel anta att en författares namn består av två delar: förnamn och efternamn, det vill säga informationen är strukturerad på detta sätt:
  Författare:
  Förnamn
  Efternamn
  Då är det lämpligt att använda FÖRFATTARE som ett element och FÖRNAMN och EFTERNAMN som underordnade element. Fördelen med att använda element i det här fallet är att ange struktur som gör det enkelt att förstå informationen.

- Samma attributnamn får bara förekomma en gång i en tagg. Man bör använda element om det finns fler av samma sak. Ett bra exempel är en telefonlista. Om attribut väljs är det tvunget att lista alla tänkbara varianter som skulle kunna vilja lagras, exempelvis hemnummer, mobiltelefonnummer och nummer till arbetsplatser. Istället kan det i DTD:n eller XML-schemat anges att elementet får användas valfritt antal gånger och sedan knyts ett attribut eller ett barnelement till det som specificerar av vilken typ det är. Fördelen med det att man slipper att skriva i onödan.
- Ordningen på attribut har aldrig någon betydelse. Element bör användas om så är fallet. Till exempel om vi har två olika attributnamn i samma tagg då spelar det ingen roll vilken vi har först, men om vi har dem som element då spelar det en väldigt stor roll i vilken ordning de står i, för de förekommer i den angivna ordningen.

Det finns fördefinierade attribut, som man känner igen på prefixet XML:. Man måste inte använda dem – man kan använda sin egen uppmärkning istället. XML: lang är ett exempel på ett fördefinierat attribut, som används då ett dokument är skrivet på flera språk så kan det vara bra att märka upp vilka delar som är skrivna på vilka språk. [3]

Exempel på användning av attribut [5]:

- För att ange versionsnummer för en rapport t.ex. <rapport ver=”1.2”>
- För att ange dokuments eller element status t.ex. <memo status = ”utkast”>
- För att identifiera text som skall genereras av systemet t.ex. numrering av lista t.ex. < lista typ=”Heltal”>
- För att sätta en enhet på ett elements innehåll, t.ex. valuta för en prisuppgift t.ex. <pris valuta = ”SEK”> 599 </pris>

4.1.5 Entiteter

En entitet i XML är en bit data som kan representeras med ett alternativt namn till en information som kan vara långt och upprepas flera gånger. Som ett makro inom programmering [15]. Ett

makro är en samling instruktioner, dvs. ett annat namn för procedur eller funktion, som används för att automatisera ofta återkommande och/eller repetitiva arbetsmoment i datorprogram. Entiteter sparar tid, minskar storleken på XML-dokumenten och gör det möjligt att bygga dokument i modulform. Syntaxen för entiteter liknar den som används för att deklarera element och attribut i giltiga XML-dokument. [8]
Entiteter skrivs i dokumentet via *entitetsreferenser*. En entitetsreferens är namnet på entiteten mellan ett et-tecken (&) och ett semikolon. När XML-tolken hittar anropet till entitetsreferensen ersätter tolken den med dess värde. Om vi antar att vi har definierat entiteten ”kungen” med värdet ” Carl XVI Gustaf”>:

<! ENTITY kungen ”Carl XVI Gustaf”>

Sedan anropar vi entiteten med hjälp av entitetsreferensen:

<Stycke> kung & kungen; hälsade gästerna välkomna </Stycke>

Tolken kommer att ersätta entitetsreferensen på det här sättet:

<Stycke> kung *Carl XVI Gustaf* hälsade gästerna välkomna </Stycke>. [6]

Det finns 5 stycken fördefinierade entitetsreferenser i XML. De listas i följande tabell och kan infogas där det inte är tillåtet att infoga tecknet direkt. [8]

Fördefinierad entitetsreferens	**Infogat tecken**
& amp;	&
<	<
>	>
'	,
"	“

Tebell 4.1 Fördefinierade entitetsreferenser

Exemplet på användning är då et-tecknet behöver användas i elements innehåll:
<Company> Marks & Spencer </Company>
Det är inte tillåtet att använda et-tecken i elements innehåll, då måste et-tecknet bryts med en & -entitet:
<Company> Marks & Spencer </Company>. [7]

XML stödjer även *teckenreferenser* som används till att peka på ett Unicode- tecken genom att ange dess teckennummer. Teckenreferenser som börjar med &#X använder en hexadecimal representation av koden, som är det vanligaste sättet. Teckenreferenser som börjar med &# använder en decimal representation av koden. Teckenreferenser används i första hand för specialtecken som inte finns på tangentbordet, som √ och ¾. Man kan också använda dem till tecken på främmande språk. [6]

4.1.6 CDATA-sektioner

Som vi har sett måste märkningsskiljetecken, så som mindre-än-tecken och et-tecken, som står i ett elements innehåll ersättas med en entitet. Ibland behöver man infoga många < eller & tecken, exempelvis i matematiska ekvationer, då kan det vara svårt att ersätta alla dessa märkningstecken. Dessutom är det svårt att inkludera ett XML-dokument i ett XML-dokument. CDATA- sektioner utvecklades för just dessa fall.
Här följer ett exempel på en CDATA-sektion som används för att länka in ett XML-dokument i ett annat XML-dokument:

```
  <? Xml version=”1.0”>
  <exempel>
  <![ CDATA[
  <?xml version =”1.0”?>
<entry>
   <name> John Doe </name>
   <email href =mailto:John@emailaholic.com/>
</entry>]]>
  </exempel>
```

Exempel 4.2 CDATA-sektion används för att inkuldera ett XML-dokument i ett XMl-dokument

I exemplet ovan ser vi att CDATA-sektioner börjar med strängen <!CDATA[och avslutas med teckenkombinationen]]>. Alla tecken är tillåtna inom dessa avgränsade teckengrupper förutom strängen]]>, vilket innebär att det inte går att inkludera en CDATA sektion i en annan CDATA-sektion. [7]

4.2 Välutformade dokument (Well-formed document)

Ett välutformat XML-dokument följer en minsta uppsättning regler, det vill säga de regler som vi nämner här nedan, som gör att det kan bearbetas av en webbläsare eller ett annat XML-program.[8]

Regler för välutformade dokument

- Dokumentet måste ha exakt ett rotelement från vilket alla andra element härstammar.
- XML åtskiljer stora bokstäver från små, tillexempel <NAMN> och <namn> är starttagg för två olika element.
- Elements namn i starttaggen måste vara exakt detsamma som namnet i motsvarande sluttagg.
- I XML måste varje öppnat element, inklusive tomma element stängas med korresponderande sluttagg.
- Elementen måste vara korrekt nästlade. Det betyder att ett element som inleds inom ett element måste också avslutas inom detsamma.

  Rätt

   <front> </front>

  Fel

   <front> </front>
- Inget attribut får förekomma mer än en gång i en tagg. Alla attribut måste tilldelas ett attributvärde. [14]

Om ett dokument inte är välutformat är det inte ett XML-dokument.

4.3 Giltiga dokument (vaild document)

Ett giltigt XML-dokument är ett välutformat dokument som uppfyller minst ett av följande två krav:

- En korrekt dokumenttypsdeklaration måste finnas i prologen som hänvisar till en DTD. Resten av dokumentet måste överensstämma med innehållet och strukturen som definieras i DTD:n
- Dokumentet följer reglerna för dokumentets innehåll och uppbyggnad som definierats i ett XML-schema.

[8]

Se kapitel 5 för mer information om DTD och XML-schema.

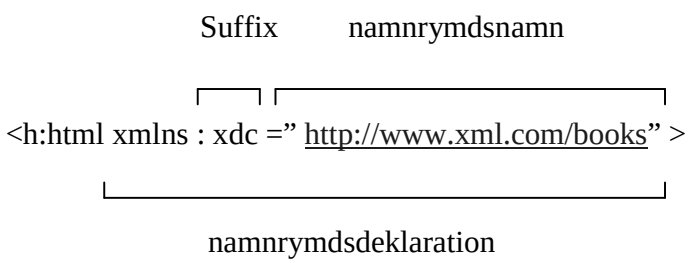
4.4 Namnrymder

Namnrymder är en ny teknik som W3C tagit fram som stöd för XML. XML är utbyggbart som vi har nämnt tidigare. Utbyggbarheten måste underhållas i en nätverksmiljö, som till exempel webben, för att undvika konflikter. Namnrymder är då en lösning som hjälper till att hantera XML:s utbyggbarhet. [7]

XML- data som kombineras från flera källor kan innebära krockar mellan elementens och attributens namn. Betrakta exempelvis att vi har två separata XML-dokument, och de båda dokumenten använder elementet ”tal” för olika saker. När man slår ihop dokumenten till ett enda XML-dokument så uppstår en semantisk konflikt. Namnrymder gör det möjligt för XML-dokument att identifiera och särskilja varje innebörd av ett element så att de två olika dokumenten kan användas tillsammans utan risk för sammanblandning. [17]

Konceptet med namnrymd liknar det med definitionsområde för variabler i programspråk. Om man deklarerar variabeln *i* i funktionen ComputeAverage( ) är defintionsområdet för *i* inuti funktionen ComputeAverage ( ). Om någon annan funktion, till exempel ComputeMax( ), deklarerar en variabel som har samma namn som den föregående variabeln, det vill säga *i*, blir de två variablerna olika eftersom de är definierade i olika funktioner. De har olika definitionsområde. [3]
För att kunna använda en namnrymd måste man deklarera den i ett elements starttagg med hjälp av en särskild attributspecifikation. [8]

Syntaxen för en namnrymdsdeklaration:



Ordet xmlns i namnrymdsdeklarationen är ett fördefinierat prefix som uteslutande används för att definiera namnrymden. Ett suffix, är det som knyter namnrymdsnamnet till prefixen i element- och attributnamn. Namnrymdsnamnet identifierar namnrymden med en unik sträng. För att garantera att strängen verkligen är unik använder man syntaxen för webbadresser (URI- Uniform Resource Identifier) i namnrymdsnamn [8]. Det intressanta i detta sammanhang är endast att de är unikt identifierade. Det som URI:erna pekar på är alltså helt ointressant, eftersom namnrymdsnamnet är ett namn och inte en adress. [18]
Namnrymder ger inte ett absolut skydd, för betrakta att vi har två XML-dokument som skall kombineras och de båda har samma namnrymddeklaration med referens till samma namnrymd som har olika innebörd då kommer det att uppstå en semantisk konflikt.

Deklarationen kan skrivas på två sätt. Med prefix: [6]

```
<ns1:a xmlns:ns1 =" http://example.org /ns1/" >
<ns1:b> text </ns1:b>
```

</ns1:a>

Eller utan prefix:

```
<a xmlns = ”http://example.org/ns1/” >
    <b> text </b>
</a>
```

Attributet xmlns, utan ett följande prefix, deklarerar standardnamnrymden, det vill säga namnrymden för de element som saknar attribut. [7]

En namnrymd är giltig för det element den är deklarerad för och dess innehåll, inklusive element som finns i det elementet. [6]

Exemplet nedan som är hämtad från [7] förklarar defintionsområdet för namnrymder:

```
<?xml version=”1.0” encoding =”UTF-8” ?>
<bk:bookmarks xmlns:bk=”http://www.pineapplesoft.com/2001/bookmark”>
    <bk:site href=http://www.marchal.com>
        <bk:title> Benoït Marchal </bk:title>
         <ns:rating xmlns:ns=http://www.playfield.com/parental/en/1.0>
         </ns:rating>
    </bk:site>
</bk:bookmarks>
```

Exempel 4.3 Defintionsområde för namnrymder

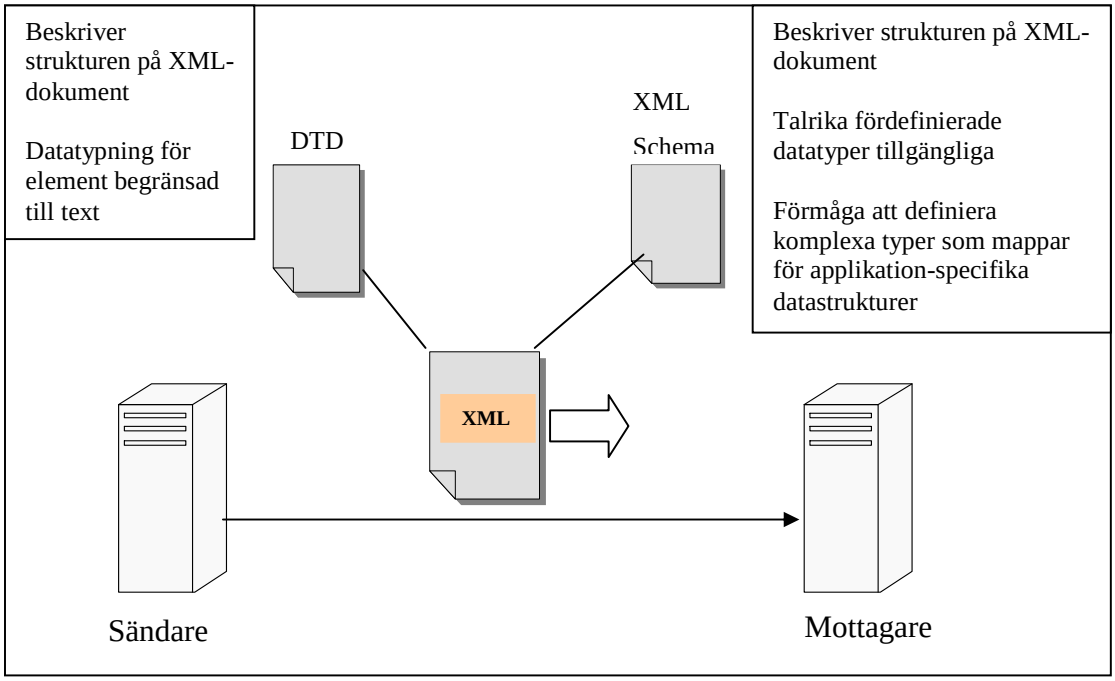
Två namnrymder är deklarerade i exemplet ovan. Som rotelement är bk deklarerat och är därför giltigt för alla element. ns är deklarerat och är giltig endast för rating-elementet.

5 Schemaspråk

När XML kom var begreppet schemaspråk knappt uppfunnet. Det fanns bara ett och det ingick i standarden. De scheman man skrev kallades dokumenttypsdefinition (DTD) och var liksom allt annat i XML en förenkling av något i SGML. Idag används därför DTD som beteckning på scheman skrivna enligt det gamla skrivsättet. [6]. Scheman är ett sätt att definiera grammatiken i ett märkspråk. Man använder scheman till att validera dokument av ett visst slag. Valideringen är en maskinell kontroll av strukturen och ibland innehållet i dokumentet. Ett dokument som överensstämmer med schemat sägs vara giltigt. [7]
Historiskt sett går de nya schemaspråken tillbaka ända till dagarna för XML:s tillkomst. Många års användning av SGML visade på brister i DTD-syntaxen och medan XML tog form började man arbeta på nya schemaspråk. [6]
I det här kapitlet skall vi titta i tur och ordning på tre vanliga sätt att uttrycka scheman:

DTD - klassiska scheman
XML Schema - det nya schemaspråket från W3C
Relax NG - ett annat schemaspråk.

Det är viktigt att förstå att DTD:er och XML-scheman i grund och botten tjänar samma syfte. Båda beskriver strukturen på XML-dokument och båda används för att validera dokument mot deras modeller. (se figuren nedan)
Det smartaste valet mellan DTD:er och XML-scheman beror således på vilka verktyg man har. Om de verktyg man använder fungerar bäst med DTD:er använder man dem. Om de däremot fungerar bättre med XML-scheman använder man XML-scheman. [7]



Figur 5.1 Båda DTD och XML-schema kan användas för att definiera strukturen av XML-dokument

5.1 DTD

Det är inte särskilt praktiskt att alla som lagrar information i XML skapar sitt eget märkspråk. Är man i en bransch där man gärna delar dokument med varandra måste man använda sig av samma taggar och en viss struktur. Det är DTD:s styrka, att få applikationer, organisationer och andra grupper att enas om att följa en standard, för att effektivt kunna utbyta information [4]. Här definierar man de regler som skall gälla när man organiserar sina XML-dokument. Typiska regler är i vilken ordning olika element får läggas in i dokumentet eller om ett element måste finnas med i ett XML-dokument eller inte. DTD:n kvalitetssäkrar helt enkelt att ett dokument är korrekt enligt de regler man satt upp.[12]

DTD:n är ett arv från SGML, som blev en ISO-standard 1986. Många utvecklare i XML:s samfund kände att DTD:er inte är tillräckligt flexibla för att möta dagens programmeringsbehov [3].

Det kanske mest kännetecknande för DTD:er, i jämförelse med de flesta andra schemaspråk, är att DTD:er inte skrivs i XML-syntax. DTD:er skrivs med en särskild syntax. Detta kan lättast visas med ett exempel:

```
<!ELEMENT  address-book (entry +) >
<!ELEMENT  entry (name, adress*, tel*) >
<!ELEMENT  name   (# PCDATA) >
<!ELEMENT  address (# PCDATA)>
<!ELEMENT  tel        (# PCDATA)>
```

Exempel 5.1 DTD:s syntax

I exemplet beskriver DTD:n en dokumenttyp som heter "*address-book*" som kan innehålla en eller flera element av typen "*entry*", därav +-tecken. Elementet "*entry*" består i sin tur av elementen "*name*", "*address*" och "*tel*", där adressen och telefonnumret kan vara noll eller flera, därav *-tecken. Elementen "*name*", "*address*" och "*tel*" kan innehålla text, därav #PCDATA. PCDATA står för "Parsed Character DATA". Genom att skriva #PCDATA berättar vi att elementet inte innehåller några underordnade element, utan bara textinnehåll. [13]

Det kan vara både en fördel och en nackdel att skriva DTD:er med en annan syntax än XML-syntax. Fördelen är att de blir mycket kompakta och lätta att överblicka för oss människor. Nackdelen är att hantering av DTD:er ställer extra krav, inte bara på en validerande tolk, utan också på alla andra program som ska förstå koden.

En DTD kan alltså hjälpa till med standardiseringen av dokument om den är utformad på ett riktigt sätt, det vill säga använder "förståliga" namn och bra struktur. Om DTD:n är väl genomtänkt bör skapandet av XML-dokument var enkelt och smärtfritt eftersom större delen av XML-filens uppbyggnad redan är bestämd. Det är bara att fylla i med sin text eller data och sedan är det klart.

5.1.1 Interna eller externa DTD:er

I ett dokument som skall följa en specifik DTD måste den aktuella DTD:n anges. DTD:n kan antingen finnas internt i XML-dokumentet eller som en separat fil. I fallet med en intern DTD, definieras denna i dokumentets DOCTYPE declaration subset. Ett subset anges genom "hakparenteser" enligt följande exempel <!DOCTYPE address-book [subset]> .

I exemplet nedan visas ett XML-dokument med en internt deklarerad DTD. [13]

```
<?XML version=”1.0” encoding =”Iso-8859-1”?>
<! DOCTYPE address-book
[
  <!ELEMENT  address-book (entry +) >
  <!ELEMENT  entry (name, adress*, tel*) >
  <!ELEMENT  name   (# PCDATA) >
  <!ELEMENT  address (# PCDATA)>
  <!ELEMENT  tel        (# PCDATA)>
]>
<address-book>
    <entry>
       <name> John Doe </name>
       <address> Storgatan 34 </address>
       < tel > 08-9997098 </tel>
    </entry>
</address-book>
```

Exempel 5.2 : Intern DTD

En extern DTD kan pekas ut genom att ange en sökväg i det egna systemet eller någonstans på Internet genom att ange en URL till den plats där filen lagras. Nedan har vi ett exempel på ett XML-dokument med en externt deklarerad DTD. [13]

```
<?XML version=”1.0” encoding =”Iso-8859-1”?>
<! DOCTYPE address-book SYSTEM “abok-dtd.dtd”>
    <address-book>

     .....

    </address-book>
```

Exempel 5.3: Extern DTD

Att använda en extern DTD-delmängd är fördelaktigt huvudsakligen om man har en gemensam DTD som används i en hel grupp dokument. Varje dokument kan referera till en enda DTD-fil (eller en kopia av filen) i form av en extern DTD-delmängd. Därmed behöver DTD:ns innehåll inte kopieras till alla dokument där den används, och det blir också lättare att underhålla den.

5.1.2 Fördelar och nackdelar med DTD:er

Fördelarna kan sammanfattas så här: [6]

- Syntaxen är inte XML-välskrivna DTD:er är korta och överblickbara.
- Programstöd - inget annat schemaspråk har så många implementationer.
- Standard - alla XML-och SGML-verktyg kan använda dem.

Och nackdelarna kan sammanfattas så här:

- Syntaxen är inte XML - DTD:er har sin egen syntax som inte är kompatibel med XML-dokument.
- Namnrymder - DTD:er och namnrymder är ingen bra kombination.
- Brist på datatyper - DTD saknar konventionella datatyper
- Gamla modelleringskoncept - DTD:er baseras på 20 år gamla modelleringskoncept. De har inget stöd för modern utformning, som till exempel objektorienterad modellering.

5.2 XML-Schema

Eftersom vår tillämpningsuppgift bygger på just detta schemaspråk kommer vi att skriva mer i detalj om dess syntax och vad det har för funktionalitet, jämfört med DTD.

Alla tycks vara överens om att DTD saknar en del väsentlig uttryckskraft, som visat sig angelägen i vissa fall. Bland annat villkor som exempelvis 0:3 eller < 1000 går inte att uttrycka. [1]

Inte heller går det att precisera vilken datatyp som ska gälla för ett dataelement (annat än # PCDATA). Det kan exempelvis vara viktigt att kunna deklarera att ordernummer är ett fyrsiffrigt heltal mellan 2700 och 5800 eller att summa inte får vara större än 100 000 kronor. Förutom tillgång till standardiserade datatyper bör egna datatyper kunna specificeras för unika behov, exempelvis olika typer av tidsintervall och symbolsekvenser. [1]
Detta har lett till viss frustration. Som svar har W3C år 1998 utvecklat ett försök till ersättning som kallas XML-schema. Syftet med XML-schemat är att åstadkomma ett betydligt kraftfullare språk för dokumenttypsspecifikation än vad XML 1.0-standarden erbjuder med sitt DTD-språk. [7] Förutom de egenskaper som återfinns hos en DTD så har XML-schemat även ett antal nya. Ett XML-schema är i sig självt ett XML-dokument, till skillnad från DTD [12]. Detta gör att XML-scheman kan editeras och behandlas som vilket annat XML-dokument som helst. Det betyder att dagens verktyg ganska enkelt kan anpassas till att även hantera XML-scheman. [13]
XML-schema behandlas i tre stora dokument som finns beskrivna i XML-schemas specifikation: [6]

- XML Schema part 0: primer
En välbehövlig introduktion som ger inblick i det mesta i standarden.

- XML Schema part 1: structures
Ger en beskrivning av hur man upprättar regler för grammatiken i ett dokument.

- XML Schema part 2: datatypes
Ger en beskrivning av de inbyggda datatyperna och deras egenskaper, samt hur man ska bilda egna datatyper utifrån de befintliga.

Dokumenten finns på W3Cs webbplats www.w3c.org.

5.2.1 Namnområde

Schemat börjar med elementet Schema, som alltid knyts till följande namnområde:

http://www.org/2001/XMLSchema

Namnområdet innehåller fördefinierade begrepp såsom ”schema”, ”complexType”, ”element”, med flera, som används i XML-schema-dokumentet.

Av konvention används prefixet xsd, som är en förkortning för XML Schema Definition, för att koppla elementen till ett namnområde. Rotelementet i ett schema ser alltså ut så här:

```
<xsd:Schema xmlns : xsd =”http://www.org/2001/XMLSchema”>

.....

</xsd:Schema>
```

Exempel 5.4 Rotelementet i XML-schema

[6]

5.2.2 Element

Att deklarera ett element eller attribut i ett schema innebär att man tillåter att elementet eller attributet med det angivna namnet, typer och andra egenskaper förekommer i ett visst sammanhang i ett överensstämmande XML–dokument. [8]

Element deklareras med schemaelementet *xsd:element*: [6]

```
<xsd:element name="TITEL" type="xsd:string"/>
```

Exempel 5.5 Elements deklaration

Elementets namn anges i attributet *name*. Innehållet regleras endera genom attributet *type* som ovan, eller genom elementet *complexType*, som i sin tur kan innehålla andra element- och attributdeklarationer.

```
<xsd:element name="INVENTARIUM">
<xsd:complexType>
      .....
    </xsd:complexType>
</xsd:element>
```

Exempel 5.6 Användning av complextype

Man skiljer på enkla och komplexa datatyper. Element som enbart innehåller teckendata sägs ha enkla datatyper, medan element som har komplexa datatyper kan innehålla förutom teckendata ett eller flera underordnade element eller attribut.Attribut har alltid enkla datatyper, eftersom attributvärden inte har någon inre struktur.

Elementet *TITEL* i exempel 5.5 har en enkel datatyp, eftersom elementet varken innehåller attribut eller underelement, bara ett värde. I exempel 5.6 har *INVENTARIUM* underelement. Elementet är därför en komplex datatyp. [8]

5.2.3 Attribut

Attribut deklareras med schemaelementet *xsd:attribute*:

<xsd:attribute name=”nummer” type =”xsd:integer”/>

Egenskaperna för attribut liknar i hög grad egenskaperna för element. Enligt reglerna för välutformning får ett attribut förekomma högst en gång i ett element. För att styra förekomsten av ett attribut i elementet där det deklareras kan man använda sig av attributet *use*, som kan anta tre värden:

Värde	Innebörd
Optional	Attributet är valfritt
Required	Attributet är obligatoriskt
Prohibited	Attributet får inte användas

Om inget anges är attributet valfritt (*optional*), så normalt är det bara *required* man behöver använda. Värdet *prohibited* används i scheman som spänner över flera filer, där man i vissa filer vill förbjuda användningen av ett attribut som definierats i en annan fil. [6]

5.2.4 Datatyper

En av de stora skillnaderna i förhållande till DTD:erna är de utbyggda möjligheterna att definiera information som tillhör en särskild datatyp och att kontrollera att informationen följer de regler som är knutna till datatypen. Exempelvis vill man att ett pris skall vara ett numeriskt värde istället för en textsträng, eftersom man kan behöva göra beräkningar. Det finns även möjligheter för datumhantering. Man kan även definiera att ett postnummer skall bestå av siffror och vara fem tecken långt. [13]
XML-Schemat introducerar ett stort antal datatyper som ger möjlighet att göra en innehållsmässig validering. Datatyper är användbart i tillämpningar där man överför data mellan program och databaser.
Här följer en förteckning över de fördefinierade enkla datatyper som finns att tillgå: [6]

Typ	Beskrivning
String	En vanlig sträng
normalizedString	En sträng utan tabulering, radmatning och vagnretur
Token	En sträng utan tabulering, radmatning och vagnretur, utan mellanslag före och efter och utan flera mellanslag i följd

Integer	Ett heltal
Long	Ett heltal från -9 223 372 036 854 775 808 till 9 223 372 036 854 775 807
Int	Ett heltal från -2 147 483 648 till 2 147 483 647
Short	Ett heltal från -32 768 till 32 767
Byte	Ett heltal från -128 till 127
nonPositiveInteger	Ett heltal som inte är större än 0
nonNegativeInteger	Ett heltal som inte är mindre än 0
negativeInteger	Ett heltal som inte är större än -1
positiveInteger	Ett heltal som inte är mindre än 1
unsignedLong	Ett heltal från 0 till 18 446 744 073 709 551 615
unsignedInt	Ett heltal från 0 till 4 294 967 295
unsignedShort	Ett heltal från 0 till 65 535
usignedByte	Ett heltal från 0 till 255
Decimal	Ett decimaltal
Float	Ett 32-bitars decimaltal enligt IEEE 754
Double	Ett 64-bitars decimaltal enligt IEEE 754
Date	Ett datum (2007-04-14)
Time	Ett klockslag (11:04:23)
dateTime	En kombination av datum och tid (2007-04-14T11:04:23)
Duration	En löptid (P1Y2M3DT4H5M6S)
gYear	Ett år med fyra siffror (2007)
gMonth	En månad med två siffror (04)
gDay	En dag med två siffror (14)
gYearMonth	En kombination av år och månad (2007-04)
gMonthDay	En kombination av månad och dag (04-14)
Boolean	Något av värdena true, false, 1 eller 0
hexBinary	Binär data kodad med hjälp av tecken 0-9 och A-F
ID IDREF	Attributtyper för DTD-syntaxen

IDREFS ENTITY ENTITIES NMTOKEN NMTOKENS NOTATION	Attributtyper för DTD-syntaxen

Tabell 5.1 Fördefinierade enkla datatyper

Inbyggda och härledda datatyper

Av dessa ovannämnda datatyper anses några vara inbyggda:

dataTime	decimal	boolean	date
float	gDay	double	duration
gYear	gYearMonth	hexBinary	NOTATION
gMonth	gMonthDay	string	time

Alla de andra kallas för härledda datatyper. Härledda datatyper kan bara härleddas från andra existerande datatyper. Den vanligaste typen av härledning är att begränsa intervallet av tillåtna tecken eller tal. Exempelvis *token* är härledd ur *normalizedString*, som i sin tur är härledd ur den inbyggda datatypen *string*. datatyperna bildar en hierarki, se bilaga A.

Bygga egna datatyper

Det går att definiera sina egna datatyper. Den nya typen härleds från en av de inbyggda enkla typerna, eller från en annan, redan definierad och härledd enkel typ.

```
<xsd:element name="PRIS">
  <xsd:simpleType> <!-- nya enkla typer definieras med hjälp av schemaelementet
   xsd:simpleType-->
   <xsd:restriction base="xsd:decimal">
   <!-- elementet xsd:restriction anger bastypen och innehåller särskilda
    schemaelementet som kallas aspekter, vilka visar exakt hur bastypen begränsas -->
<xsd:minExclusive value=”0” />
<xsd:maxExclusive value=”1000”/>
 </xsd:restriction>
 </xsd:simpleType>
 </xsd:element>
```

Exempel 5.7 Begränsa decimal typen

Den här deklarationen tilldelar *PRIS*-elementet en ny definierad typ som härleds från den inbyggda typen *xsd:decimal*. Den nya typen har alla *xsd:decimal* egenskaper, förutom att värdet som matas in i elementet måste vara större än 0 och mindre än 1000.
Man kan även använda en följd av *xsd:enumeration*-aspekter om man vill begränsa elementets innehåll (eller attributets värden) till ett i en mängd av specifika värden. Betrakta följande exempel:

```
<xsd:element name="FILM">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="action"/>
        <xsd:enumeration value="komedi"/>
        <xsd:enumeration value="dokumentär"/>
    </xsd:restriction>
</xsd:simpleType>
 </xsd:element>
```

Exempel 5.8 Användning av enumaration för att begränsa elementets innehåll

I exemplet ovan har *FILM*-elementet begränsats till en specifik uppsättning strängar. Ett *FILM*-element som innehåller något annat än en av de tre strängarna som har angetts i attributet *value* till elementen *xsd:enumeration* är ogiltigt.[8]

Komplexa datatyper

Komplexa datatyper definieras med schemaelementet *xsd:complexType*, som motsvarar elementet *xsd:simpleType* i definitionen av en enkel typ. *xsd:complexType* används för att deklarera ett element med elementinnehåll (enbart underordnade element), blandat innehåll (underordnade element och teckendata) eller inget innehåll alls (varken underordnade element eller teckendata). [8]

Element med elementinnehåll

Elementen av denna typ måste ha en innehållsmodell som beskriver tillåtna underordnade element, deras ordning och reglerna för deras förekomst. Innehållsmodellen anges med schemaelementen *xsd:sequence, xsd:choice, xsd:all* eller en kombination av dessa.
En grupp underordnade element som deklareras i elementet *xsd:sequence* måste stå exakt i den ordning de förekommer, det liknar kommatecknet i DTD:er [7]. I deklarationen av *TITEL, FÖRFATTARE, SIDOR* och *PRIS,* i nyss nämnda ordning:

```
<xsd:element name="BOK">
 <xsd:complexType >
   <xsd:sequence minOccurs="1" maxOccurs="99">
     <xsd:element name="TITEL" type="xsd:string">
     <xsd:element name="FÖRFATTARE" type="xsd:string">
     <xsd:element name="SIDOR" type="xsd:string">
     <xsd:element name="PRIS" type="xsd:string">
   </xsd:sequence>
 </xsd:compexType>
</xsd:element>
```

Exempel 5.9 Användning av sequence för att ange ordningen av elementen

Ett exempel på ett giltigt *BOK*-element i ett XML-instansdokument (XML-dokument som är anpassat till schemat):

```
<BOK>

  <TITEL> Legenden om sleepy Hollow          </TITEL>
  <FÖRFATTARE> Washington Irving             </FÖRFATTARE>
  <SIDOR> 98                                 </SIDOR>
  <PRIS> 87                                  </PRIS>

</BOK>
```

Exempel 5.10 Giltigt XML-dokument

För att styra upprepningar används attributen *minOccurs* och *maxOccurs*, som vi hr gjort med exemplet ovan. Standardvärdet för båda dessa är 1, vilket innebär att elementet endast får förekomma en gång, men vilket heltal som helst är giltigt. Bland annat så visar detta att scheman är kraftfullare än DTD:er eftersom man kan välja hur många gånger ett element får förekomma i dokumentet. I exemplet ovan måste elementen *TITEL, FÖRFATTARE, SIDOR* och *PRIS* förekomma minst 1 gång och högst 99 gånger. [7]
Om en grupp underordnade element deklareras i schemaelementet *xsd:choice* kan vilket som helst förekomma i det överordnade elementet [8]. Choice ersätter lodstrecket hos DTD:er. Följande deklaration innebär att elementet *FILM* får innehålla elementet *STJÄRNA* eller *BERÄTTARE* eller *INSTRUKTÖR*

```
<xsd:element name="FILM">
  <xsd:complexType >
    <xsd:choice>
        <xsd:element name="STJÄRNA" type="xsd:string">
        <xsd:element name="BERÄTTARE" type="xsd:string">
        <xsd:element name="INSTRUKTÖR" type="xsd:string"
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Exempel 5.11 Användning av cshoice för att ange större frihet för ordningen av element

Här är ett exempel på ett giltigt *FILM*-element:

```
<FILM>
    <STJÄRNA> Sandra Bullock    </STJÄRNA>
</FILM>
```

Exempel 5.12 Giltigt FILM-element

En grupp underordnade element som definieras i schemaelementet *xsd:all* får förekomma i valfri ordning. Precis som *xsd:sequence* kräver *xsd:all* att alla beståndsdelar används, men det definierar inte någon ordning. Det finns dock vissa begränsningar för *xsd:all* det kan endast finnas högst upp i en innehållsmodell och dess beståndsdelar måste vara element. Det finns ingen motsvarighet i DTD - ytterliggare ett exempel där XML-scheman är kraftfullare. Om man har behov av en mer komplicerad innehållsmodell kan man bädda in elementet *xsd:sequence* i elementet *xsd:choice* eller tvärtom. Men det går inte att bädda in *xsd:all* i elementen *xsd:sequence* eller *xsd:choice.* Det är likaledes inte tillåtet att bädda i *xsd:sequence* i elementet *xsd:all*. [8]

Element med blandat innehåll

Element med text och underelement jämsides har blandat innehåll [6]. Element med blandat innehåll deklareras ha elementinnehåll på precis samma sätt som i föregående avsnitt, men attributspecifikationen *mixed = ”true”* måste läggas till i elementets *xsd:complexType* starttagg. [3]
Enligt följande deklaration får t.ex. elementet *TITEL* innehålla teckendata före eller efter sitt enda underordnade element *UNDERTITEL*:

```
<xsd:element name="TITEL">
   <xsd:complexType mixed=”true”>
    <xsd:sequence>
        <xsd:element name="UNDERTITEL" >
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Exempel 5.13 Vilken som helst textsträng kan komma före och efter elementet UNDERTITEL

Exempel på ett giltigt element:

```
   <TITEL> Moby Dick <UNDERTITEL> eller Valen </UNDERTITEL> </TITEL>
```

Exempel 5.14 Ordet eller kom före namnet på undertiteln vilket är giltigt

Exemplet visar att vi får mer kontroll över blandat innehåll i ett XML-schema än vad som är möjligt i en DTD. Motsvarigheten till blandat innehåll i DTD:er blir så här i ett XML-schema: [8]

```
  <xsd:element name="STYCKE">
     <xsd:complexType mixed=”true” >
        <xsd:choice  minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="FET" type="xsd:string"/>
            <xsd:element name="KURSIV" type="xsd:string"/>
            <xsd:element name="UNDERSTRUKEN" type="xsd:string"/>
        </xsd:choice>
    </xsd:complexType>
  </xsd:element>
```

Exempel 5.15 Blandat innehåll i XML-schema

Exemplet stipulerar att elementet *STYCKE* får innehålla de underordnande elementen *FET*, *KURSIV* och *UNDERSTRUKEN* i valfri ordning och hur många gånger som helst (noll eller flera) uppblandade med teckendata.

Element utan innehåll

Ett element som har deklareras vara tomt får varken innehålla underordnade element eller teckendata. Den här elementtypen definieras med hjälp av elementet *xsd:complexType*. Innehållsmodellen utlämnas emellertid, som i följande exempel:

```
<xsd:element name="BR">
    <xsd:complexType >
    </xsd:complexType>
</xsd:element>
```

Exempel 5.16 Elementet BR saknar innehåll

5.3 RELAX NG

XML-schema är inte det enda alternativet för den som behöver något mer än DTD:er. Ett av de alternativ som kom till som en direkt respons på XML-schema var Regular Language Description for XML, som förkortas RELAX. Språket utvecklades av japanen Makoto Murata, som dessförinnan varit med i utvecklingen av XML. Tack vare dess relativa enkelhet fick RELAX mycket uppmärksamhet under en period.
Ett annat schemaspråk som dök upp under samma period var Tree Regular Expressions for XML (TREX), som utvecklades av James Clark. James Clark och Makoto Murata arbetade ihop och resultatet blev RELAX NG, där NG står för New Generation. [6]
Sedan sammanslagningen utvecklades RELAX NG under OASIS (Organization for the Advancement of Structured Information Standards, är ett internationellt konsortium som arbetar med standarder inom strukturerad information.) försorg. Den syftar till att ge samma fördelar som XML-schema men den ska vara enklare att använda och mindre komplex. [7]
RELAX NG handlar enbart om kontroll av struktur och datatyper – ingenting i schemaspråket kan påverka innehållet i dokumentet. Element och attribut behandlas i hög grad lika i RELAX NG som de andra schemaspråk. RELAX NG stödjer datatyperna i XML-schema, men är också

öppet för andra alternativ.
RELAX NG kan skrivas på två sätt: endera i XML eller i en kompakt syntax, som lämpar sig bättre om man skriver för hand.

5.4 Skapa XML-schemat och XML-filen för tillämpningsuppgiften

Efter att vi har förklarat XML-schemat detaljerat är det nu dags att ge ett förslag på ett XML-schema till vår tillämpningsuppgift med avseende på de begränsningar som ges i tabellen 3.1 i kapitel 3. Dessa begränsningar kommer att tas upp igen när vi förklarar varje element som vi skall ha i XML-schemat längre fram i kapitlet. Vi har valt att skapa ett XML-schema som illustrerar nästan alla tekniker som har diskuteras i det här kapitlet. Det kan finnas ett annat sätt att skriva XML-schemat på men som sagt vi har valt att illustrera de tekniker vi tog upp i det här kapitlet för att det ska bli enkelt att förstå. Vi kan inte säga att det sättet vi har valt att förklara och därefter skapa XML-schemat efter är det bästa eller sämsta sättet för vi har inte prövat olika varianter, så att vi kan dra en slutsats som säger vilken som är bäst eller sämst.
Namn på elementen har vi skapat enligt reglerna för elementnamn som finns i avsnitt 4.1 i kapitel 4.

XML-schemat

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="SCENARIO-LIBRARY">
<xsd:complexType>
                    <xsd:sequence>
                              <xsd:element name="SCENARIO"
                              type="Scenariotyp" minOccurs="0"
                              maxOccurs="unbounded"/>
                    </xsd:sequence>
</xsd:complexType>
```

```
</xsd:element>

<xsd:complexType name="Scenariotyp">
<xsd:sequence>
                    <xsd:element name="SCENARIO-ID" type=" xsd:integer "/>
                     <xsd:element name="DESCRIPTIVE-NAME">
                              <xsd:simpleType>
                                       <xsd:restriction base="xsd:string">
                                                <xsd:maxLength
                                                value="60" />
                                       </xsd:restriction>
                              </xsd:simpleType>
                    </xsd:element>

                    <xsd:element name="DESCRIPTION">
                              <xsd:simpleType>
                                       <xsd:restriction base="xsd:string">
                                                <xsd:maxLength
                                                value="1000" />
                                       </xsd:restriction>
                              </xsd:simpleType>
                    </xsd:element>

          <xsd:element name="SOURCE">
                              <xsd:simpleType>
                                       <xsd:restriction base="xsd:string">
                                                <xsd:enumeration
                                                value="Trace"/>
                                                <xsd:enumeration
                                                value="Analytical"/>
                                                <xsd:enumeration
                                                value="Simulation"/>
```

```
                    </xsd:restriction>
                </xsd:simpleType>
</xsd:element>

<xsd:element name="PROVIDING-ORG">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength
                        value="60"/>
                    </xsd:restriction>
                </xsd:simpleType>
 </xsd:element>

<xsd:element name="PROVIDING-PERS">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength
                        value="60"/>
                    </xsd:restriction>
                </xsd:simpleType>
</xsd:element>

<xsd:element name="CONTACT-EMAIL">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength
                        value="60"/>
                    </xsd:restriction>
                </xsd:simpleType>
</xsd:element>

<xsd:element name="NR-OF-PATTERNS" >
```

```
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:integer" >
                            <xsd:minInclusive
                            value="1"/>
                            <xsd:maxInclusive
                            value="99"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>

                <xsd:element name="PATTERN-INFO">
                    <xsd:complexType >
                        <xsd:sequence minOccurs="1"
                        maxOccurs="99">
                            <xsd:element
                            name="PATTERN-TYPE"
                            type="xsd:string">
                            <xsd:element
                            name="PATTERN-
                            MODE"
                            type="xsd:string">
                            <xsd:element
                            name="PATTERN-SIZE"
                            type="xsd:string">
                            <xsd:element
                            name="PATTERN-
                            DESCRIPTION"
                            type="xsd:string">
                        </xsd:sequence>
                    </xsd:compexType>
                </xsd:element>
            </xsd:sequence>
```

</xsd:complexType>

</xsd:schema>

Innan vi börjar beskriva varje element är det viktigt att nämna att de begränsningar vi har använt här, till exempel att SCENARIO-ID skall vara ett heltal, var ett krav som fanns med i uppgiften. Allt som är understruket är begränsningar som fanns med i specifikationen till uppgiften. Andra begränsningar som till exempel att *minOccurs* och *maxOccurs* till elementet SCENARIO tilldelas 0 respektive *unbounded* var bara antaganden vi gjorde.

- Rotelementet heter SCENARIO-LIBRARY och innehåller ett godtyckligt antal SCENARIO-element därav attributen *minOccurs* och *maxOccurs* som används för att styra upprepning av ett element. vi tilldelar *minOccurs* till 0 vilket innebär att SCENARIO-elementet inte behöver förekomma alls, och *maxOccurs* till *unbounded* som innebär att SCENARIO-elementet kan förekomma hur mycket som helst. Vi kunde tilldela *maxOccurs* till en begränsad siffra så som 100 om vi ville att SCENARIO-elementet skall förekomma maximalt 100 gånger.
- Ett SCENARIO-element innnehåller följande underordnade element: SCENARIO-ID, DESCRIPTIVE-NAME, DESCRIPTION, SOURCE, PROVIDING-ORG (providing organization), PROVIDING-PERS (providing person), CONTACT-EMAIL, NR-OF-PATTERNS (number of patterns) och PATTERN-INFO. Elementet NR-OF-PATTERNS måste förekomma minst en gång och högst 99 gånger. Alla andra element förekommer exakt en gång. SCENARIO är ett element bestående av en delstruktur av en sammansatt typ (complex type): scenariotyp (antagande).
- SCENARIO-ID-elementet innehåller ett heltal (xsd:integer).
- DESCRIPTIVE-NAME-elementet innehåller valfria teckendata (xsd:string). Man får skriva upp till 60 tecken.
- Elementet DESCRIPTION innehåller valfria teckendata. Rymmer information upp till 1000 tecken.
- Elementet SOURCE innehåller en av de följande tre strängarna: trace, analytical eller simulation.
- PROVIDING-ORG- , PROVIDING-PERS-, och CONTACT-EMAIL-elementen innehåller valfria teckendata. Alla tre elementen får innehålla upp till 60 tecken.

- Elementet NR-OF-PATTERNS innehåller ett heltal som är större än eller lika med 1 och mindre eller lika med 99.
- Elementet PATTERN-INFO innehåller fyra underordnade element; PATTERN-TYPE, PATTERN-MODE, PATTERN-SIZE och PATTERN-DESCRIPTION; som alla innehåller godtyckliga teckendata och förekommer minst 1 gång och högst 99 gånger i den angivna ordningen. De underordnade elementen förekommer beroende på antalet mönster vi har, det vill säga NR-OF-PATTERNS. PATTERN-INFO-elementet var ett antagande som vi gjorde för att den skulle innehålla all mönsterinformation.

Enligt punkt 2 i avsnitt 3.3 i kapitel 3 skall vi skapa ett giltigt XML-dokument som följer reglerna eller begränsningarna i XML-schemat som vi har skapat ovan. Texten som finns i XML-dokumentet är bara ett exempel på giltiga elements innehåll. Det vi skall göra nedan är bara ett exempel på ett giltigt XML-dokument som sagt och det måste inte innehålla just de informationer vi har som elementinnehåll. Vi skall ha ett SCENARIO-element i det här exemplet. Alla underordnade element till SCENARIO måste förekomma exakt 1 gång förutom PATTERN-INFO-elementet och dess underordnade element som kan förekomma beroende på hur många mönster vi har.

XML-filen

```
<?xml version="1.0" encoding="UTF-8"?>

<SCENARIO-LIBRARY>

  <SCENARIO>

    <SCENARIO-ID> 12312349 </SCENARIO-ID>

    <DESCRIPTIVE-NAME> WLAN Handover – 1 </DESCRIPTIVE-NAME>
```

<**DESCRIPTION**> This scenario emulates the IP -level conditions seen by a single WLAN user as he moves between two access points located in at ENSICA, Toulose. 802.11b is used, and there were no other active traffic during the measurements. This scenario is part of a larger collection with scenario files for the same conditions. </**DESCRIPTION**>

<SOURCE> Trace **</SOURCE>**

<**PROVIDING-ORG**> ENSICA </**PROVIDING-ORG**>

<**PROVIDING-PERS**> Tanguy Perennou </**PROVIDING-PERS**>

<**CONTACT-EMAIL**> tanguy.perennou@gmail.com </**CONTACT-EMAIL**>

<**NR-OF-PATTERNS**> 2 </**NR-OF-PATTERNS**>

<**PATTERN-INFO**>

<**PATTERN-TYPE**> Loss </**PATTERN-TYPE**>

<**PATTERN-MODE**> Time </**PATTERN-MODE**>

<**PATTERN-SIZE**> 200 seconds </**PATTERN-SIZE**>

<**PATTERN-DESCRIPTION**> 5 losses </**PATTERN-DESCRIPTION**>

</**PATTERN-INFO**>

<**PATTERN-INFO**>

<**PATTERN-TYPE**> Bandwidth change </**PATTERN-TYPE**>

<**PATTERN-MODE**> Time </**PATTERN-MODE**>

<**PATTERN-SIZE**> 200 seconds </**PATTERN-SIZE**>

<**PATTERN-DESCRIPTION**> 154 bandwidth changes
</**PATTERN-DESCRIPTION**>

</**PATTERN-INFO**>

</**SCENARIO**>

</**SCENARIO-LIBRARY**>

XML-dokumentet ovan är bara ett litet exempel och för att det inte skall ta mer plats så hänvisar vi till bilaga B där vi har skapat ett giltigt XML-dokument som innehåller mer SCENARIO-element .

6 XSLT

6.1 Grunderna i XSLT

Det här kapitlet utgör bara en introduktion till XSLT, ett ämne som lätt skulle kunna fylla en hel bok. Vi har valt att berätta om just XSLT eftersom vår tillämpninguppgift utgår från just denna typ av teknik.

En av fördelarna med att använda XML som standard för datautväxling mellan program är enkelheten att omvandla XML-datadokument till andra format och strukturer som t'ex HTML-filer. XSLT (Extensible Stylesheet Language Transformations) är ett exempel på processinriktat märkspråk baserat på XML för transformering av dokument [19]. Den grundläggande formen av XSLT som beskrivs i det här kapitlet transformerar ett XML-dokument till en HTML-sida, som webbläsaren sedan visar.

Att transformera innebär att man kopierar data från ett dokument och skapar ett nytt, ofta med helt andra namn och en annan struktur i uppmärkningen. Transformeringen behövs dels när ett dokument ska presenteras, och dels när man överför data mellan två snarlika märkspråk [6]. Vi ska ägna oss åt presentationen.

För att förstå hur XSLT fungerar måste man känna till hur webbläsare, till exempel Internet Explorer, hanterar XML-data. Ett XML-dokument som öppnas i en webbläsare avkodas som en trädliknande struktur. När webbläsaren hittar en referens till ett XSLT-dokument i XML-dokumenten, öppnas formatmallsfilen och tillämpas på indataträdet. Webbläsaren går igenom källfilen och försöker hitta mönster i XSLT-filen som motsvarar specifika element, även kallat noder, i XML-dokumenten. När ett sådant mönster hittas tillämpas alla motsvarande mallar på elementen, och informationen visas i enlighet med reglerna i mallen. Resultatet blir ett utdataträd som avkodas av webbläsaren och visas baserad på HTML-instruktionerna som användes i XSLT-filen [7].
XSLT ger möjlighet att välja ut just de XML-data man är intresserad av, visa informationen i valfri ordning och utformning och lägga till information eller välja ut valda delar av

grunddokumentet.
XSLT ger tillgång till alla komponenter i XML-dokumentet, till exempel element, attribut, kommentarer och processinstruktioner. [8]

Eftersom XSLT är baserat på XML innebär det att XSLT är ett välutformat XML-dokument som efterlever vissa regler. Precis som alla andra XML-dokument består XSLT-dokument av vanlig text och skrivs med i princip vilken texteditor som helst [8].

XSLT länkas till XML-dokumentet i processinstruktionen XML-stylesheet, som i allmänhet ser ut på följande sätt:

<? xml-stylesheet type =”text/xsl” href =”sökvägTillXSL”?>

sökvägTillXSL är en URL som talar om var xsl-dokumentet lagras. I vår tillämpningsuppgift har vi skrivit processinstruktionen på följande sätt:

<? xml-stylesheet type =”text/xsl” href =”XslScenario.xsl”?>
Processinstruktionen XML-stylesheet måste stå i början av XML-dokumentet, efter XML-deklarationen och dokumenttypsdeklarationen, om en sådan står med [6].

Som prefix används nästan alltid xsl. Rotelementet heter xsl:stylesheet och innehåller alltid attributet version för att ange vilken version av XSLT som används.

```
<xsl:stylesheet version =”1.0”

        xmlns:xsl =”http://www.w3.org/1999/XSL/Transform” >

        <!--  ett eller flera element  -- >

</xsl:stylesheet>
```

Exempel 6.1 Rotelementet i XSLT

Alla XSLT-element tillhör namnområdet http://www.w3.org/1999/XSL/transform som bör

deklareras i rotelementets starttagg, så att man kan använda det definierade namnområdeprefixet i alla element. [19]

Rotelementet kan innehålla en eller flera mallar, som alla definieras i varsitt xsl:template-element. Xsl:template-elementet innehåller alltid något av attributen match eller name. Attributet match i xsl:template pekar ut vilken komponent eller vilka komponenter i XML-dokumentet som mallen avses transformera.

```
<xsl:template match =" / " >

    <! --  underordnade element -->

</xsl:template>
```

Exempel 6.2 "/" matchar XSLT-rotnoden
[6]

Innehållet i match-attributet följer syntaxen för XML-sökvägar, som kallas Xpath. Xpath används tillsammans med XSLT för navigering mellan elementen och attributen i en XML-datastruktur. Uttrycket "/" som användes i exempel 6.2 är ett Xpath-uttryck som matchar XSLT-rotnoden, som representerar hela XML-dokumentet [19]. Sökvägar kan även matcha attribut, och det gör man genom att infoga attributnamnet föregånget av ett at-tecken (@), matcha med ett villkor, i så fall matchar det endast om villkoret är sant (villkor står inom vinkelparenteser efter det element villkoret gäller). Anta till exempel att elementet BOK har ett attribut som heter ILager och nu vill vi välja ut alla BOK-element som har ett attribut vid namn ILager med värdet ja, då gör vi på följande sätt:
<xsl:for-each select="INVENTARIUM/BOK[@ILager='ja']">

De kan även matcha text och funktioner. Nedan visas en tabell med de vanligaste XSLT-funktionerna [7].

XSLT-funktion	Beskrivning
position()	Returnerar positionen för den aktuella noden i noduppsättningen
text()	Returnerar texten (innehållet) för ett element
last()	Returnerar positionen för den sista noden i den aktuella noduppsättningen
count()	Returnerar antalet noder i den aktuella noduppsättningen
not()	Negerar argumentet
contains()	Returnerar true om det första argumentet innehåller det andra argument
starts-with()	Returnerar true om det första argumentet börjar med det andra argumentet

Tabell 6.1 Fördefinierade funktioner i XSLT

Namngivna mallar har attributet *name* istället för *match*. Namnet i attributet gör att mallen kan anropas som en subrutin. Precis som i andra programspråk kan anropet innehålla parametrar och mallen kan ge ifrån sig ett returnvärde [6].

Transformationsinstruktionerna i en mall består av två slags XML-element:

- XML-element som representerar HTML- element, till exempel <TITLE>, <H2>, <HEAD>, <BODY>
- XSLT-element

Exempelvis <xsl: value-of select = ”FÖRFATTARE”/>
[8].

Vi ska nu ge ett enkelt exempel på hur man skapar XSLT- filen och hur den länkas till XML-dokumentet.

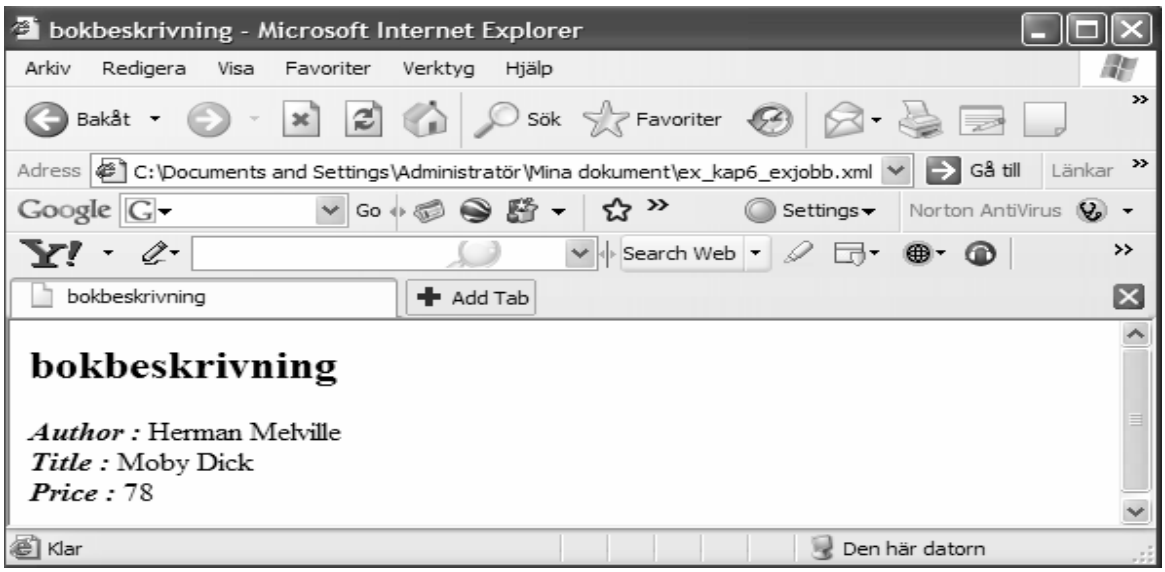
1. först skriver vi XSLT-dokumentet:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Filnamn: XSLTExempel.xsl -->

<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version ="1.0">

        <xsl:template match= "/">
          <HTML>
           <HEAD>
                <TITLE> bokbeskrivning </TITLE>
          </HEAD>
          <BODY>
          <H2> bokbeskrivning</H2>
          <B><SPAN STYLE="font-style:italic">Author : </SPAN></B>
          <xsl:value-of select="BOOK/AUTHOR"/> <BR/>
          <B><SPAN STYLE="font-style:italic">Title : </SPAN></B>
          <xsl:value-of select="BOOK/TITLE"/><BR/>
          <B><SPAN STYLE="font-style:italic">Price : </SPAN></B>
          <xsl:value-of select="BOOK/PRICE"/>
          </BODY>
          </HTML>
        </xsl:template>
</xsl:stylesheet>
```

2. Sedan gör vi ett enkelt XML-dokument som länkas till XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Filnamn:XSLT-test.xml -->
<?xml-stylesheet type="text/xsl" href="XSLTExempel.xsl"?>

    <BOOK>
        <AUTHOR>  Herman Melville </AUTHOR>
        <TITLE> Moby Dick        </TITLE>
        <PRICE> 78 </PRICE>
    </BOOK>
```

Så här visar Internet Explorer XML-dokumentet enligt instruktionerna i XSLT-dokumentet:



Visa alla element

Xml-dokumentexemplet i avsnitt 5.4 innehöll bara ett SCENARIO-element. Om dokumentet hade innehållit flera skulle endast det första ha visats om vi hade använt oss av tekniken i avsnitt 5.4.
För att visa alla element behöver man använda elementet *xsl:for-each*, som matar ut innehållet i varje element [8]. För att se ett exempel på hur xsl:for-each-element används se avsnitt 6.2.

En annan metod som man använder för att visa ett upprepat XML-element är att skapa en separat mall som matchar det och sedan anropa mallen med hjälp av elementet *xsl:apply-templates*. Apply-templates fungerar som ett rekursivt anrop [7].

Fördelen med att ha separata mallar istället för att använda en eller flera xsl:for-each-instruktioner i samma mall är att separata mallar gör det möjligt att dela upp koden i mindre och mer lätthanterliga moduler, precis som det kan vara bra att dela upp källkod i separata funktioner i olika programmeringsspråk [2].

Elementen som väljs ut bearbetas normalt i den ordning de förekommer i XML-dokumentet. Man kan låta ett eller flera *xsl:sort*-element ingå i elementen *xsl:for-each* och *xsl:apply-templates* och därigenom kontrollera i vilken ordning elementen bearbetas. [19]

Loopar och villkorssatser

I nästan alla programspråk finns det loopar och villkorsatser, och så även i XSLT.
Villkorsatser i XSLT gör det möjligt att välja vilka transformationsinstruktioner som ska utföras med ledning av vissa villkor, till exempel värdet på ett element eller attribut .
xsl:if är den grundläggande villkorssatsen, som med ett test avgör om ett instruktionsblock körs eller hoppas över.
Finns det flera alternativ använder man istället *xsl:choose.*
Xsl:choose är bara en behållare till *xsl:when* och *xsl:otherwise*. *xsl:when* har precis samma effekt som *xsl:if*, men används i samband med *xsl:choose.* Inom *xsl:choose* finns det en eller flera *xsl:when* och noll eller en *xsl:otherwise* [8].

6.2 Skapa XSLT- filen till tillämpningsuppgiften

I detta avsnitt skall vi utifrån teknikerna som vi har förklarat i detta kapitel skapa XSLT-filen till vår tillämpningsuppgift. I kapitel 3 hade vi två punkter som skall skapas enligt XSLT-tekniken:

- Skapa en XSLT-fil som ger en översiktlig presentation av informationen i XML-filen i en webbläsare.

- Skapa en XSLT-fil som ger en detaljerad presentation av informationen i XML-filen i en webbläsare. Utöver den presentation man åstadkom ovan anges även här patternbeskrivningarna som kan vara bra om man vill veta lite mer om mönstren som ingår.

För att lösa deluppgifter ovan har vi skapat två XSLT-filer. Det går även att skapa en enda

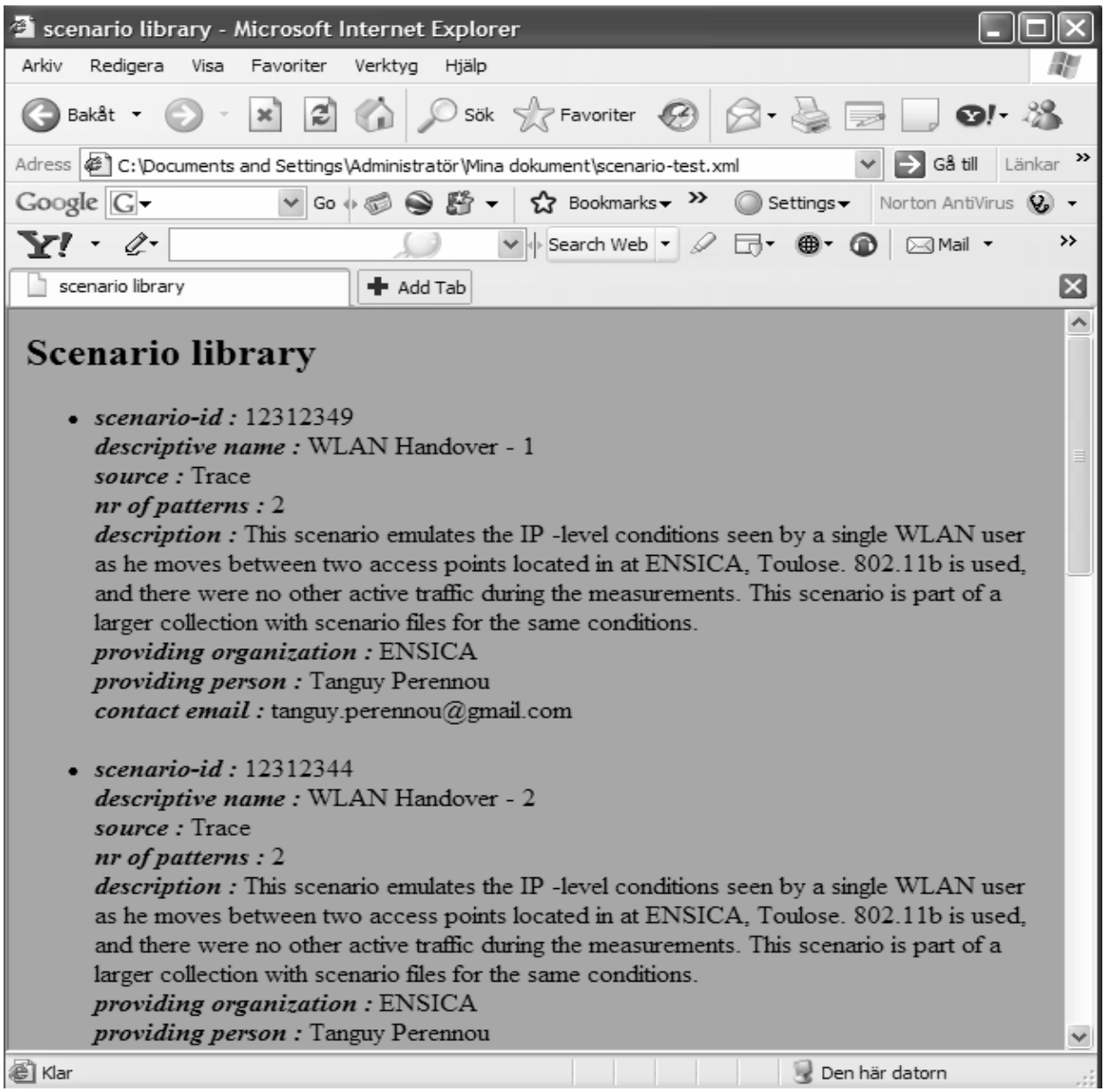
XSLT-fil men då måste man skapa ett webbgränssnitt där man kan välja exakt vilken information man vill ha. I XSLT har man tillgång till slingor och villkorsatser, vilket gör att man kan filtrera XML-informationen.

För första punkten gäller:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Filnamn: XslScenario.xsl -->
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version ="1.0">

        <xsl:template match= "/">
          <HTML>
          <HEAD>
                <TITLE> scenario library </TITLE>
          </HEAD>
          <BODY BGCOLOR="#66CC99">
          <H2> Scenario library</H2>
          <xsl:for-each select="SCENARIO-LIBRARY/SCENARIO">
                <UL>
                <LI><B><SPAN STYLE="font-style:italic">scenario-id :
                </SPAN></B></LI>
                <!-- SPAN-elementet binder HTML-element till XML-element eller
                attribut. När man binder ett HTML-element till ett XML-element eller
                –attribut visar HTML-elementet automatiskt innehållet i XML-
                elementet eller värdet på XML-attribut -->
                <xsl:value-of select="SCENARIO-ID"/> <BR/>
                <B><SPAN STYLE="font-style:italic">descriptive name :
                </SPAN></B>
                <xsl:value-of select="DESCRIPTIVE-NAME"/><BR/>
                <B><SPAN STYLE="font-style:italic">source : </SPAN></B>
```

```
                    <xsl:value-of select="SOURCE"/><BR/>
                    <B><SPAN STYLE="font-style:italic">nr of patterns :
                    </SPAN></B>
                    <xsl:value-of select="NR-OF-PATTERNS"/><BR/>
                    <B><SPAN STYLE="font-style:italic">description :
                    </SPAN></B>
                    <xsl:value-of select="DESCRIPTION"/><BR/>
                    <B><SPAN STYLE="font-style:italic">providing organization :
                    </SPAN></B>
                    <xsl:value-of select="PROVIDING-ORG"/><BR/>
                    <B><SPAN STYLE="font-style:italic">providing person :
                    </SPAN></B>
                    <xsl:value-of select="PROVIDING-PERS"/><BR/>
                    <B><SPAN STYLE="font-style:italic">contact email :
                    </SPAN></B>
                    <xsl:value-of select="CONTACT-EMAIL"/><P/>
                    </UL>
                </xsl:for-each>
                </BODY>
                </HTML>
            </xsl:template>
</xsl:stylesheet>
```

Så här ser sidan ut i Internet Explorer**:**



Figuren visar de två första SCENARIO-elementen (om man rullar nedåt ser man de resterande).

För andra punkten gäller:

<?xml version="1.0" encoding="UTF-8"?>

<!-- Filnamn: XslScenario2.xsl -->

<xsl:stylesheet

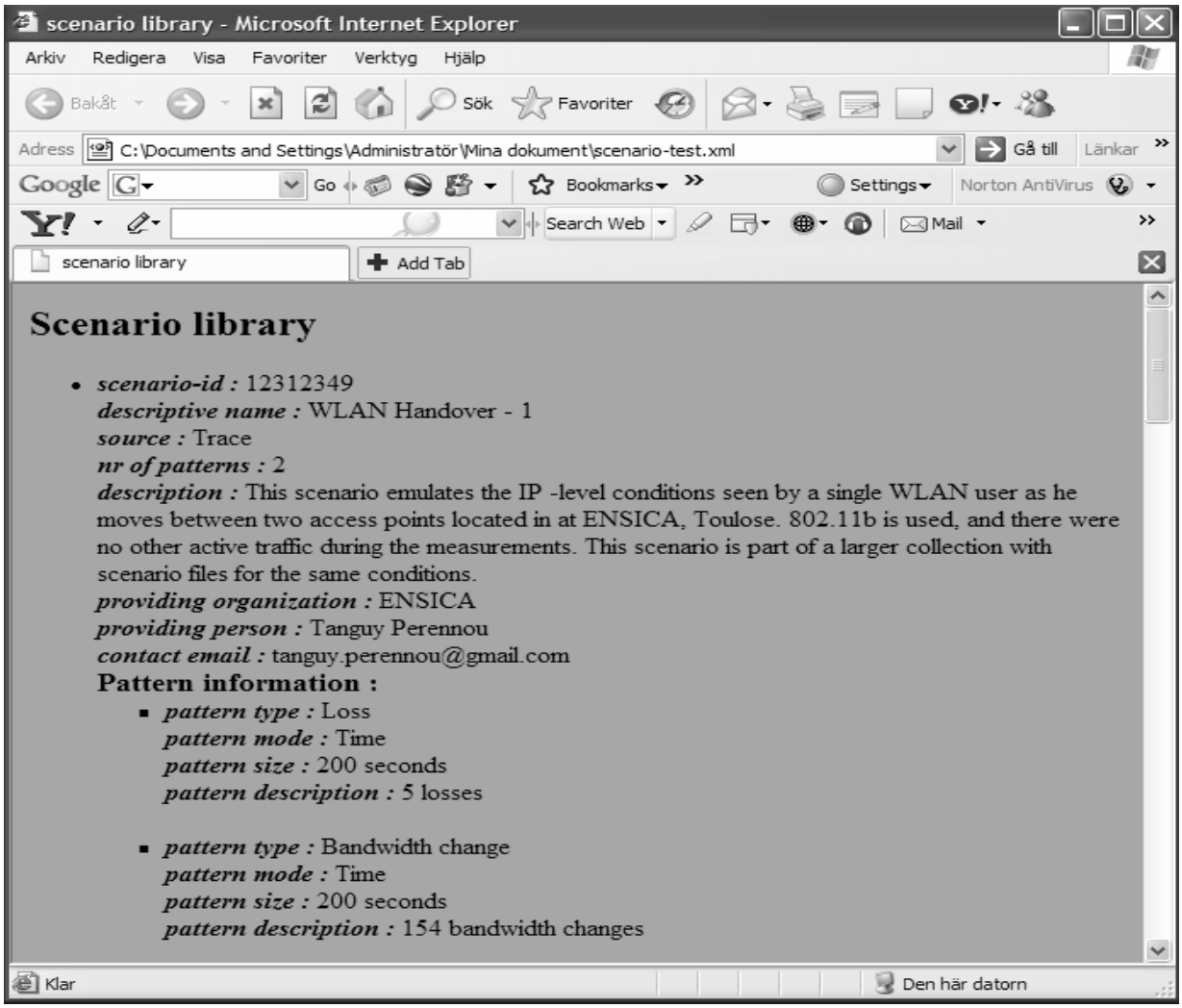
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

version ="1.0">

```
<xsl:template match= "/">
        <HTML>
        <HEAD>
                <TITLE> scenario library </TITLE>
        </HEAD>
        <BODY BGCOLOR="#66CC99">
        <H2> Scenario library</H2>
        <xsl:for-each select="SCENARIO-LIBRARY/SCENARIO">
                <UL>
                <LI><B><SPAN STYLE="font-style:italic">scenario-id :
                </SPAN></B></LI>
                <xsl:value-of select="SCENARIO-ID"/> <BR/>
                <B><SPAN STYLE="font-style:italic">descriptive name :
                </SPAN></B>
                <xsl:value-of select="DESCRIPTIVE-NAME"/><BR/>
                <B><SPAN STYLE="font-style:italic">source : </SPAN></B>
                <xsl:value-of select="SOURCE"/><BR/>
                <B><SPAN STYLE="font-style:italic">nr of patterns :
                </SPAN></B>
                <xsl:value-of select="NR-OF-PATTERNS"/><BR/>
                <B><SPAN STYLE="font-style:italic">description :
                </SPAN></B>
                <xsl:value-of select="DESCRIPTION"/><BR/>
                <B><SPAN STYLE="font-style:italic">providing organization :
                </SPAN></B>
                <xsl:value-of select="PROVIDING-ORG"/><BR/>
                <B><SPAN STYLE="font-style:italic">providing person :
                </SPAN></B>
                <xsl:value-of select="PROVIDING-PERS"/><BR/>
                <B><SPAN STYLE="font-style:italic">contact email :
                </SPAN></B>
                <xsl:value-of select="CONTACT-EMAIL"/><BR/>
```

```
<B><font size="4">Pattern information : </font></B><BR/>
<xsl:for-each select="PATTERN-INFO">
<UL type="square">
<LI><B><SPAN STYLE="font-style:italic">pattern
type : </SPAN></B></LI>
<xsl:value-of select="PATTERN-TYPE"/><BR/>
<B><SPAN STYLE="font-style:italic">pattern
mode : </SPAN></B>
<xsl:value-of select="PATTERN-MODE"/><BR/>
<B><SPAN STYLE="font-style:italic">pattern size
: </SPAN></B>
<xsl:value-of select="PATTERN-SIZE"/><BR/>
<B><SPAN STYLE="font-style:italic">pattern
description : </SPAN></B>
<xsl:value-of select="PATTERN-
DESCRIPTION"/><P/>
</UL>
</xsl:for-each>
</UL>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Det slutliga resultatet är att utmatningen innehåller alla data från alla SCENARIO-element i dokumentet. Så här ser sidan ut i Internet Explorer:



7 Resultat och rekommendationer

Som mål för uppsatsen hade vi, att utföra en uppgift som skall tillämpas med hjälp av XML och eftersom uppgiften kommer att ta upp några begrepp som har med XML att göra, hade vi ett extra mål och det är att presentera terminologi som förekommer i samband med XML.

Vi har i uppsatsen nått detta mål. Vi har utfört uppgiften samt förklarat syntaxen och semantiken som vi har använt i uppgiften. Vi har även förklarat de flesta begrepp som förekom i samband med uppgiften, till exempel vad ett XML-schema är och vad syntaxen är som användes för att kunna skriva ett XML-schema. Vi har faktiskt inte bara presenterat terminologin utan vi har presenterat XML mer generellt med exempel och så vidare.

Uppgiften delades in i deluppgifter som vi kan se nedan och som vi redan sett i kapitel 3:

1. Ta fram ett förslag på ett XML-schema för dataformatet vi fick i uppgiften.
   Vi har nått målet med den deluppgiften och lösningen finner du i sista avsnittet i kapitel 5.

2. Ta fram ett förslag på en XML-fil inehållande exempelinformation som följer begränsningarna man har i XML-schemat.
   Målet är nått efter att vi har gått igenom syntaxen för XML och som förslag till denna deluppgift kan du se sista avsnittet i kapitel 5.

3. Skapa en XSLT-fil som ger en översiktlig presentation av informationen i XML-filen på en webb-sida.
   Efter att vi har gått igenom XSLT-tekniken i kapitel 6 och gett förslag på en XSLT-fil har vi även nått målet med denna deluppgift.
4. Skapa en XSLT-fil som ger en detaljerad presentation av informationen i XML-filen på en webb-sida.
   I sista avsnittet i kapitel 6 kan man se en XSLT-fil som överensstämmer med det som efterfrågas här och då har vi på det sättet nått målet med denna deluppgift också.

Efter att vi har nått vårt mål vill vi nu ge några rekommendationer som kan utnyttjas av dem som tänker skapa egna XML-dokument. Dessa kan man se i avsnitt 7.1.
Vi sammanfattar även det vi skrivit och det vi har lärt oss om XML genom att ange kortfattat vilka för- och nackdelar XML har. Det kan man ta del av i sista avsnittet i det här kapitlet.

7.1 Skapa egna XML-dokument

Att skapa egna XML-dokument ger betydligt mer förståelse om XML-syntaxen. Om man skapar egna XML-dokument är det bra att tänka på följande viktiga punkter.

- Att välja bra elementnamn. Det är viktigt att välja ett sätt att namnge element och hålla sig till det. Man bör välja namn som är självförklarande och lätta att komma ihåg , eftersom någon kan tänkas behöva ändra struktur eller innehåll i dokumenten ifall specifikationerna eller kraven på dokumenten ändras under dess livstid.
- Att ha bra struktur bidrar till ett lättöverskådligt dokument där det är enkelt att se vad som hör ihop.
- Att välja en lämplig mängd element. Det är svårt att avgöra hur datat skall brytas ned. Ibland kan det bli helt fel om datat inte bryts ned. Datum är ett bra exempel, eftersom det finns flera sätt att skriva ett datum blir det svårt att tolka följande element:

<DATUM> 070504 </DATUM>, vilket kan betyda antingen 7 maj 2004 eller 4 maj 2007.

För att undvika misstolkning kan man bryta ned datumet på följande sätt:

```
<DATUM>
    <ÅR> 2007 </ÅR>
    <MÅNAD> 05 </MÅNAD>
    <DAG> 04 </DAG>
</DATUM>
```

Undvik däremot att dokumentet bryts ned för mycket vilket bidrar till att dokumentet blir oöverskådligt och svårläst. En tumregel för hur långt det skall brytas ned är att om datat behöver vara åtkomligt, kan det vara lämpligt att märka upp den för sig.

- Rätt eller fel?

Det finns inget som säger att dokumenten man skriver är rätt eller fel, bara välutformat/icke välutformat och giltigt/icke giltigt. XML-dokument kan skrivas på vilket språk som helst. Det som är viktigt att tänka på innan man skriver XML-dokument är att följa de regler som finns, till exempel hur elementnamn skall skrivas, och försöka anpassa sig efter situationen.

7.2 För- och nackdelar med XML

Fördelar

- ❖ XML är en öppen standard. Med det menas att ingen kan ta betalt för att du använder det, eller för att du skriver program som använder det. Eftersom XML är en standard är många utvecklare villiga att ge stöd åt det. Detta medför att XML används av många människor.
- ❖ XML-dokument lagras som rena textfiler och kan läsas av alla texteditorer.
- ❖ Informationen är skild från presentationen. Man behöver bara använda sig av en enda källa och kan sedan utifrån detta skapa valfria dokument samt bestämma presentationssätt. Informationen ändras aldrig, bara själva utseendet.
- ❖ XML hanterar stora informationsmängder på ett strukturerat, lättläst och framtidssäkert sätt
- ❖ XML är plattformsoberoende vilket möjliggör enkelt utbyte av information mellan datorsystem.
- ❖ Enkelt att bygga ut.

Nackdelar

- XML är en mycket omfattande teknik som innehåller flera delar, vilket gör att det är komplicerat och resurskrävande att använda.
- Många tekniker inom XML är ej färdiga standarder.
- Det kan vara svårt att komma överens om en gemensam taggstruktur inom en specifik bransch.

8 Summering av uppsatsen

8.1 Vad har vi lärt oss?

Vi kom i kontakt med XML när vi läste en kurs i datakommunikation och det var första gången vi hörde talas om XML. I den kursen har vi bara skapat ett litet XML-dokument och för att skapa detta XML-dokument har vi följt ett exempel vi fick av läraren. VarförXML-dokument skrivs på det sättet var inte riktigt klart för oss, men efter att vi har gjort denna uppsats har vi nu mer förståelse om vad XML är och hur man skriver XML-kod och varför den skrivs på det sättet. Vi har även lärt oss nya begrepp som vi aldrig hört talas om förut som till exempel XSLT och SGML.

8.2 Vad har varit svårt/lätt?

Det som var svårt från början var att välja ett ämne till examensarbetet. Sen efter att vi har valt ämnet var det svårt att begränsa sig inom ett visst område.

Hur man skriver ett så stort arbete var inte heller lätt för oss eftersom vi inte har sysslat med ett liknande förut.

Problemet var att vi inte hade tillräckligt med kunskap om ämnet XML från början vilket gjorde det svårt för oss att utföra uppgiften, som var en viktig del i uppsatsen. Det första vi fick göra då att lägga ner tid på att läsa in oss på vad XML är och hur den skrivs. Det var lite förvirrande till en början, för det fanns så många nya termer och begrepp att sätta sig in i.

8.3 Hur lång tid har det tagit?

Vi började skriva uppsatsen i mitten av februari och nu är vi i slutet av maj så det har nästan tagit mer än tre månader att jobba med uppsatsen. Men under den tiden har vi även läst andra kurser

parallellt med uppsatsen, vilket också ledde till att vi inte jobbade hela veckan med den. Vi arbetade 8-9 timmar om dagen, 4-5 dagar i veckan. Den tiden som var avsatt för uppsatsen var 10 veckors heltidsarbete, men fyra till fem dagars heltidsarbete i ca tre månader innebär att vi har jobbat mer är den avsatta tiden.

8.4 Förslag på framtida studier

Under skrivandeprocessen har idéer väckts om vidare studier inom ämnet.

- XML:s användningsområde?
- Vad har XML för skillnader från HTML? Några skillnader har vi nämnt i uppsatsen men det finns andra saker som skiljer HTML från XML som vi tänkte skriva om vi hade haft mer tid.

Vi fick med uppgiften även några deluppgifter som ansågs vara möjliga påbyggnader om vi har mer tid.

- Sortera informationen baserat på olika kategorier som användaren kan klicka på.
- Ta fram funktionalitet för att via ett webbgränssnitt söka efter specifika scenariofiler.
- Ta fram funktionalitet för att via ett webbgränssnitt lägga till information om nya scenarier.

Det skulle ha varit roligt att göra dem men tyvärr tiden räckte inte till. Det kan vara någonting som vi kan ha som framtida studier.

Referenser

[1] Berild, Stig (2004). XML Schema – en översikt [pdf]. Hämtad från http://www.skriver.nu/esociety/archives/XML%20Schema.PDF. (2007-04-08)

[2] Coyle, Frank P. (2002). XML, Web services, and the data revolution. Boston: Pearson Education,inc

[3] Deitel, Deitel, Nieto, Lin och Sadhu (2001). XML How to program. New Jersey: prentice-hall, Inc.

[4] Eriksson, Magnus (1999). XML-guiden [www]. Hämtat från http://www.acc.umu.se/~alpha/xml. (2007-04-05)

[5] Liljegren, Gustaf (2001). Vad är XML? [www]. Hämtat från http://www.xml.se/xml/vad.html. (2007-02-24)

[6] Liljegren, Gustaf (2004). XML –begreppen och tekniken. Lund :Studentlitteratur

[7] Marchal, Benoît (2002). XML genom exempel. Sundbyberg: Pagina

[8] Michael J. Young (2002). XML steg för steg, 2:a upplagan. Sundbyberg: Pagina

[9] Mikael Bonnie (2002). Rapport om XML [www]. Hämtat från http://www.df.lth.se/~mikaelb/xml/xml-rapport.html. (2007-05-06)

[10] Pitts-Moultis, N. & Kirk, C.(1998) XML Black Book, Coriolis Group, elektronisk upplaga

[11] Statskontoret (2002). Vad är XML?.

[12] Statskontoret (2002). Vad är XML-Familjen?.

[13] Statskontoret (2000). Vad är XML-Schema och XML-DTD*?*.

[14] Von Pepel, Eva. XML referensbok [www]. Hämtad från

http://vonpepel.com/ref/xml/valutform.html.(2007-04-04)

[15] Wikipedia. XML [www]. Hämtad från http://sv.wikipedia.org/wiki/Xml. (2007- 04-13)

[16] W3C XHTML. Introduktion till XML och XHTML [www]. Hämtad från http://www.isk.kth.se/kursinfo/6b4079/rep/xhtml/index.html. (2007-04-13)

[17] XML Sweden (2003). Anpassad utbildning [www]. Hämtad från http://www.xml.se/utbildning/anpassad.html. (2007-04-28)

[18] XML Sweden (2003). Standarder kring XML [www]. Hämtad från http://www.xml.se/xml/standarder.html#names. (2007-03-14)

[19] XSLT. Omvandla XML-filer med XSLT [www]. Hämtad från http://office.microsoft.com/sv-se/access/HA010345761053.aspx . (2007-04-05)

Ordlista

ANSI American National Standards Institute. Amerikansk standardiserinsorgan.

ASCII American Standard Code for Information Interchange. Är en teckenkodning som används för att representera bokstäver och andra tecken i datorer. Representerar endast engelska tecken. Finns även som Extended ASCII som rymmer alla västereuropiska tecken. Se även unicode.

Attribut Attribut används för att ge element olika egenskaper, till exempel ett unikt id. I ett XML-dokument sätt ett elements attribut i elementets starttagg.

DTD Document Type Definition. Beskriver ett XML-dokuments struktur. DTD:n definierar dokumentets tillåtna element

Element Varje informationsbärare i ett XML-dokument är ett element. Element består av märkord och innehåll, som kan vara text eller andra element. Består av en starttagg, en sluttagg och texten de innehåller.

Entitet En entitet representerar någon form av främmande data eller ersättningsinformation i ett XML-dokument, till exempel en textsträng.

Giltig Giltiga XML-dokument är välutformade XML-dokument som dessutom rättar sig efter ett visst schemaspråk, till exempel en DTD.

HTML HyperText Markup Language. Är ett märkspråk och webbstandard för strukturering av text, media och inbyggda objekt på exempelvis webbsidor.

ISO International StandardisationOrganization. Organisation för framtagandet av internationella standarder.

Märkord Märkord markerar ett elements början och slut. Skrivs vanligen inom ”< >”

SGML Standard Generalized Markup Language. Komplext metaspråk som specificerar regler för uppmärkning av information. En ISO-standard som ses som grunden till XML.

Unicode Är en teckenstandard som rymmer alla skrivtecken i världens alla skriftsystem: bokstäver, siffror, skiljetecken, matematiska symboler, m.m. Till exempel □□ (Shanghai), مكة (Mecka) och กรุงเทพฯ (Bangkok) hade inte gått att skriva utan Unicode.

Upp-märkning Se märkord.

UTF-8 Används för att representera text kodad i Unicode, som en sekvens av bytes, till exempel i textfiler.

Väl-utformade Ett XML-dokument som uppfyller vissa krav, bland annat att alla starttaggar har en sluttagg och att det finns endast ett rotelement.

W3C World Wide Web Consortium. Utvecklar rekommendationer för gemensamma standarder för nya tekniker inom Internet.

XML eXtensible Markup Language. Är ett universellt och utbyggbart märkspråk och en förenklad efterträdare till SGML

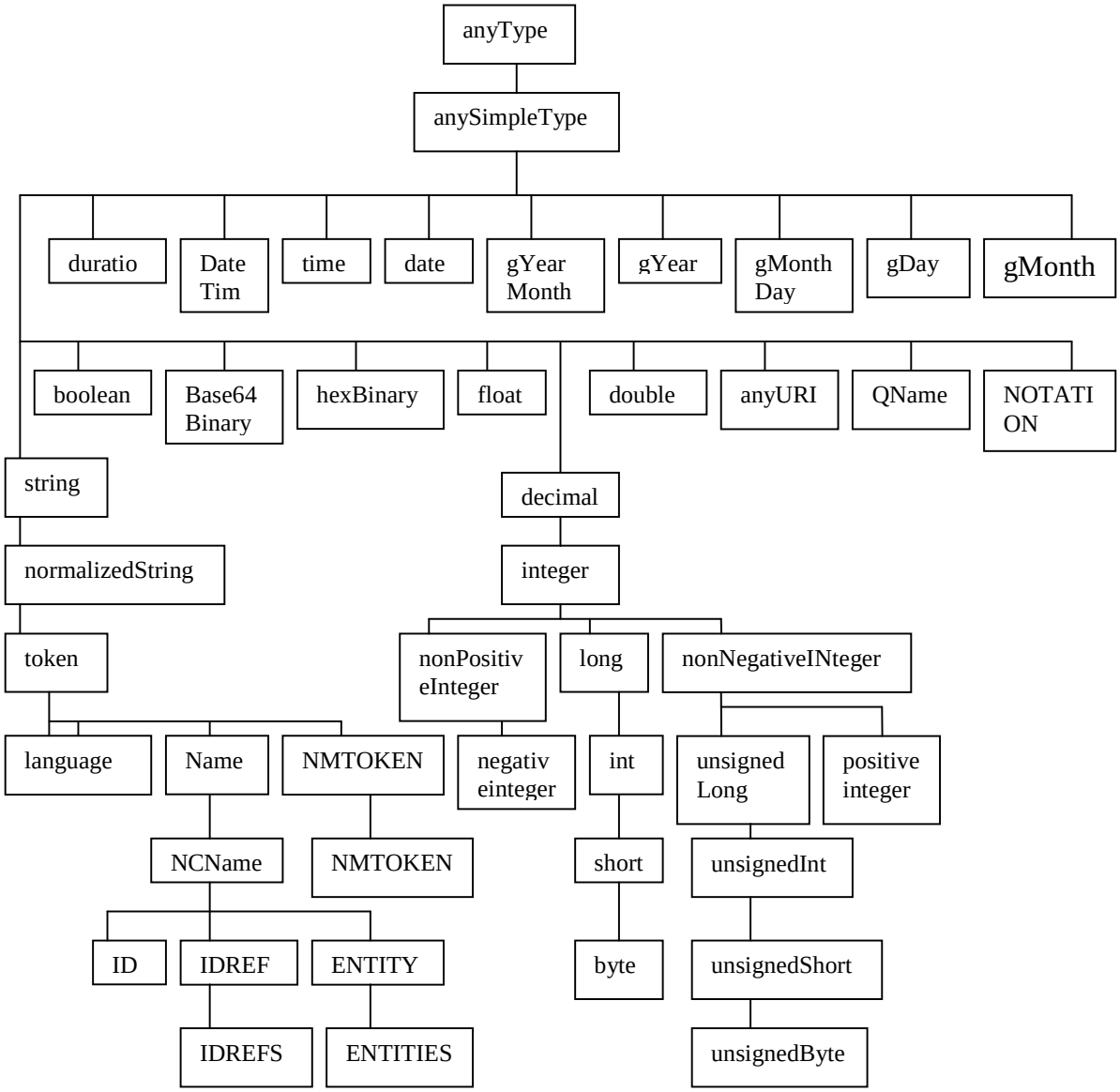
XML-Dokument Ett XML-dokument består av en prolog och ett rotelement. Prologen kommer först i ett XML-dokument och kan till exempel innehålla hänvisningar till DTD:er. I rotelementet ingår alla övriga element (märkord och innehåll).

XML- Ett XML-schema är ett nyare sätt att beskriva XML-dokument. Scheman är

schema uppbyggda som "normala" XML-dokument och förväntas ta över DTD:ns roll för att definiera ett dokuments struktur.

XSLT eXtensible Stylesheet Language Transformation. Teknik för att strukturera om XML-dokument.

Bilaga A – Inbyggda och härledda datatyper



Bilaga B – XML-dokument

<?xml version="1.0" encoding="UTF-8"?>

<!-- Filnamn:Scenariofilsinfo.xml -- >

<SCENARIO-LIBRARY>

<SCENARIO>

<SCENARIO-ID> 12312349 </SCENARIO-ID>

<DESCRIPTIVE-NAME> WLAN Handover – 1 </DESCRIPTIVE-NAME>

<DESCRIPTION> This scenario emulates the IP -level conditions seen by a single WLAN user as he moves between two access points located in at ENSICA, Toulose. 802.11b is used, and there were no other active traffic during the measurements. This scenario is part of a larger collection with scenario files for the same conditions. </DESCRIPTION>

<SOURCE> Trace </SOURCE>

<PROVIDING-ORG> ENSICA </PROVIDING-ORG>

<PROVIDING-PERS> Tanguy Perennou </PROVIDING-PERS>

<CONTACT-EMAIL> tanguy.perennou@gmail.com </CONTACT-EMAIL>

<NR-OF-PATTERNS> 2 </NR-OF-PATTERNS>

<PATTERN-INFO>

<PATTERN-TYPE> Loss </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 200 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 5 losses </PATTERN-DESCRIPTION>

</PATTERN-INFO>

<PATTERN-INFO>

<PATTERN-TYPE> Bandwidth change </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 200 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 154 bandwidth changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>

</SCENARIO>

<SCENARIO>

<SCENARIO-ID> 12312344 </SCENARIO-ID>

<DESCRIPTIVE-NAME> WLAN Handover – 2</DESCRIPTIVE-NAME>

<**DESCRIPTION**> This scenario emulates the IP -level conditions seen by a single WLAN user as he moves between two access points located in at ENSICA, Toulose. 802.11b is used, and there were no other active traffic during the measurements. This scenario is part of a larger collection with scenario files for the same conditions.</**DESCRIPTION**>

<**SOURCE**> Trace **</SOURCE>**

<**PROVIDING-ORG**> ENSICA </**PROVIDING-ORG**>

<**PROVIDING-PERS**> Tanguy Perennou </**PROVIDING-PERS**>

<**CONTACT-EMAIL**> tanguy.perennou@gmail.com </**CONTACT-EMAIL**>

<**NR-OF-PATTERNS**> 2 </**NR-OF-PATTERNS**>

<**PATTERN-INFO**>

<**PATTERN-TYPE**> Loss </**PATTERN-TYPE**>

<**PATTERN-MODE**> Time </**PATTERN-MODE**>

<**PATTERN-SIZE**> 200 seconds </**PATTERN-SIZE**>

<**PATTERN-DESCRIPTION**> 5 losses </**PATTERN-DESCRIPTION**>

</**PATTERN-INFO**>

<**PATTERN-INFO**>

<**PATTERN-TYPE**> Bandwidth change </**PATTERN-TYPE**>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 200 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 154 bandwidth changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>

</SCENARIO>

<SCENARIO>

<SCENARIO-ID> 12312344 </SCENARIO-ID>

<DESCRIPTIVE-NAME> WLAN Handover – 3 </DESCRIPTIVE-NAME>

<DESCRIPTION> This scenario emulates the IP -level conditions seen by a single WLAN user as he moves between two access points located in at ENSICA, Toulose. 802.11b is used, and there were no other active traffic during the measurements. This scenario is part of a larger collection with scenario files for the same conditions.
</DESCRIPTION>

<SOURCE> Trace </SOURCE>

<PROVIDING-ORG> ENSICA </PROVIDING-ORG>

<PROVIDING-PERS> Tanguy Perennou </PROVIDING-PERS>

<CONTACT-EMAIL> tanguy.perennou@gmail.com </CONTACT-EMAIL>

<NR-OF-PATTERNS> 2 </NR-OF-PATTERNS>

<PATTERN-INFO>

<PATTERN-TYPE> Loss </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 200 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 5 losses </PATTERN-DESCRIPTION>

</PATTERN-INFO>

<PATTERN-INFO>

<PATTERN-TYPE> Bandwidth change </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 200 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 154 bandwidth changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>

</SCENARIO>

<SCENARIO>

<**SCENARIO-ID**> 17612351 </**SCENARIO-ID**>

<**DESCRIPTIVE-NAME**> VoIP Jitter - 1</**DESCRIPTIVE-NAME**>

<**DESCRIPTION**> This scenario provides a jitter source that provides poisson distributed inter-packet delays. This scenario is part of a larger collection with scenario files for the same conditions. </**DESCRIPTION**>

<**SOURCE**> Analytical **</SOURCE>**

<**PROVIDING-ORG**> KaU </**PROVIDING-ORG**>

<**PROVIDING-PERS**> Johan Garcia </**PROVIDING-PERS**>

<**CONTACT-EMAIL**> johan.garcia@kau.se </**CONTACT-EMAIL**>

<**NR-OF-PATTERNS**> 2 </**NR-OF-PATTERNS**>

<**PATTERN-INFO**>

<**PATTERN-TYPE**> Bandwidth change </**PATTERN-TYPE**>

<**PATTERN-MODE**> Packet </**PATTERN-MODE**>

<**PATTERN-SIZE**> 100000 packets </**PATTERN-SIZE**>

<**PATTERN-DESCRIPTION**> 1 bandwidth changes </**PATTERN-DESCRIPTION**>

</**PATTERN-INFO**>

<**PATTERN-INFO**>

<**PATTERN-TYPE**> Delay change </**PATTERN-TYPE**>

<**PATTERN-MODE**> Time </**PATTERN-MODE**>

<**PATTERN-SIZE**> 100000 packets </**PATTERN-SIZE**>

<**PATTERN-DESCRIPTION**> 100000 delay changes
</**PATTERN-DESCRIPTION**>

</**PATTERN-INFO**>

</**SCENARIO**>

<**SCENARIO**>

<**SCENARIO-ID**> 17612351 </**SCENARIO-ID**>

<**DESCRIPTIVE-NAME**> VoIP Jitter – 2 </**DESCRIPTIVE-NAME**>

<**DESCRIPTION**> This scenario provides a jitter source that provides poisson distributed inter-packet delays. This scenario is part of a larger collection with scenario files for the same conditions. </**DESCRIPTION**>

<**SOURCE**> Analytical **</SOURCE>**

<**PROVIDING-ORG**> KaU </**PROVIDING-ORG**>

<**PROVIDING-PERS**> Johan Garcia </**PROVIDING-PERS**>

<**CONTACT-EMAIL**> johan.garcia@kau.se </**CONTACT-EMAIL**>

<NR-OF-PATTERNS> 2 </NR-OF-PATTERNS>

<PATTERN-INFO>

<PATTERN-TYPE> Bandwidth change </PATTERN-TYPE>

<PATTERN-MODE> Packet </PATTERN-MODE>

<PATTERN-SIZE> 100000 packets </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 1 bandwidth changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>

<PATTERN-INFO>

<PATTERN-TYPE> Delay change </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 100000 packets </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 100000 delay changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>

</SCENARIO>

<SCENARIO>

<SCENARIO-ID> 27412454 </SCENARIO-ID>

<DESCRIPTIVE-NAME> Signalling traffic - 1
</DESCRIPTIVE-NAME>

<DESCRIPTION> This scenario emulates the traffic charcteristics on an IMS signalling link. This scenario is part of a larger collection with scenario files for the same conditions. </DESCRIPTION>

<SOURCE> Simulations </SOURCE>

<PROVIDING-ORG> TietoEnator </PROVIDING-ORG>

<PROVIDING-PERS> Karl-Johan Grinnemo </PROVIDING-PERS>

<CONTACT-EMAIL> karl-johan.grinnemo@tietoenator.se
</CONTACT-EMAIL>

<NR-OF-PATTERNS> 3 </NR-OF-PATTERNS>

<PATTERN-INFO>

<PATTERN-TYPE> Loss </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 2000 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 7 losses </PATTERN-DESCRIPTION>

</PATTERN-INFO>

<PATTERN-INFO>

<PATTERN-TYPE> Bandwidth change </PATTERN-TYPE>

<PATTERN-MODE> Packet </PATTERN-MODE>

<PATTERN-SIZE> 2000 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 567 bandwidth changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>

<PATTERN-INFO>

<PATTERN-TYPE> Delay change </PATTERN-TYPE>

<PATTERN-MODE> Time </PATTERN-MODE>

<PATTERN-SIZE> 2000 seconds </PATTERN-SIZE>

<PATTERN-DESCRIPTION> 4932 delay changes
</PATTERN-DESCRIPTION>

</PATTERN-INFO>
</SCENARIO>

</SCENARIO-LIBRARY>