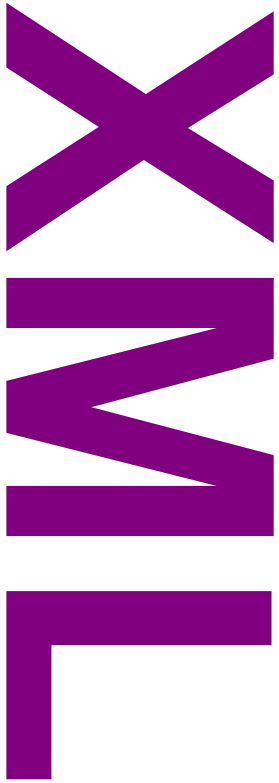
En skonsam men effektiv introduktion



Lars Celander

Om denna bok ...

I enlighet med den nya ekonomins logik, kan den här boken laddas ner gratis över nätet. Boken var ursprungligen tänkt att säljas till självkostnadspris. När folk själva laddar ner och skriver ut boken, blir tillverkningskostnaden noll. Är tillverkningskostnaden noll, skall följaktligen också priset vara noll. Boken finns därmed upplagd gratis i form av en PDF-fil på adressen http://www.raserbaden.com.

Det är fritt fram att distribuera boken vidare, både i elektronisk form och i pappersform. Boken skall dock alltid distribueras vidare i sin helhet och med copyright-notisen intakt.

Boken kommer också att utvecklas som bok betraktat. Läsare är mycket välkomna att komma in med förslag på förbättringar av den. Det finns alltid utrymme för att göra den ännu bättre.

Lars Celander

E-post: lars@raserbaden.com

INNEHÅLL

Förord

Denna bok är tänkt som en grundläggande introduktion till XML. Att sätta sig in i vad XML är, och inte är, kan vara besvärligt. XML är helt enkelt inte så enkelt som det ibland ges ut för att vara.

Det finns många XML-böcker på marknaden. Tyvärr är många av dessa snabb-producerade för att slå mynt av intresset runt XML. Ofta är de tjockare än vad de är bra. De finns också många enklare broschyrer, som försöker förklara XML. Dessa brukar vara väl ytliga för att fungera som något mer än en aptitretare. Vad som är svårt att hitta, är böcker med "smält", sorterad och strukturerad information. Alltså böcker där man med en måttlig läsningsansträngning kan få en hyfsat kvalificerad förståelse för ämnet.

Syftet med boken är alltså att göra XML mer lättillgängligt och lättbegripligt. Avsikten är inte att "sälja" XML, dvs. att tala om hur fantastiskt bra det är och hur det kan lösa alla världens problem. Sådana texter är mest tröttsamma och barnsliga. Istället är avsikten att på ett kunnigt, nyanserat och pedagogiskt sätt, ge folk en god uppfattning om vad XML i praktiken innebär, på gott och på ont. Efter att ha läst boken skall man ha tillräcklig förståelse av XML för att i stora drag kunna lägga upp vettiga tillämpnings-strategier.

1 Inledning

Internet och webben har slagit igenom och nått en i det närmaste total acceptans. Alla använder det. Detta är helt fundamentalt. För första gången i informationsteknikens historia, har vi ett sätt att kommunicera på som når alla.

Internet bygger på ett litet antal rätt enkla standarder. TCP/IP talar om hur det grundläggande nätverket fungerar. Standarder som smtp och http sköter överföring av enkla textmeddelanden och länkning mellan dem. HTML beskriver hur formaterade sidor ser ut. Dessa standarder är alltså vad vi har att spela med idag. De kan göra en hel del, men långt ifrån allt.

Enklaste sättet att beskriva XML (eXtensible Markup Language), är som en fortsättning på HTML. Beskrivningen är inte helt fel, men XML är mycket mer än så.

Ur teknisk synvinkel kan man beskriva XML som ett generellt och standardiserat sätt att representera strukturerad information. Varje gång man vill lagra eller överföra information, typiskt då strukturerad information, har man nytta av XML. XML är helt klart en central del av den framtida webben. Troligtvis kommer det att användas en hel del även utanför egentliga web-applikationer.

XML tas fram av W3C (World Wide Web Consortium), de som har hand om det mesta vad gäller webben. XML stöds också av alla viktiga aktörer på marknaden. Trovärdigheten är därmed total. Man kan därför mycket lugnt satsa på XML.

Den stora nackdelen med XML just nu, är att det fortfarande är under utveckling. Även om utvecklingstakten är i det närmaste febril, så är alltså ännu inte alla bitarna på plats.

2 Bakgrund till XML

XML kommer ur SGML (Standard Generalised Markup Language). SGML är en ISO-standard (ISO 8879) och blev klar som ISO-standard 1986.

SGML är inriktat på sofistikerad dokumenthantering. Huvudskälet till att man gjorde en standard, var att man ville uppnå leverantörsoberoende. SGML:s huvudsakliga användningsområde är för kvalificerad teknisk dokumentation inom stora företag. Kundbasen domineras av militärindustrin.

Vad SGML tillhandahåller är en syntax för att representera innehållet i ett dokument, ett slags filformat helt enkelt. SGML definierar också en syntax för att skriva så kallade DTD:er (Document Type Definitions). En DTD beskriver regler för vad ett SGML-dokument får innehålla.

SGML definierar inget utseende på dokumentet. Detta sköts av DSSSL (Document Style and Semantics Specification Language, uttalas "dissel", ISO 10179). HyTime (Hypermedia/Time-based Structuring Language, ISO 10744) är en standard för hyperlänkar och multimedia. HyTime använder SGML och är tänkt att användas ihop med SGML.

SGML är en komplex standard. Den har mängder med kluriga och mer eller mindre obskyra finesser. Ingen använder hela standarden. SGML kräver mycket kompetens för att använda, vilket gör att kostnaden blir hög. Kundbasen minskar och leverantörerna blir färre och dyrare.

DSSSL och HyTime är också komplexa standarder. Speciellt DSSSL är extremt komplext. Ett komplett LISP-baserat programmeringsspråk är bara början. HyTime är mer harmlöst, mest utflippad teknikglädje i största allmänhet. Varken DSSSL eller HyTime har lyckats få något nämnvärt genomslag på marknaden. Ett fåtal implementationer har gjorts av några tappra entusiaster.

När webben skapades var SGML etablerat, i alla fall i vissa kretsar. Man snodde helt enkelt spridda delar från SGML och gjorde ett gammalt hederligt "hack" som man kallade HTML (Hyper-Text Markup Language). HTML är formellt sett en SGML-applikation. En av de grundläggande principerna i SGML, separationen mellan innehållet och presentationen av innehållet, struntade man helt rått i. HTML blandar friskt innehåll och presentation.

XML är ungefär samma sak. Man tog det bästa från SGML och kastade bort det som folk inte i praktiken har använt. En upprensning helt enkelt. Samtidigt gör man en del förändringar för att göra det mer web-anpassat. Men grundkoncepten och terminologin från SGML, är i allt väsentligt intakt.

Man kan göra web-anpassningen utan att göra upprensningen. Det har man gjort, resultatet kallas för Web-SGML, och finns som ett tillägg till SGML-standarden. Web-SGML har inte gjort mycket väsen av sig och kan i regel ignoreras.

Det som hindrar webbens vidare utveckling idag, är HTML. HTML sysslar bara med att beskriva web-sidor. Som språk för sidbeskrivning är det allmänt uselt, troligen det sämsta som överhuvudtaget finns för professionellt användande. Bristerna i HTML, och ett oändligt tricksande för att komma runt de bristerna, har givit upphov till en hel yrkeskår. Att det som sidbeskrivningsspråk är dåligt, är inte så fruktansvärt allvarligt. Det centrala och det fantastiska med HTML, är att alla använder det. Den avgörande begränsningen med HTML är att det sysslar med web-sidor och enbart med web-sidor. Vill man överföra generell information, alltså inte bara web-sidor, måste man slänga ut

HTML och skapa ett helt nytt språk med en helt annan struktur. Tillbaka till ritbordet alltså.

Vad som behövs är alltså ett generellt sätt att representera och strukturera ett innehåll. Språket måste också vara modulärt och utvecklingsbart, det gick inte att fördefiniera vad det skulle klara av.

Går man tillbaka till rötterna för HTML, till SGML, hittar man ungefär vad man söker. Tyvärr hade ju SGML ett grundmurat rykte som ett paradis för gurus och tyvärr behövdes det en del ändringar innan det dög för webben, men grundidéerna i SGML var av friskt kärnvirke.

XML, liksom det mesta som har med webben att göra, sköts av W3C (World Wide Web Consortium). ISO (International Standards Organisation) är helt överspelat, tempot på webben är alldeles för högt. W3C är egentligen bara en privat organisation, fristående från stater och regeringar. W3C har helt enkelt den status och det inflytande som medlemmarna och omgivningen ger det.

3 Avskalad XML

Allt är dokument

XML pratar i termer av "dokument". Begreppet dokument används alltså även när man inte åsyftar dokument ur den invanda meningen, som något som kan läsas av människor och som kan skrivas ut på papper. Ur XML:s synpunkt är ett dokument bara en klump med information, punkt slut. Inget alls är sagt om innehåll eller användning. Innehållet, strukturen och användningssättet kan vara vad som helst.

Dokumentet är i grunden ett logiskt begrepp. Det kan implementeras på många olika sätt. Det existerar oftast i form av en fil men det kan lika gärna existera i form av ett objekt i en objektdatabas eller som några tabeller i en relationsdatabas.

Fundamentalistisk XML

Ett XML-dokument består av ett eller flera element. Element kan innehålla text eller andra element. Elementen bildar en hierarki. Man brukar tala om att dokumentet har en trädstruktur.

Ett XML-dokument har alltid exakt ett element på högsta nivån, det s.k. dokument-elementet eller rot-elementet. Alla andra element ligger i trädstrukturen under det elementet. Dokumentets typ är detsamma som rot-elementets typ. Terminologin innebär alltså att träd växer neråt och att roten är högsta punkten. Botanister må rysa hur mycket de vill.

Ett element består av en start-tagg, ett innehåll och en slut-tagg. Ordet "tagg" är något av slang i branschen och är en lätt vanvördig översättning av engelskans betydligt vettigare benämning "tag", ungefär "etikett". En mera korrekt översättning är att säga "märkord".

En start-tagg består av elementnamnet mellan "<" och ">"-tecken. En slut-tagg består av samma elementnamn mellan "</" och ">"-tecken.

```
<NAMN>
   Detta element innehåller text samt två element.
   <FÖRNAMN>James</FÖRNAMN>
   <EFTERNAMN>Bond</EFTERNAMN>
</NAMN>
```

I princip kan man stanna här. Man behöver inte alls bli mer komplicerad än så här. I många sammanhang väljer man också att stanna här. Man hittar på sina märkord och draperar dem i vinkelparenteser. Enkelt, rent och flexibelt. Med den tillgång till utmärkta XML-verktyg som finns idag, är det också mycket kostnadseffektivt.

Minimal XML

XML må vara en radikalt bantad version av SGML, men XML är fortfarande en rätt lönnfet historia. Det finns faktiskt kvar en del från SGML som inte är så särskilt smart eller så nödvändigt, i synnerhet inte för enkla applikationer.

Minimal XML går ett steg vidare, det är XML avskalat till sin absoluta kärna. Minimal XML finns inte ännu som standard, bara som ett inofficiellt förslag. Namnet skall ses mera som en benämning på en inriktning eller som en attityd. Kanske blir det någon gång formaliserat som en W3C-rekommendation.

Typisk användning

XML handlar om att strukturera och kommunicera information. XML är inte bundet till något visst applikationsområde eller till någon viss typ av problem. XML kan därför användas i väldigt många olika sammanhang.

Det finns i princip två sätt att använda XML; något som bara är tänkt att läsas av datorer eller något som (också) skall kunna läsas av människor.

Om ett XML-dokument används för att skicka meddelanden mellan applikationer, fungerar det som en sorts middleware på en rätt hög abstraktionsnivå. Det här upplägget är mycket användbart på mellannivån i en client/server-arkitektur. Här brukar det handla om rätt enkla XML-dokument.

Om ett XML-dokument skall läsas av en människa, hamnar man i princip i en klassisk dokumenthantering à la SGML. Dessa dokument brukar vara komplexa, bland annat måste de innehålla information om hur de skall presenteras på en bildskärm eller på ett papper. I så fall behövs något som kallas för stilmallar. Skall dokumentet bara läsas av datorer, behövs inga stilmallar.

Man kan också behöva definiera regler för struktur och innehåll i ett dokument, till exempel när man vill tvinga dokument att ha en viss struktur eller ett viss innehåll. Vet man att dokumenten alltid skapas korrekt, till exempel om de skapas automatiskt av en applikation, behöver man kanske inte kontrollera att reglerna efterlevs.

Man kan också skilja på användningssätt, där dokumenten har olika förväntad livslängd. Meddelanden mellan applikationer brukar innebära att dokumenten har kort livslängd, kanske några sekunder. Klassisk dokumenthantering innebär ofta att dokumenten har lång livslängd, från veckor och månader ända upp till flera decennier.

Det finns alltså en extremt stor spridning i hur man använder XML, från de enklaste applikationerna till de mest komplexa. Ändå är det samma grundstandard som man använder.

XML har en viss koppling till Java. Java är ju det programmeringsspråk som har mest "mindshare" i websammanhang. XML och Java passar mycket bra ihop tekniskt. Ett kanske viktigare skäl för att de brukar användas tillsammans är att de bägge fokuserar flyttbarhet och att de funktionellt kompletterar varandra. Java ger flyttbara applikationer, XML ger flyttbara data. Jon Bosak från Sun yttrade det nu klassiska "XML gives Java something to do". Kopplingen skall dock inte överdrivas, XML kan implementeras i alla programmeringsspråk.

Den stora semantiska oredan ...

Det stora problement med XML är att komma överens om hur märkorden skall se ut, om vilka "vokabulärer" man skall ha och om vilken semantik dessa ska ha. Om detta är XML helt tyst. XML tillhandahåller bara en standardiserad syntax, inget mer. Man kan likna det vid att tillhandahålla ett alfabet, inte ett ordförråd eller ett helt språk.

Problemet är i grunden inte tekniskt, det är sociologiskt. Problemet ligger i att komma överens. Kan man själv bestämma vokabulären, till exempel för en intern applikation i ett företag, är det inget större problem. Där det däremot inte finns ett tydligt ledarskap, där är problemet betydligt värre.

Vad som händer är givetvis att folk sätter igång och publicerar vokabulärer med stor frenesi, i hopp om att just deras av omvärlden skall bli accepterade som de facto standard. XML-världen är fullt av sådant. I många fall lyckas det också, i alla fall de mera seriösa förslagen. I andra fall händer det ingenting. Omvärlden gäspar, sorterar det plikttroget i mappen "XML-applikationer" och glömmer sedan bort det. Man bör ha en mycket Darwinistisk syn på vokabulärer.

De vokabulärer som överlever kommer att vara dåligt integrerade. Olika grupper från olika branscher kommer att beskriva liknande saker på olika sätt. Detta är inte så mycket att göra något åt. Man kan streta emot och kräva bättre samordning, man kan stöna över kostnaderna de medför, men i slutändan kommer problematiken att kvarstå. Det handlar egentligen inte om någon defekt hos XML. Det är helt enkelt en egenskap hos vårt samhälle. Man bör egentligen se det som ett utslag av "pluralism" och inte av "bristande samordning". I slutändan troligen en bra egenskap.

Oredan innebär samtidigt en potentiell marknad. Den som kan lindra oredan, fyller ett behov. Den som lindrar oredan får genom sitt ledarskap ett mått av makt, en makt som givetvis kan utnyttjas affärsmässigt. Hur marknaden kommer att utvecklas, vem eller vilka som blir de dominerande aktörerna, återstår att se. Man kan också ifrågasätta om marknaden är kommersiellt bearbetningsbar i någon större utsträckning. Ett behov är inte alltid samma sak som en marknad.

Den vettigaste vägen framåt för ett företag, är troligen att nöja sig med att i görligaste mån hålla ordning i sin egen röra och att avvakta vad som händer i omvärlden. Filosofin för ett företag blir då att definiera en egen uppsättning vokabulärer som företagsstandard och att använda dessa för att koppla ihop sina interna system. Allteftersom omvärlden hittar på nya och förmodligen illa integrerade vokabulärer, bygger man bryggor till dessa.

4 XML med farträndер

Om förra kapitlet gav en känsla för kärnan i XML, så är det nu dags att gå in på olika finesser. Det här kapitlet är en listning av olika möjliga farträndер man kan använda i samband med XML. Det kan skippas vid en första översiktlig genomläsning.

Attribut

Ett element kan ha attribut, utöver det ordinarie innehållet. Attributen läggs inuti elementets start-tagg. Attributen definieras som attributnamn och attributvärde.

```
<TIDNING Typ="Dilbert">
  ...
</TIDNING>
```

Ett element kan också vara tomt, dvs. det finns inget innehåll mellan start-taggen och slut-taggen. Slut-taggen kan då elimineras och man skriver då start-taggen mellan "<" och "/>". Enda innehållet i ett sådant element, blir då dess attribut.

```
<DATUMSTÄMPEL Datum="2000-01-01"/>
```

Den information som kan läggas i form av ett attribut, kan också läggas i form av ett element. Attribut tillför i princip inget. Attribut kan heller inte nästlas så som element kan (ett attribut kan inte innehålla andra attribut). Syftet med attribut-konstruktionen var en gång i tiden att man ville kunna skilja på data och på meta-data. Den distinktionen fungerar om man på förhand vet vilket applikationsområde man skall arbeta i och om det inom området finns en vedertagen skillnad på data och på meta-data. Generellt sett är distinktionen mindre lyckad och användningen av attribut kan ifrågasättas. Attribut-konstruktionen är helt enkelt ett arv från SGML, där den fyller en naturlig roll inom klassisk dokumenthantering.

Entiteter

En entitet är ett namngivet block med information av något slag. En entitet kan till exempel användas som förkortning på något ofta använd text-sträng. En annan användning, är för att bygga upp ett dokument av ett antal moduler.

Användningen av entiteter styrs i slutändan enbart av vad som är praktiskt och bekvämt. Entiteter uttrycker inget som inte kan uttryckas som vanligt inuti element.

Entiteter måste definieras. Definitionen innebär att entitetsnamnet associeras med ett informationsblock. Informationsblockets innehåll kan ges direkt i entitets-definitionen. Entiteten kallas då för en intern entitet.

```
<!ENTITY EntitetsNamn "en massa tjusig text">
```

En definierad entitet kan sedan anropas med entitetsnamnet mellan "&" och ";"-tecken:

```
<stycke>
text &EntitetsNamn; mera text
</stycke>
```

Informationsblockets innehåll kan också definieras till att vara innehållet i en extern fil. Entiteten kallas då för en extern entitet.

```
<!ENTITY EntitetsNamn SYSTEM "text.xml">
```

Här antages filen vara tillgänglig på det lokala systemet, antagligen i en mapp som applikationen känner till. En annan variant är att använda någon typ av allmänt känd adress-mekanism för att peka ut filen.

```
<!ENTITY EntitetsNamn PUBLIC "http://www.foretaget.com/text.xml">
```

Filen måste inte nödvändigtvis vara i form av text. Det kan vara något som inte är tänkt att tolkas som XML-kod, till exempel en bild på ett binärt format. Innan man kan uttrycka att en entitet har något särskilt format, måste det formatet definieras.

```
<!NOTATION jpeg "någon förklarande text till notationen">
<!ENTITY EntitetsNamn SYSTEM "bild.jpeg" NDATA jpeg>
```

En entitet som inte är XML, kallas för en "unparsed" entitet, dvs. en entitet som en XML-tolk bara noterar att den finns, men som den inte försöker tolka.

Entitetsdefinitioner görs formellt i DTD:n (för mer om DTD:er, se nedan och kapitel 5) för dokumentet. DTD-snutten kan ligga direkt i dokumentet, den kan också definieras i en extern DTD, vilket är det normala.

Det finns också några fördefinierade entiteter som inte behöver definieras. Syftet med dessa är att ersätta de tecken som XML använder för eget bruk:

```
&gt;       >
&lt;       <
&amp;      &
&apos;     '
&quot;     "
```

Entitetsbegreppet är alltså XML:s egen mekanism för att fysiskt bygga upp ett dokument på ett strukturerat och modulariserat sätt. Det finns färdiga produkter på marknaden som hjälper till med hanteringen av entiteter. I synnerhet brukar det vara en viktig del av mera kvalificerade verktyg för dokumenthantering. Funktionaliteten brukar kallas för "entity management".

Distinktionen mellan logisk och fysisk struktur är olika värdefull i olika sammanhang. I sammanhang där XML-dokumenten är färskvara ger det en implementationsmässig flexibilitet. I sammanhang där XML-dokumenten är mer långlivade, får man se upp med vad man håller på med. Ofta vill man i dessa sammanhang också ha en versions- och variant-hantering av dokumenten. Versionshantering av dokument, sammansatta av flera olika delar, kräver en hel del av både användaren och av versionshanteringssystemet.

Man behöver inte nödvändigtvis alltid underkasta sig entitets-begreppet för att flexibelt bygga upp ett dokument. Ett XML-dokument är ett XML-dokument oavsett

vilka metoder man använder för att plocka ihop delarna. Ett annat alternativ till entiteter är XML Inclusions (se nedan).

Ett XML-dokument har inte alltid en DTD. Om det inte har det, kan det heller inte använda andra entiteter än de fördefinierade.

Processing Instructions

En processinstruktion består av någon text mellan "<?" och "?>"-tecken. Tanken är att de skall användas för att styra hur dokumentet skall behandlas i något avseende. En processinstruktion är kopplad till någon viss applikation. Vilken applikation det är, pekas ut av den första ordet (tecknen fram till första mellanslaget) i instruktionen, det så kallade målet för processinstruktionen. Den applikationen måste på förhand veta om, hur innehållet i instruktionen skall tolkas. Andra applikationer ignorerar instruktionen.

Processinstruktioner är mycket behändiga för diverse pyssel som inte passar in någon annanstans. Exempelvis använder XML en processinstruktion för att definiera vilken XML-version som används:

```
<?xml version="1.0"?>
```

Noteras bör att XML reserverar mål som börjar på "xml" för eget bruk. Ett annat bra exempel på användning av processinstruktioner, är för inbäddning av PHP-skript (PHP är populärt på Linux-plattformar och har ungefär samma funktionalitet som Microsofts ASP):

```
<?php echo "Hello, world!"; ?>
```

I det här exemplet lägger skriptet in texten ”Hello, world!” i dokumentet (på den plats där skriptet låg).

Kommentarer

Läggs in mellan "<!--" och "-->"-tecken. Texten får innehålla "<" och ">" men inte "--"-tecken.

Dokumenttypsdefinitioner (DTD:er)

DTD:er är egentligen en separat företeelse och har egentligen inte så mycket med själva XML-filen att göra. Syftet med en DTD är att definiera regler för vilka märkord och attribut som får finnas i något XML-dokument. En DTD innehåller alltså inte något egentligt innehåll, sett ur användarens synvinkel.

XML 1.0-specifikationen definierar både syntaxen för egentligt innehåll i XML-filen och syntaxen för DTD:er. Det hade varit bättre om man hade skiljt ut DTD-

syntaxen och lagt det i en separat specifikation. I denna bok avhandlas DTD:erna i kapitel 5: "XML med regler för innehållet".

Bakgrunden till sammanblandningen är att det faktiskt är tillåtet att lägga DTD-syntax direkt inuti en XML-fil. DTD-konstruktioner känns igen på att de ligger mellan "<!" och ">"-tecken. Inte särskilt vanligt men det förekommer.

Sammanblandningen har också lett till att många kurser och böcker om XML i praktiken handlar onödigt mycket om att traggla DTD-syntax.

Begreppen "well formed" och "valid"

Detaljerna i hur ett XML-dokument beskrivs i W3C-rekommendationen XML 1.0. När ett dokument följer XML-specifikationen, säger man att dokumentet är "well formed". Skulle det dessutom uppfylla kraven i en DTD eller liknande, kallas det för "valid".

Att uppfylla villkoren för "well formed" är grundläggande. De flesta mjukvaror kräver "well formed" för att kunna behandla dokumentet korrekt. Egenskapen "valid" är i högsta grad förhandlingsbar.

XML Namespaces

XML Namespaces är enkelt och viktigt. Det borde faktiskt ha ingått i grundstandarden XML 1.0. Namespaces är ett relativt färskt påhitt, framdrivet av krav som globalitet och framtidssäkerhet.

Ett dokument består inte nödvändigtvis av element från en enda vokabulär. Ibland vill man i ett dokument ha information som använder två eller flera vokabulärer. Dessa vokabulärer är inte nödvändigtvis samordnade på något sätt. Risken är då att vokabulärerna råkar använda samma namn på något element eller attribut. XML Namespaces ("namnrymder") löser det problemet.

XML Namespaces bygger på att URI:er är globalt unika. Det som är intressant med URI:er, är bara att de är globalt unika. Vad som råkar ligga dit URI:n pekar, är alltså helt ointressant.

Man gör så att man associerar ett namnprefix med en URI. Associeringen görs i form av ett attribut till ett element. Syntaxen är "xmlns:prefix=URI". Attributnamnet består alltså av två delar. Första delen är specificerad till att vara "xmlns" (för XML Namespaces). XML har reserverat alla namn som börjar på "xml" för eget bruk, så xmlns kommer säkert inte att kollidera med något annat namn. Andra delen är det prefix som användaren vill kunna använda i sitt dokument. Man har helt enkelt bara definierat en förkortning på URI-strängen.

Om inget annat är sagt, ärver underelementen den namnrymd som föräldraelementet har, utan att man behöver ge prefixet. En annan av finesserna är att man kan associera flera namnprefix med olika URI:er. De olika namnprefixen används sedan för att särskilja vilket namnrymd ett element eller attribut tillhör.

```
<pryl xmlns:frtgA="http://www.foretagetA.se"
     xmlns:frtgB="http://www.foretagetB.se">
     <frtgA:artikelnummer>123</frtgA:artikelnummer>
```

```
        <frtgB:artikelnummer>DEF456</frtgB:artikelnummer>
    </pryl>
```

I exemplet ovan har två företag publicerat kataloger i form av XML-dokument. Bägge katalogerna använder begreppet artikelnummer. Antag att artikelnummer-begreppen betyder olika saker och att de används olika i de bägge katalogerna. Ett tredje företag kanske vill skapa dokument som använder bägge artikelnummerbegreppen. Genom att använda namnrymder kan man göra detta, utan att man riskerar att blanda ihop dem.

XML Inclusions (Xinclude)

XML Inclusions är en ny uppfinning som snabbt håller på att utvecklas. Grundsyftet är att göra det möjligt att bygga upp modulära dokument, ungefär som med entiteter, men med en funktionalitet som kompletterar entitets-mekanismen.

Inclusions kräver ingen DTD (som entiteter gör). Vill man inkludera ett annat XML-dokument är det bara att anropa det med en vanlig URI, man behöver alltså inte ge det ett entitetsnamn först.

Inclusions arbetar i termer av dokumentens logiska innehåll, det som beskrivs av dess Infoset (se nedan om Infoset). Det är underförstått att man bara är intresserad av att arbeta med import av XML-dokument.

XML Fragments Interchange

XML Fragments gör det möjligt att ta ut ett fragment av ett dokument och titta eller ändra på det utanför dokumentets sammanhang. För att korrekt kunna tolka ett fragment behövs det ofta kompletterande information, som man normalt hade fått av sammanhanget i dokumentet. XML Fragments specificerar hur den kompletterande informationen skall se ut.

XML Fragments säger ingenting om hur man återförenar fragmentet med dess ursprungsdokument. Detta behöver inte innebära att det inte är möjligt, bara att XML Fragments inte känner sig ansvarigt för att tillhandahålla en lösning.

Fragmentet behöver inte ha någon särskild struktur. Det behöver inte heller uppfylla villkoren på att vara "well-formed".

Ett exempel på när man kan använda fragment, är när man av sekretess-skäl inte vill avslöja hela innehållet i ett dokument. Ett annat exempel är när man vill titta på en liten del av ett väldigt stort dokument, utan att behöva ladda ner eller titta på hela dokumentet.

XML Information Set

XML Information Set ("Infoset") är inte något som en normal användare behöver bekymra sig om. Infoset-begreppet fyller mest en funktion för andra XML-relaterade standarder, inte någon direkt användbar funktion för en användare.

Infoset definierar en abstrakt beskrivning av innehållet i ett XML-dokument. Infoset skiljer alltså ut den informationsmängd i dokumentet, som i princip kan finnas oberoende av dokumentet och som dokumentet försöker vara en behållare för, från den informationsmängd i dokumentet som har uppkommit av interna implementations-mässiga anledningar inuti dokumentet.

Specifikationen för XML Infosets listar alltså exakt vad som är att betrakta som dokumentets "innehåll" och vad som bara finns där ändå, alltså vad som är signifikant information och vad som är icke signifikant information.

Det här förklaras kanske bäst med ett exempel. Antag att en tolk läser in ett XML-dokument och tillhandahåller detta, gentemot en applikation, som en hierarki av objekt. När vet man att objekten innehåller all information som fanns i dokumentet? Svaret blir att det gör det när det implementerar dokumentets infoset, så som det definieras enligt specifikationen för XML Infoset. Till exempel är ett attribut med dito värde givetvis signifikant information. Det ingår i dokumentets infoset. Attributens ordningsföljd inuti ett element är dock inte signifikant information. Det ingår inte i dokumentets infoset. Attributens ordningsföljd behöver därför inte vara bevarad i objekten.

Teckenuppsättningar

XML använder teckenuppsättningen i Unicode. Nationella tecken är alltså inbyggda från början. Kodningen av teckenuppsättningen skall vara UTF-8 eller UTF-16. Detta är mycket föredömligt, alla världens tecken finns med och de kodas på ett konsekvent och entydigt sätt. Den genomsnittlige användaren är inte alls lika föredömlig. I Sverige använder man troligen ISO 8859-1 (”8-bitars ASCII”), både som teckenuppsättning och som kodning. Översättningen är inte svår att göra men måste alltså hanteras på något sätt. Oftast sker det automatiskt inuti XML-verktyget, utan att man märker så mycket av det.

Sammanfattning

XML må vara en förenkling av SGML men XML är ändå inte enkelt på något sätt. XML är fortfarande ett komplext och kraftfullt verktyg som klarar av även mycket kvalificerade tillämpningar.

I smärre tillämpningar använder man ofta bara delar av alla finesserna. Namnrymder och processinstruktioner brukar det vara svårt att klara sig utan. Attribut går att undvara, men brukar vara bekvämast att ta med. Entiteter och DTD:er är mer diskutabla. Marginalnyttan varierar, samtidigt som de tenderar att vara mer krävande att arbeta med. Övriga delar är det lite för tidigt att uttala sig om, men XML Inclusions bör kunna utvecklas till att bli mycket flitigt använd. Användningen av XML Fragments beror väldigt mycket på applikationen.

Arvet från SGML är både en fördel och en nackdel. Största fördelen är givetvis kontinuiteten och att kunna bygga vidare på redan etablerade lösningar. Många gamla SGML-konstruktioner klarar dock inte riktigt av de nya krav som XML-världen har. De håller därför nu på att ersättas med helt nya lösningar, som ofta är både enklare och effektivare. Tills dess att de nya konstruktionerna har etablerats och "satt sig", är det rätt förvirrande och besvärligt att hitta fram till optimala arkitekturer.

5 XML med regler för innehållet

XML är i grunden självbeskrivande. Märkorden finns ju som en integrerad del av dokumenten. Dessa räcker långt för att beskriva innehållet i dokumentet. En applikation som läser ett dokument, kan mycket väl klara sig med bara detta.

Ibland vill man låsa ett dokument till att ha ett visst innehåll eller struktur. Man kanske också vill kunna testa att det följer de givna reglerna för innehåll och struktur. Man brukar prata om att man vill ha ett schema för hur innehållet i dokumentet ser ut, eller borde se ut. W3C erbjuder idag två notationer för att definiera dessa schema, DTD:er och XML Schema.

När ett dokument följer reglerna i ett schema, säger man att dokumentet är "valid". Det är underförstått att dokumentet då också är "well-formed", dvs. att det följer de grundläggande reglerna för XML-syntax.

Inget hindrar att ett dokument säger sig följa mer än ett schema samtidigt. Inget hindrar heller att olika schema använder olika notationer. Inget hindrar i princip heller att ett schema använder någon helt annan notation än DTD eller XML Schema. Andra alternativ, utan officiell uppbackning från W3C, är RELAX NG och Schematron liksom systemutvecklingens UML eller STEP-världens EXPRESS. Det behöver inte ens vara någon egentlig schema-notation som sådan, det kan i princip vara vilken ”anordning” som helst som uttrycker något om innehållet. Olika scheman kan ha olika syften och fungera på olika begreppsmässiga nivåer.

Dokumenttypsdefinitioner (DTD:er)

DTD-syntaxen ingår i grundstandarden för XML. DTD-konceptet har XML ärvt från SGML. DTD-syntaxen är lite udda och inte särskilt kraftfull. Den har sitt ursprung i klassisk dokumenthantering och är knappast lämpad för generellt bruk.

Exemplet nedan visar på hur man kan definiera en tillåten struktur på ett biofilmsbesök. Besöket börjar med en eller flera reklamsnuttar följt av själva filmen. Filmen i sin tur börjar med ett block där läget förklaras, därefter ett block där någon råkar illa ut (oftast hjälten eller hjältens kompis), därefter en sängkammarscen och till sist en biljakt.

```
<!ELEMENT Biofilmsbesök (ReklamSnutt*, Film) >
<!ELEMENT Film (Förklara, NågotRåkarIllaUt, Sängkammarscen, Biljakt) >
```

I exemplet ovan betyder *-tecknet att elementet kan förekomma godtyckligt antal gånger. En komma-separerad lista har innebörden att elementen i listan skall förekomma i den ordningen som de listas.

En TV-sändning från en Formel 1-tävling kan definieras till att ha en annan struktur. Den består av reklamsnuttar, biljakter och krascher i obestämd ordningsföljd och upprepad godtyckligt antal gånger:

```
<!ELEMENT TVsändningFormel1 (ReklamSnutt|Biljakt|NågonRåkarIllaUt)* >
<!ATTLIST TVsändningFormel1 StartTid CDATA #IMPLIED>
```

I exemplet ovan används |-tecknet till att indikera att man skall välja ett av flera möjliga element. Elementet definierades också till att ha attributet StartTid (av typen CDATA, character data, dvs. en sträng). Ett praktiskt förekommande dokument, enligt det senaste DTD-exemplet ovan, skulle därmed kunna se ut så här:

```
<TVsändningFormel1 StartTid="1400">
   <Biljakt> ... </Biljakt>
   <ReklamSnutt> ... </ReklamSnutt>
   <Biljakt> ... </Biljakt>
   <NågonråkarIllaUt> ... </NågonRåkarIllaUt>
   <ReklamSnutt> ... </ReklamSnutt>
</TVsändningFormel1>
```

En DTD påverkar normalt sett inte innehållet i ett dokument (som hävdar att det följer DTD:n). Det finns dock två sätt där innehållet ändå påverkas. Det enkla och självklara sättet är att en DTD definierar eventuella entiteter som anropas i dokumentet. Det lömska sättet är påverkan via definierade default-värden på attribut.

XML Schema

Bristerna i DTD-syntaxen är uppenbara. Flera förslag till ersättare har lanserats i olika omgångar, däribland DDML, DCD, SOX och XML Data. Det bästa i dessa förslag håller nu på att bakas in i det som kallas XML Schema. XML Schema betraktas som en mycket strategiskt viktig standard.

Den första stora förbättringen är att XML Schema använder XML som syntax. Man slipper alltså hålla reda på en separat syntax. Ett XML Schema-dokument är därmed ett vanligt XML-dokument. Schemat för dessa schema-dokument ingår givetvis i standarden (det schema-dokumentet har därmed den något udda egenskapen att följa sig självt).

Den andra stora fördelen är att XML Schema har mycket bättre möjligheterna att uttrycka datatyper. XML Schema fördefinierar en stor uppsättning datatyper av allmänt användbart slag.

```
<complexType name="TVsändningFormel1">
   <attribute name="StartTid" type="time" />
   <complexType name="SändningsDel" order="choice" />
      <element name="ReklamSnutt" />
      <element name="Biljakt" />
      <element name="NågonRåkarIllaUt" />
   </complexType>
</complexType>
```

I schema-exemplet ovan beskrivs i princip samma struktur som i andra DTD-exemplet ovan. Element av typen "complexType" används för att definiera nya elementtyper av typen complexType. Element av typen "element" används för att lista vilka (tidigare definierade) elementtyper som kan ingå i ett element. Syntaxen för XML Schema är som synes inte lika kompakt som den för DTD, men den är "renare", mer generell och med större uttrycksmöjligheter för datatyper.

Sammanfattning

Att låsa ett dokument till att ha en viss struktur kan vara bra. Det kan till exempel ge ett mått av säkerhet i något avseende. Låsningen innebär dock också just en låsning. Flexibiliteten minskar och önskade förändringar blir svårare att genomföra, i alla fall om dokumenten skall fortsätta att uppfylla kraven för "valid".

När man håller på med scheman, skall man också komma ihåg att det i princip är en bisak man håller på med. Det är alltid XML-dokumenten som gör själva jobbet med att innehålla och transportera informationen. Scheman är i slutändan alltid en extra kostnad. Ibland är kostnaden motiverad men långtifrån alltid.

När man testar ett dokument för att se om det är "valid", innebär det att man inte litar på den som genererat dokumentet. Om det är automatiskt genererat av en mjukvara, innebär det i praktiken att man håller på med att leta efter buggar i mjukvaran. Om dokumentet är manuellt genererat, är det vettigare att testa för "valid". Genom att kräva egenskapen "valid", kan man tvinga författaren att till exempel följa en definierad mall för hur ett dokument skall se ut.

En vanlig användning för scheman är när man vill definiera en uppsättning märkord inom något visst område. Scheman tillhandahåller ju en väl definierad och allmänt förstådd syntax för att beskriva vad som får finnas. Om man bara vill beskriva vilka märkord som ingår, utan att vilja definiera formella regler utöver detta, kan det ibland räcka med vanlig prosa eller med ett exempel-dokument.

De scheman som används inom XML sysslar med syntax, även om det är syntax på en lite högre nivå än själva grundstandarden för XML. De säger i stort sett ingenting om märkordens semantik. Det är luddigheter i märkordens semantik, glidningar i deras betydelse, som brukar vara det stora problemet när man försöker standardisera informationsblock.

Det finns tyvärr inte någon grafisk notation för att beskriva innehållet och strukturen för något XML-dokument. Detta skulle göra det lättare för icke-experter att förstå innehållet. Klassdiagrammen i UML (Unified Modeling Language) är givetvis ett mycket näraliggande och frestande alternativ, men i princip är fältet helt öppet.

6 XML med stil

Ett XML-dokument innehåller i princip bara just innehåll. Det innehåller inget om hur innehållet skall presenteras, till exempel om hur en layout skall se ut. Informationen om detta ligger normalt separat, oftast i form av en fil. Denna fil brukar kallas för en "style sheet" eller "stilmall".

Ett dokument kan alltså presenteras på olika sätt, beroende på vilken stilmall man använder. Man kan också använda flera stilmallar samtidigt, men om stilmallarna då överlappar, måste man bestämma vilken stilmall som har sista ordet. Inget hindrar heller att olika stilmallar har olika syntax.

Stilmallsbegreppet är mycket vettigt. Det är nästan alltid en mycket god idé att separera layout och innehåll. Att smidigt kunna ändra layouten på många dokument samtidigt, eller att kunna presentera samma innehåll på olika sätt i olika sammanhang, är en stor och fundamental fördel.

Det där med separationen av innehållet och av presentatationen, behöver man inte alltid se som en så fruktansvärt grundläggande princip. Det är inte alltid som det går att särskilja vad som är innehåll och vad som är innehållets utseende. När man till exempel använder XML för att representera vektorgrafik (se vidare om SVG i kapitel 9), är ju innehållet ett utseende. Man kan ha en rätt avspänd inställning till uppdelningen. Man kan se det som att en XML-fil kan anropa en annan fil och att en applikation kan tolka den anropade filen som en uppsättning formateringsinstruktioner. Ibland är detta ett bra sätt att jobba, ibland inte. Har man en formateringsdel, och om man vill kunna ändra formateringen på ett smidigt sätt, är uppdelningen självklart mycket vettig. I andra fall kanske man väljer att hårdkoda eventuell formatering. Man kanske också kan tänka sig att använda den anropade filen till något annat än till formatering. Som tidigare nämnts, HTML är ett alldeles lysande exempel på en extremt tillbakalutad inställning till den här uppdelningen.

När man går in i stil-branschen skall man göra det på ett mycket medvetet sätt. Problemet är att stil och layout snabbt kan bli hur komplext som helst. Det finns en linjär skala av stigande komplexitet. Man kan lägga sig i princip var som helst på den skalan. Det finns inga naturliga gränser för komplexiteten. Det finns hela tiden goda skäl för att det ska bli ytterligare ett snäpp "bättre" och därmed mer komplext. Det här kan lätt bli rena träsket. Enda riktigt verksamma gränsen tycks vara akut utmattning hos de inblandade. Långt innan man har nått det stadiet, har man nog glömt allt vad kostnadseffektivitet heter.

Det har förts en debatt om CSS kontra XSL (se nedan). Det är ju två standarder som gör i princip samma sak och som överlappar varandra mycket. Stundtals har debatten varit intensiv. Vad man har för åsikter om dessa standarder verkar mer avspegla åskådarens relativa position på komplexitetsskalan än några absoluta sanningar om standarderna. Kommer man från databas- eller web-världen räcker befintlig CSS fint och XSL är onödigt komplicerat. Kommer man från publishing är XSL en kompromiss, helst skulle man vilja ha DSSSL (om nu bara någon orkade implementera det).

HTML beter sig något irreguljärt i sammanhanget. HTML innehåller mycket stil-information direkt i märkorden. De bläddrare man normalt använder för att beskåda HTML-dokument, innehåller en hel del inställningar där man kan styra hur olika märkord skall tolkas. HTML kan också använda stilmallar, till exempel CSS-mallar. Om HTML hade använt stilmallar konsekvent från början hade HTML varit mycket rationellare att jobba med.

CSS

CSS (Cascading Style Sheets) fungerar med både HTML och XML. CSS använder en alldeles egen syntax. Varje CSS-kommando består i princip av en urvalsdefinition följt av en klammerparentes innehållande olika definitioner på vilka stilattribut som skall appliceras på urvalet.

```
urval {attributnamn:attributvärde}
```

De stilattribut som det vanligen handlar om är till exempel fonter, färger och marginaler.

```
NAMN               {font-family:"Helvetica"; color:black}
NAMN.FÖRNAMN       {font-size:10pt}
NAMN.EFTERNAMN     {font-size:12pt}
```

En av grundtankarna i CSS är att ett dokument skall kunna ha flera stilmallar i kaskad (därav namnet). Man applicerar alltså flera stilmallar efter varandra. Om något element får något stilattribut definierat mer än en gång, har man en uppsättning regler för att räkna ut vilket attribut som vinner. Reglerna är att sist definierad vinner, att definierad på lägst hierarkisk nivå vinner och att attribut definierade som "important" vinner. Reglerna var här uppräknade i styrkeordning med den sista som den starkaste.

Själva anropandet av en stilmall görs inifrån ett XML-dokument genom att använda en fördefinierad så kallad processinstruktion (se kapitel 4):

```
<?xml-stylesheet href="stilmall.css" type="text/css"?>
```

XML stödjer inte stilmallskod inuti själva dokumentet på det sätt som HTML tillåter (inuti <STYLE>-element). Avsaknaden är inte någon förlust, att blanda innehåll och stil på det viset är mest klassiskt "risig" HTML-stil. XML har heller inte HTML-tricket med det fördefinierade CLASS-attributet hos element. I XML kan man använda det om man vill, men man är inte hänvisad till att använda just den konstruktionen.

CSS finns ute både i version 1 och i version 2 med version 3 på "g". CSS1 stödjer textfont, textdekorering, textfärg, box-layouter, bakgrundsfärg, bakgrundsbild och liststilar. Stora haken med CSS1 är att det inte stödjer tabeller. CSS2 tillför det efterlängtade stödet för tabeller. Vidare har man lagt till ett rätt omfattande stöd för olika typer av media. CSS3 handlar mest om att dela upp den numera rätt omfattande CSS-specifikation i mera hanterbara moduler. Man börjar ana att folket bakom CSS inte längre är lika intresserade av att lägga till ny funktionalitet och den därmed vidhängande ökande komplexiteten. Kanske börjar man så smått närma sig en viss grad av utmattning.

CSS tillhandahåller i grunden rätt enkla stilbegrepp. Man kan inte definiera någon särskilt kvalificerad logik i hur layouten skall göras. Vill man göra det, kan man alltid via DOM bygga upp en egen logik som ordnar om i dokumentet, innan man skickar det för visning enligt någon stilmall.

XSL

XSL (eXtensible Stylesheet Language) har samma funktion som CSS, dvs. som ett språk att skriva stilmallar i. En XSL-stilmall anropas av ett XML-dokument på samma sätt som en CSS-stilmall, alltså med hjälp av en fördefinierad processinstruktion (se ovan och kapitel 4).

Syftet med XSL är att dels tillhandahålla mera sofistikerade layouter, dels att fungera bra även för vanliga pappersdokument. XSL har bland annat ett sidbegrepp där man kan hålla reda på vilken sida man är på. Detta saknas i (nuvarande) CSS och möjliggör klassiska pappersdokument-konstruktioner som till exempel innehållsförteckning och index. Arbetsfördelningen mellan CSS och XSL är alltså i princip att CSS är inriktad mot web-sidor och att XSL är mer ämnat för traditionella publikationer, på webben eller på papper.

XSL kan mycket väl användas till HTML, i alla fall i princip. I praktiken fungerar det inte alls, praktiskt använd HTML är inte god XML. Stödet för XSL i de vanligaste bläddrarna är ännu obefintligt av naturliga skäl.

XSL använder XML som syntax. Man slipper alltså en speciell syntax utan kan använda alla XML-verktyg rakt av. XSL har inget eget programmeringsspråk, på det sätt som till exempel DSSSL har. Istället används något som kallas XSLT (se nedan).

XSL har nackdelen att komma efter CSS och måste i praktiken visa sig vara bättre än CSS för att slå igenom ordentligt. XSL är i dagsläget en betydligt mer osäker investering än CSS men ser ändå ut att kunna nå kritisk massa. XSL är en mera "genomarbetad" och "sofistikerad" standard. Marginalnyttan i XSL i förhållande till CSS är troligen tillräckligt stor. Primära risken med XSL är att det visade sig bli för komplext. På sin kärnmarknad har XSL också konkurrens från det väl spridda och väl fungerande PDF-formatet.

XSLT

När man gör en layout av ett dokument, görs layouten ofta i två steg. Första steget är att göra en omstrukturering (dvs. transformeringen) av dokumentet på något sätt, andra steget är att hänga på olika stil-attribut som typsnitt och färg etc.

XSLT (XSL Transformations) sysslar med transformationer av dokument. XSLT är tänkt att användas ihop med CSS eller XSL där XSLT sköter transformeringarna, CSS eller XSL sköter resten.

XSLT är mycket användbart för transformationer i allmänhet, till exempel mellan två olika scheman. En särskilt populär användning är för att generera HTML från XML.

XSLT använder XML som sin syntax. En uppsättning av XSLT-definitioner är alltså samtidigt ett XML-dokument.

Ett XSLT-dokument kallas också för en stilmall. Det anropas normalt från ett XML-dokument på samma sätt som andra stilmallar. Liksom för andra stilmallar, är det inget som hindrar att man anropar flera mallar i sekvens.

XSLT används bara för att definiera vilka transformationer som skall göras. Det säger inget om hur transformationerna rent implementeringsmässigt skall genomförs. Det förutsätts att någon XSLT-tolk finns tillhanda för att se till att jobbet blir gjort på något sätt.

XSLT sysslar med dokumentets logiska struktur. Hur dokumentet ser ut fysiskt, till exempel serialiserat i form av en fil, bekymrar sig inte XSLT om. XSLT förutsätter att dokumentet finns tillgängligt i logisk form, alltså som en trädstruktur. Om

dokumentet ursprungligen fanns i form av en fil, måste en tolk alltså ha läst in det först och genererat trädstrukturen. I princip är det alltså inget som hindrar att XSLT används för att definiera transformationer på andra trädstrukturer, alltså sådana som inte nödvändigtvis har sitt ursprung som ett XML-dokument.

Ett alternativ till XSLT är att skriva transformeringarna i form av ett vanligt program, till exempel med DOM som API mot dokumentet. I regel bör dock XSLT vara enklare och mer produktivt att använda, då det arbetar på en högre abstraktionsnivå än DOM.

7 XML med länkar

XLink

XLink definierar länkar mellan XML-dokument. Länkarna är avsedda att användas till i princip vad som helst. Ett exempel på användning är den vanliga hyperlänken, välkänd från HTML, fast mer intelligent och smidigare att jobba med.

I SGML-världen användes HyTime för länkar. XLink är en förenkling av HyTime. De enkla länkmodellerna är gemensamma för XLink och HyTime men de mer komplexa länkarna följer den s.k. TEI-modellen, inte HyTime. XLink använder inte heller begreppet "architectural forms" (en sorts halvfärdig DTD) som HyTime införde.

XLink jobbar mot dokumentens logiska strukturer, hur de fysiskt är realiserade är irrelevant.

Ett grundläggande krav för länkarna i XML är att de kan existera ihop med länkmekanismen i HTML. HTML har en ytterst enkel länkmodell. En länk startar där den ligger. Den ligger i ett ankarelement (eng. "anchor") och målet definieras med attributet HREF. Den pekar antingen på något namngivet ankare definierat i samma HTML-fil som den ligger i, eller på ett annat dokument via dokumentets URL. HTML-länken kan inte peka inuti någon annat dokument, inte heller på flera ställen samtidigt. Det är underförstått i HTML-länken att den alltid traverseras från startpunkten till slutpunkten.

```
<A HREF="target.html">Klicha här!</A>
```

XLink definierar två typer av länkar, SIMPLE och EXTENDED. SIMPLE-länkar fungerar som HTML HREF-länkar, länken ligger där den startat och den pekar på bara ett slutmål. EXTENDED-länkar behöver inte ligga på något särskilt ställe, var länken ligger är alltså frikopplat från var den börjar och var den slutar. Detta innebär bland annat att man kan skapa länkar mellan dokument, som man själv inte har skrivrättigheter till. En EXTENDED-länk kan också har mer än en startpunkt eller slutpunkt. Den har heller ingen fördefinierad riktning utan riktningen kan gå åt bägge hållen i en länk.

En SIMPLE-link består av några fördefinierade attribut och ett innehåll. Länkfunktionen styrs av attributen. Innehållet är i princip oväsentligt för länkfunktionen.

```
<xlink:simple href="länkmål.xml"
              role="test"
              title="test av länk"
              show="new"
              actuate="user">
   Klicka här för att testa länken!
</xlink:simple>
```

EXTENDED-länkar består i regel av flera länkbågar (eng. "arcs"). Att lägga all länkinformationen som attribut, skulle bli ohanterligt för komplexa länkar. Länkbågarna ligger därför som element under länkelementet. Länkelementet har fortfarande några fördefinierade attribut men de ser nu lite annorlunda ut.

```
<xlink:extended role="test"
                title="test av länk"
                showdefault="new"
                actuatedefault="user">
   <xlink:from> ... </xlink:from>
   <xlink:to> ... </xlink:to>
   Klicka här för att testa länken!
</xlink:extended>
```

Själva pekardelen av en länk, dvs. utpekande av en länkbörjan eller ett länkslut, görs i två steg. Första steget är en URI som pekar ut ett dokument. Andra steget pekar ut något inuti det dokument. För XML-dokument används XPointer. För dokument som inte är XML-dokument, används någon annan syntax. Det är alltså fullt möjligt att peka inuti dokument som inte är XML-dokument. Det är upp till användaren och applikation att komma överens om detaljerna i sådana fall. Detaljerna behöver inte vara komplicerade, det kan räcka med en ID-mekanism.

XPointer

XPointer är en hjälpstandard till XLink. XPointer sysslar med utpekandet av exakt var en länk börjar eller slutar. XPointer pekar alltså ut en bit av ett XML-dokument. XPointer tar vid där URI:er slutar. En fullständig adress i XLink är på formen URI + XPointer.

En XPointer-pekare kan bestå av ett godtyckligt antal pekningssteg. En pekare kan välja ut något baserade på olika kriterier, till exempel ett visst typ av element, ett visst typ av attribut eller ett visst värde på ett attribut.

Notera att dokumentets logiska struktur kanske under layout-processen transformeras om av XSLT. Dokumentet som visas för en användare kanske har en annorlunda struktur än det som XPointer pekade in i. Man får alltså vara försiktig med vad som händer när man använder XPointer och XSLT samtidigt.

XPath

XPath är en hjälpstandard till både XPointer och XSLT.

8 Rå XML

Det här kapitlet handlar om hur man "mekar" med XML-dokument. En normal användare kan hoppa över det. En skicklig programmerare skall kunna det i sömnen. Fokus är på de standarder som finns och på vilka egenskaper dessa har, inte på verktygen. Vilka verktyg som finns är en viktig praktisk kunskap, men behandlas lite styvmoderligt i det här sammanhanget. Den kunskapen är en utpräglad färskvara. Det är trots allt en introduktion till XML du sitter och läser.

SAX

SAX (Simple API for XML parsers) är inte en W3C-standard. SAX hittades på av några av de tidiga implementatörerna av XML-tolkar, men har visat sig fungera bra och används flitigt.

SAX är händelsestyrd, till skillnad från DOM:s trädorienterade vy. I SAX skickar tolken information till applikationen om XML-filen, allt efter att den läser in XML-filen. Abstraktionsnivån är alltså inte så hög men det är lättare att bygga tolken. SAX har en nackdel i det att man inte får något inget stöd för att bygga upp strukturen hos dokumentet, det arbetet läggs på applikationen istället.

DOM

DOM (Document Object Model) är en specifikation för ett API till ett XML-dokument. Genom att standardisera API:et, blir applikationerna som använder API:et flyttbara. Ett dokumentet kan givetvis inte själv tillhandahålla något API, istället tillhandahålls API:et av en mjukvara som har läst dokumentet.

DOM beskriver innehållet i ett dokument som en hierarki av objekt. DOM talar om hur man går från en elementhierarki till en objekthierarki. DOM talar också om hur objektens gränssnitt skall se ut.

DOM sysslar enbart med den logiska strukturen på dokumentet. Hur dokumentet implementationsmässigt plockades ihop av olika delar, typ entiteter etc., det bryr sig DOM inte om.

Man kan jämföra DOM med SQL i det att det är ett standardiserat sätt att manipulera data. I fallet SQL ligger data i tabeller. I fallet DOM ligger data som XML-dokument.

Detta tenderar till att innebära att hela dokumentet måste vara inläst i minnet på applikationen innan man kan börja arbeta med dokumentet. Om dokumentets trädstruktur är mycket komplex och "djup", kan minnesåtgången bli besvärande stor. Likaså, om man bara är intresserad av en liten del av ett dokument, kan det kännas onödigt att läsa in hela dokumentet och bygga upp trädstrukturen. DOM har fördelen av

att vara bekväm och "ren" ur programmerarens synvinkel men till priset av vissa implementationsmässiga svagheter. SAX fungerar på en lägre nivå men kan ofta vara effektivare. Vilket man väljer beror på applikationen.

XML i databaser

XML är ett sätt att strukturera information. I fallet XML väljer man att strukturera informationen i termer av en hierarki av element. I grunden handlar det om strukturering på ett konceptuellt eller logiskt plan. Hur informationen sedan lagras fysiskt, är ett separat problem.

SGML har länge kunnats lagras i databaser av olika slag (entity managers etc.), likaså givetvis XML. Det traditionella publishing-orienterade SGML/XML-folket är följaktligen bara måttligt upphetsade av nya sätt att lagra XML i databaser. Det intressanta är att databas-folket verkar ha tänt på alla cylindrarna på XML. Tydligen tillför XML något nytt och viktigt till deras värld.

Databas-branschen har ju sedan länge handlat om matchen mellan relationsdatabaser och objektdatabaser, där objektdatabaserna traditionellt haft rollen som framtidens teknik och där man i brist på kommersiellt genomslag, gradvis har resignerat till att få behålla den rollen. Hybriden "object-relational"-databaser har lanserats av de stora relationsdatabas-leverantörerna och får väl ses som ett sätt att tillhandahålla objektens semantik om och inte dess prestanda. Tillräckliga prestanda får man helt enkelt via Moore's lag.

Hela den diskussionen verkar nu, i alla fall delvis, vara överspelad. XML representerar en abstraktionsnivå som ligger närmare användarens, vilket ger en högre produktivitet i utvecklingsfasen. XML är också bra på "semi-structured information", något som man traditionellt haft svårt för. Mycket återstår, framförallt vad gäller Query Language (typ XQuery), men så här långt ser det mycket lovande ut.

Det skadar inte att vara lite misstänksam i det här läget. Databas-leverantörerna har alltid förespråkat "fat servers", med så mycket intelligens som möjligt i eller nära databasen. Man bygger in applikationerna och kan ta ett kliv uppåt värde-kedjan (eng. "value chain") i organisationerna, samtidigt som man låser fast kunderna. Det ligger alltså i till exempel Oracles affärsmässiga intresse att dra kunderna mot att lägga så mycket semantik som möjligt i databasen, till exempel genom att lagra XML-dokument istället för tabeller. Samtidigt innebär ändå konceptet klara fördelar. Man skall inte utesluta "win-win".

XML med koppling till databaser

Man måste inte lagra hela XML-dokumentet i en databas. En klassisk lösning är att låta XML-dokumentet ligga som en vanlig fil och att sedan koppla innehållet i dokumentet till innehållet i någon existerande databas av traditionell typ.

Kopplingen görs i form av särskilda element inuti dokumentet, som på något sätt beskriver vilka data som skall hämtas ur vilken databas. När dokumentet efterfrågas är det någon mjukvara (i regel serverbaserad) som tyder dessa särskilda element och som ersätter dem med data ur någon databas, vartefter dokumentet levereras. I dagsläget är dessa lösningarna helt leverantörsspecifika.

9 Applikationer

Med applikation menas i XML-sammanhang i princip en vokabulär, en uppsättning ”taggar”. Det finns många hundra allmänt kända applikationer av XML. Till detta kommer de som mest är avsedda för internt bruk och som inte marknadsförs utåt. De applikationer som det berättas om här, är bara ett litet urval av vad som finns. Urvalet är inriktat mot de applikationer som är viktiga för XML-användningen, mer än mot vanliga och typiska applikationer riktade mer mot slutanvändare.

XHTML

Det här är lite av en straffspark. HTML bygger formellt på SGML. SGML är på väg att ersättas av XML. Varför inte skriva om HTML baserat på XML? Jomenvisst är det en alldeles utmärkt idé. XHTML 1.0 är därför HTML 4.0 uttryckt i XML. Då XML är betydligt mera ”tight” definierat än SGML, innebär detta en kraftig hyfsning av vad som är tillåtet i HTML.

XHTML 1.1 går vidare med hyfsningen av HTML genom att modularisera det. Ett syfte bakom modulariseringen är att kunna definiera små subset av HTML. Det finns moduler för struktur, text, hypertext, list, applet, frames, skript och formulär. Detta innebär ett bra alternativ till WML, för applikationer med liknande behov.

I praktiken är XHTML ändå rätt kört. Man kommer aldrig ifrån att det finns enorma mängder med ohyfsad HTML ute på nätet, som man knappast kan ignorera och som knappast kommer att försvinna. XHTML är helt enkelt för sent ute.

SVG

Webben har i dagsläget text (HTML) och rastergrafik (GIF, JPEG och PNG) men saknar vektorgrafik. Det närmaste man kommer vektorgrafik är PDF-formatet, en variant av Postscript. PDF-formatet är dock tekniskt helt olämpligt för användning tillsammans med XML. Något helt nytt behövs.

SVG:s bakomliggande begreppsapparat för vektorgrafik är annars inte helt olik Postscript och PDF. SVG handlar om att lägga ut grafik på ett koordinatsystem, ett koordinatsystem som är tänkt att motsvara en bildskärm eller ett papper. Man sysslar inte med 3D, man har ingen koppling till 3D och man har bara ett enda koordinatsystem.

SVG-grafik består av tre typer av komponenter. Dessa är kurvor ("paths"), text och rastergrafik. Ett rikt sortiment av olika typer av grafiska effekter finns, till exempel färg, filter, animeringar och klickbara objekt.

SOAP

SOAP (Simple Object Access Protocol) är ett riktigt vettigt upplägg. Grundsyftet är att definiera hur applikationer skall göra för att utbyta XML-meddelanden med varandra. Det handlar alltså om att bygga distribuerade applikationer, där man annars kanske skulle använda sådant som till exempel RPC:er, Corba, DCOM eller RMI.

SOAP är en enkel specifikation. Den består bara av några märkord och av beskrivningar på hur dessa skall tolkas. Själva XML-meddelandet läggs inuti SOAP-strukturen. En applikation skickar ett XML-meddelande till en annan applikation och får ett XML-meddelande tillbaka. Man använder redan befintliga TCP/IP-kommunikation, http-protokoll och http-baserade web-servrar. I stort sett hela infra-strukturen finns alltså redan på plats. En viktig finess med att använda http är att man smidigt kan komma genom brandväggar.

Den typ av distribuerade applikationer man bygger upp med SOAP, blir alltså relativt löst kopplade applikationer, där skalbarhet prioriteras över prestanda. För den stora massan av applikationer, ett helt korrekt val.

UDDI

UDDI (Universal Description, Discovery and Integration) är, enkelt uttryckt, en telefonkatalog för nät-tjänster (”web services”).

Det problem man är ute efter att lösa är alltså ”hur hittar jag min affärspartner?” och ”vilka nät-tjänster kan jag använda?”.

Man börjar med att skapa ett antal databaser för UDDI. Bland annat skaparna av UDDI tillhandahåller några sådana on-line men vem som helst är välkommen att sätta upp sin egen.

I denna databasen ligger det sedan information om vilka företag som finns och vilka nät-tjänster de tillhandahåller. Informationen är på ett UDDI-specificerat format. Det sätt som informationen läggs in och sedan görs sökningar på, är också definierat av UDDI. UDDI använder SOAP (se ovan).

WSDL

WSDL (Web Services Description Language) handlar om att beskriva hur en nät-tjänst fungerar på ett tekniskt plan. Ett WSDL-dokument definierar hur in-data till tjänsten skall se, ut, hur ut-data skall se ut och vilka operationer som kopplar ihop in-data med ut-data. WSDL används ovanpå någon mekanism för att skicka XML-meddelanden till/från tjänsten, ofta SOAP (se ovan) men man kan även använda http direkt eller till och med SMTP (dvs. e-post).

Ordförklaringar

dokument	En klump av information. Ej nödvändigtvis innehavare av de egenskaper som traditionellt associeras med ordet "dokument", såsom att det skulle vara möjligt att skriva ut på papper eller ens att det skulle vara läsbart för en människa.
element	De byggstenar som dokument byggs upp av. I andra sammanhang pratar man om "entiteter" eller "objekt" för ungefär samma sak.
entitet	Mycket specifik betydelse inom XML, se kapitel 4 "XML med fartränder". Vanligast betydelsen inom XML är att det är fördefinierad textsträng. Utanför XML brukar det oftast betyda något i stil med en informationsklump som kan bestå av flera olika mindre klumpar.
schema	En beskrivning om en informationsmängd, till exempel dess syntax eller dess semantik.
semantik	Vad informationen har för "mening", vad den "betyder".
syntax	En samling detaljerade regler för hur en informationsmängd tekniskt skall vara uppbyggd och strukturerad.
URI	Uniform Resource Identifiers. Generell benämning för en identifierare för en resurs på Internet. Finns i två varianter, URL och URN, där URL är i särklass vanligast.
URL	Uniform Resource Locators. Specialfall av URI när man använder resursens (globalt unika) åtkomstadress för att identifiera den.
URN	Uniform Resource Names. Specialfall av URI där man använder ett globalt unikt namn för att identifiera en resurs.
vokabulär	En uppsättning märkord ("taggar") som man kan använda när man definierar innehållet i ett XML-dokument. Bestämmer vad dokumentet kan innehålla för information.