



rockad



The Stockholm Chamber
of e-commerce



INTERACTIVE
INSTITUTE

infobyte

1 | februari 2001

XML Schema – koloss på lerböter? [Stig Berild]

En artikel från Rockad

Kontakta redaktionen

Nicklas Lundblad,
Stockholms
handelskammare
nicklas@acm.org

Alex Jonsson,
Interactive Institute
alex.jonsson@interactiveinstitute.se

Per Melander,
Interactive Institute
per.melander@interactiveinstitute.se

PDF- & Webbproduktion

Pär Abrahamsson,
Svenska Infobyte
par@Infobyte.se

XML Schema – koloss på lerfötter?

Av: Stig Berild

Framkom, Sveriges Tekniska Attachéer

XML Schema är en sedan länge emotsedd W3C-standard vars syfte är att åstadkomma ett betydligt kraftfullare språk för dokumenttypsspecifikation än vad XML 1.0 erbjuder med sitt DTD-språk (DTD=Document Type Definition). XML Schema är (november 2000) en Candidate Recommendation inom World Wide Web Consortium (W3C). De flesta bedömer att det blir en slutligen antagen Recommendation inom den närmaste tiden.

Syftet med denna rapport är att ge en introduktion till de viktigaste begreppen/egenskaperna i XML Schema. Avsnitt 1 beskriver kortfattat upprinnelsen. Avsnitt 2 diskuterar det övergripande syftet samt ställer det i relation till den lösning XML 1.0 erbjuder. Den som är hemmastadd i XML och DTDer kan nöja sig med att skumma detta avsnitt. Avsnitt 3 beskriver några av de mer centrala faciliteterna för dokumentstrukturbeskrivning medan avsnitt 4 exemplifierar några av de tillgängliga möjligheterna att definiera datatyper. I avsnitt 5 slutligen vågar vi oss på några kritiska synpunkter.

Inledning

Alla tycks vara överens om att XMLs DTD-språk enligt XML 1.0 saknar en del väsentlig uttryckskraft som visat sig vara angelägen för korrekt hantering vid många typer av dokumentsamverkan. Dit hör rikare möjligheter att ange strukturvillkor. Idag kan plustecken (1:M), asterisk (0:M) och frågetecken (0:1) användas för att ange antalsvillkor. Förfinade villkor som exempelvis 0:3 eller <1000 går inte att uttrycka. Inte heller går det att precisera vilken datatyp som ska gälla för ett dataelement (annat än #PCDATA). Det kan exempelvis vara viktigt att kunna deklarerat att *ordernr* är ett fyrsiffrigt heltal mellan 2700 och 5800 eller att *summa* inte får vara större än 100000 kronor samt alltid ska anges inklusive ören. Förutom standarddatatyper bör egna datatyper kunna specificeras för unika behov, t.ex. olika typer av tidsintervall och symbolsekvenser. Att DTDer uttrycks med en egen syntax istället för i XML är en annan besvärande barlast alla vill se en ändring på.

Väl medveten om bristerna initierade W3C 1998 en ny aktivitet som skulle utmytna i en rikare uppsättning begrepp och uttryck samt ett standardiserat

sätt att uttrycka dessa i XML. Den kom att gå under benämningen **XML Schema**. Fyra förslag lämnades in, vart och ett representerandes tunga aktörer inom XML:

- XML-Data (Microsoft, DataChannel)
- Data Definition Markup Language – DDML (GMD)
- Document Content Description for XML – DCD (IBM, Microsoft)
- Schema for Object-Oriented XML – SOX (Commerce One)

Varje förslag utnyttjar XML-syntax för sin definition men med delvis olika egenskaper för övrigt. Förslagen var inte hämtade ur luften utan från existerande produkter eller genomarbetade begreppsmodeller. Därmed uppstod genast problem. Respektive aktör hade och har att bevaka sina egna intressen. Processen hamnade också ett tag i dödläge innan man insåg att alla i långa loppet hade att vinna på en gemensam standard. XMLs trovärdighet stod på spel. Där ser man riskerna med att ha en process där några få partsintressen får styra med utslagsgivande makt. Dessutom representerar förslagsställarna leverantörskategorin snarare än användarkategorin. Inte helt lyckligt eftersom det är den senare kategorin som primärt känner behoven och som så småningom till vardags kommer att behöva jobba med det slutligen antagna schemaspråket.

Ett, som arbetsgruppen bedömer vara ett i princip slutligt förslag gavs den 24 oktober 2000 klassificeringen Candidate Recommendation. Framkommer inte några väsentliga invändningar under en därefter följande remissperiod återstår bara för W3C att ta det sista steget - att utnämna förslaget till en Recommendation.

XML Schema kommer, om och när det blir en antagen standard, att successivt kunna ersätta den nu tillgängliga DTD-syntaxen i XML 1.0.

Specifikationen för XML Schema är uppdelad i två separata dokument, *XML Schema Part 1: Structures* och *XML Schema Part 2: Datatypes*. Därtill kommer en introducerande skrift *XML Schema Part 0: Primer*. Observera att den senare inte är del av specifikationen men en väl så angelägen komplettering för att öka förståelsen av de två första, mycket "tunga" och kompakta dokumenten.

Förutsättningar

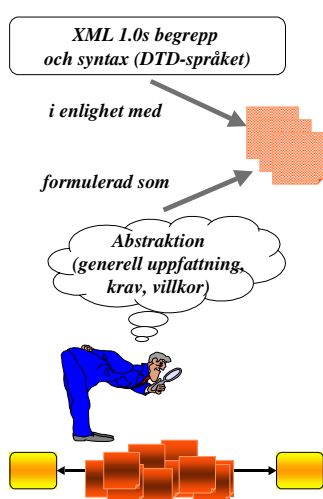
XML Schema ändrar inte på något sätt på den XML-syntax som används för att uttrycka dokument. Schemat är endast till för att uttrycka villkor, krav på vad som är tillåten struktur och acceptabelt innehåll för en viss typ av dokument. DTD-språket enligt XML 1.0 har samma syfte. Skillnaden ligger i att XML Schema har större uttrycksstyrka, kan precisera villkoren med exakt. DTD och XML Schema ser på XML-dokument med olika ögon. Av den anledningen har de också valt att använda delvis olika begrepp för att uttrycka de generella egenskaper som ska gälla för varje XML-dokument av viss typ.

Antag att ett stort antal dokument flödar mellan två (eller flera) parter. Se figur 1.

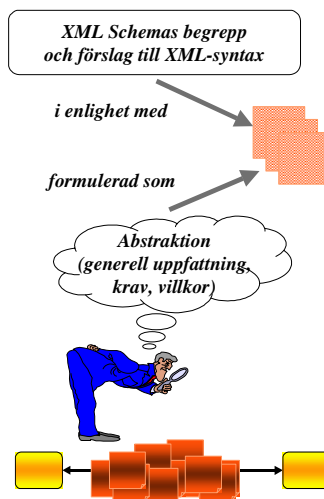


Figur 1

Parterna bestämmer sig så småningom för att styra upp innehållet enligt tre olika dokumenttyper, var och en med sin struktur, sina regler och villkor. Om de använder sig av DTD-syntaxen gäller principen enligt figur 2a (läses nerifrån). Är de trendigt moderna eller behöver uttrycka mer avancerade villkor arbetar de med hjälp av XML Schema-syntaxen (figur 2b).



Figur 2a

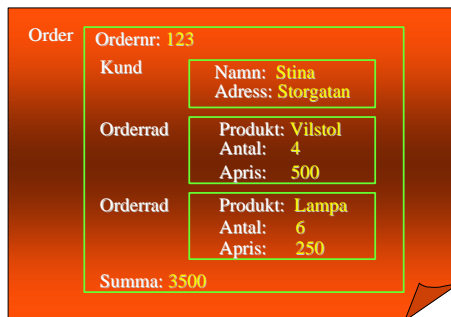


Figur 2b

Antag istället att vi börjar i andra änden – en förmodligen betydligt vanligare förutsättning än ovan. Parterna träffas och kommer överens om att utbyta information i anslutning till en gemensam affärsprocess. De förhandlar fram en exakt definition av de tre dokumenttyper som behövs för att stödja affärsprocessen. Därefter sätter utbytet igång. Samma bild men en annan tågordning.

Abstraktionen ger oss i alla händelser ett antal generella begrepp som svarar mot vår allmänna uppfattning om de tänkta eller existerande dokumentens innehåll. Vilka begrepp som definieras beror på vilka typer av data som ska utbytas, mellan vilka och för vilket syfte. Antag att vi har en mängd orderdokument som vart och ett i princip ser ut som i figur 3a (givetvis med för varje dokument sina unika värden, antal orderrader, mm). Samma order men med en layout som tydligare visar dess uppbyggnad eller struktur visas i figur 3b.

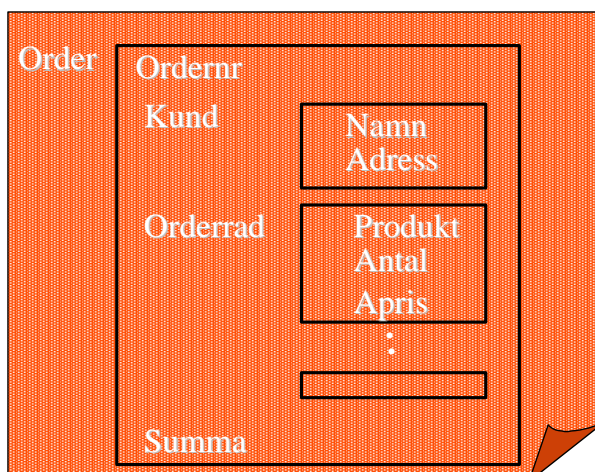
<u>Order</u>		
		Ordernr: 123
Kund:	Stina Storgatan	
Produkt	Antal	Apris
Vilstol	4	500
Lampa	6	250
Summa att betala: 3500		



Figur 3a

Figur 3b

Om vi utgår ifrån att varje orderdokument i princip har samma uppbyggnad blir den generella strukturen för varje orderdokument följdriktigt enligt figur 4. Där visas också de begrepp som de samverkande parterna valt använda för de olika elementen. Figur 4 är en abstraktion av alla orderdokument.



Figur 4

Begreppen behövs för att förklara de olika elementens innebörd när ett orderdokument skickas i form av en löpande sekvens tecken till mottagaren. I XML-syntaxen redovisas de som märkord. Med dessa märkord har mottagaren möjlighet att entydigt avtolka den inkommande teckensekvensen på välkänt sätt. Se figur 5.

```
<order>
  <ordernr> 123 </ordernr>
  <kund>
    <namn> Stina </namn>
    <adress> Storgatan </adress>
  </kund>
  <orderrad>
    <produkt> Vilstol </produkt>
    <antal> 4 </antal>
    <apris> 500 </apris>
  </orderrad>
  <orderrad>
    <produkt> Lampa </produkt>
    <antal> 6 </antal>
    <apris> 250 </apris>
  </orderrad>
  <summa> 3500 </summa>
</order>
```

Figur 5

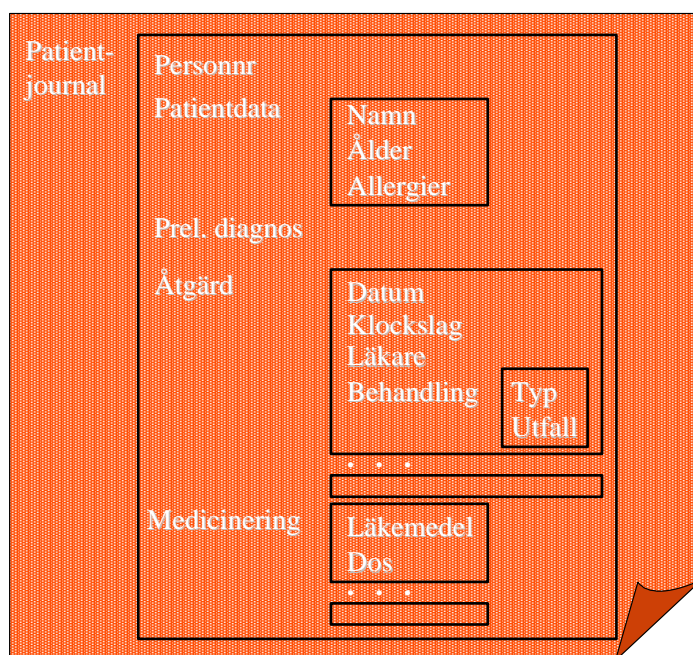
I vissa sammanhang kanske man väljer att bara muntligt komma överens om vilka begrepp som ska användas, speciellt om det är parter som känner varann och ämnesområdet väl. Och om det inte är av någon avgörande betydelse om avsändaren gör någon miss i strukturuppbyggnaden eller om mottagaren till någon del feltolkar innehållet. I andra sammanhang, kanske de flesta, är det av alldeles avgörande vikt att alla förutsättningar för informationsutbytet är reglerade och överenskomna mellan parterna. Dit hör struktur och innehåll på utväxlade dokument. Här kommer dokumenttypsdefinitionen in i bilden. Ta exempelvis dokumenttypen för order. Där gäller det att definiera den struktur och innehåll som visas i figur 4. Observera att vi här helt bortser från layout på ordern, d v s var i dokumentet de olika delarna ska placeras, vilka fonter som ska användas mm. Layouten är något för mottagaren att senare ta ställning till efter egna behov. Kanske behöver definitionen förutom strukturkravet också kompletteras med ytterligare villkor, som hur många orderrader som högst får finnas, att adress får utelämnas och så vi-

dare. Allt för att så långt möjligt precisera förutsättningarna för utbytet så att missförstånd eller oklarheter undviks och så att budskapet svarar mot något rimligt tolkbart.

På samma sätt som vi behöver generella begrepp för att prata om de olika komponenterna i orderdokument behöver vi generella begrepp för att prata om de olika komponenterna i en dokumenttyp. Varför då, kan någon undra? Jo, om parterna ska kunna komma överens om vilken eller vilka dokumenttyper som ska gälla för dokumentutbyte dem emellan måste de kunna diskutera och notera de olika ingredienserna i en dokumenttyp i begrepp man gemensamt är överens om innebörden av.

Nu blir det lite knivigare. När det gällde order kunde vi se att de alla innehöll ett antal komponenter parterna gemensamt valt begrepp för nämligen *kund*, *orderrad*, *adress*, mfl. Därigenom kunde parterna prata med varann på ett generellt plan med hjälp av dessa begrepp, bland annat för att kunna komma överens om dokumenttypspecifikationen.

Om vi nu ur ett likaledes generellt perspektiv vill kunna prata om dokumenttyper behövs det på motsvarande sätt begrepp för detta. Vad kan vi ur detta perspektiv se finns som typiska komponenter i dokumenttyper? Vad har de alla gemensamt som vi kan sätta namn på? En dokumenttyp finns i figur 4 ovan. En annan som visas i figur 6 har ett helt annat innehåll och en annan struktur.



Figur 6

Dokumenttyper kan finnas för alla upptänkliga behov och med alla upptänkliga varianter på strukturer. Oavsett vilken, måste dokumenttypen kunna beskrivas på ett enhetligt sätt om XML ska bli "lingua franca" för all typ av dokumentutbyte. Alltså, vad har dokumenttyper gemensamt, vilka generella begrepp kan vi nyttja för att beskriva dem? Det gemensamma med dokumenttyperna i figurerna 4 och 6 (liksom för övriga tänkbara) är att de innehåller ett antal typer av element; enkla (t.ex. *Adress*, *Summa*, *Prel. diagnos*) eller sammansatta (t.ex. *Kund*, *Order*, *Åtgärd*). Se, där dyker begreppet *elementtyp* upp. På samma sätt som vi kan tala om *kunden* Stina i *ordern* 123 kan vi nu tala om *elementtypen* kund i *dokumenttypen* order.

W3C valde att i standarden XML 1.0 utnyttja det engelska språket och kalla elementtyp för ELEMENT och dokumenttyp för DOCTYPE.

Även andra begrepp behövs för att uttrycka alla upptänkliga egenskaper hos en dokumenttyp. Tillsammans utgör de en så kallad **begreppsmodell**. Med väl begreppsmodellen definierad gäller det att skapa en tilltalande syntax att inordna begreppen i så att dokumenttyper entydigt kan formuleras. Låt oss för enkelhets skull titta närmare på ELEMENT. I XML 1.0 har man valt att syntaktiskt innesluta varje elementtyp inom hakparenteser och efter första hakparentesen inkludera ett utropstecken. Exempelvis

```
<!ELEMENT namn (#PCDATA)>
<!ELEMENT adress (#PCDATA)>
```

Låt oss gå vidare. De olika elementtyperna är inbördes ordnade i en hierarkisk struktur. Denna struktur måste också beskrivas. Det skulle i princip kunna ske med hjälp av exempelvis begreppet SUBORDINATE i kombination med någon syntax. Istället valde man att ange underelement i den hierarkiska strukturen genom att på den övre nivån innesluta underelementen inom parentes. Exempel:

```
<!ELEMENT kund (namn, adress)>
```

Vår ordertyp definieras i XML 1.0-syntaxen (DTD-språket) som framgår av figur 7.

```

<!DOCTYPE order[
  <!ELEMENT order (ordernr, kund, orderrad+, summa)>
  <!ELEMENT ordernr (#PCDATA)>
  <!ELEMENT kund (namn, adress?)>
    <!ELEMENT namn (#PCDATA)>
    <!ELEMENT adress (#PCDATA)>
  <!ELEMENT orderrad (produkt, antal, apris)>
    <!ELEMENT produkt (#PCDATA)>
    <!ELEMENT antal (#PCDATA)>
    <!ELEMENT apris (#PCDATA)>
  <!ELEMENT summa (#PCDATA)>
]>

```

Figur 7

I XML 1.0 kallas dokumenttypen för en Document Type Definition eller kort och gott DTD.

En dokumenttyp kan i princip uttryckas på en mängd olika sätt. Allt beror på hur betraktaren uppfattar de generella egenskaperna, vilken abstraktion denne gör i perspektiv av syfte, konventioner, mm. Istället för ELEMENT kunde vi lika gärna ha använt begreppet KOMPONENT, INGREDIENS, OBJEKT, INFORMATIONSTYP, W3C som "ansvarig betraktare" och representant för alla användare gjorde sitt val i XML 1.0. Specifikationen har fått stort genomslag, d v s en slags begreppsöverenskommelse var etablerad.

In på arenan träder XML Schema. Arbetsgruppen har uppgiften att ta fram en begreppsmodell som erbjuder bättre uttrycks kraft än vad begreppen i XML 1.0 klarar av för att uttrycka DTDer. Som snart kommer att framgå av nästa avsnitt har man valt att uppfatta dokumenttyper på ett i vissa stycken helt annat sätt än XML 1.0. Avsnitt 3 diskuterar innehållet i XML Schema Part 1: Structures medan avsnitt 4 ägnas åt XML Schema Part 2: Data Structures.

Notera också att man helt naturligt har valt XML som syntax. Dokumenttyper är ju i det aktuella perspektivet att se som vilka data som helst. Vi får inte hänga upp oss på att det är fråga om beskrivningsdata. Även denna typ av data är data. Vad kan i det läget vara lämpligare än att uttrycka dokumenttyper med hjälp av XML? Ändring av syntaxen så att schemat uttrycks som ett XML-dokument har ansetts vara en av de fundamentala ändringarna jämfört med DTD. Att man inte från början valde denna syntax är en gåta.

Låt oss se vad arbetsgruppen hittills kommit fram till.

Structures

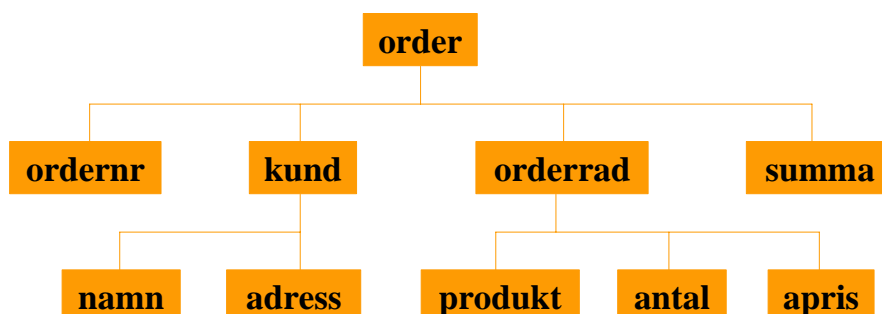
Först ett par brasklappar: Av utrymmesskäl diskuteras endast ett urval av de mer framträdande egenskaperna i Part 1. För att inte onödigtvis tynga ner framställningen visas vissa exempel med en del syntaktiska förenklingar/utelämnningar. Den som avser använda XML Schema i praktiskt arbete bör konsultera specifikationen för exakta regler, exakta syntaktiska krav.

Grundläggande struktureringsegenskaper

Låt oss nu i ett antal steg diskutera hur order skulle definieras om det uttryckes med hjälp av XML Schema-syntax istället för DTD-syntax. Notera återigen att XML-syntaxen är helt oförändrad. Det betyder att XML-dokument ser ut som de alltid har gjort med märkord, element, attribut, hierarkisk uppbyggnad, mm. Till exempel som i figur 5 ovan.

En DTD deklarerar hur dokument ska vara uppbyggda och vilket innehåll de får ha för att accepteras. En DTD kan grovt sett ses som en dokumentmall. Språket i XML Schema har som tidigare nämnts samma syfte som DTD-språket. Skillnaden ligger i att XML Schema har valt att se på dokument på ett något annorlunda sätt, av vilket följer en annan uppsättning modellbegrepp. Vissa begrepp känns igen från DTD-språket, andra är helt nya. Dessutom använder sig XML Schema av andra språkkonstruktioner som på ett betydligt rikare sätt kan uttrycka villkor och restriktioner för acceptabla dokument.

Vi startar med en syntaxneutral hierarkisk bild av orderdokumentets struktur så som XML 1.0 ser på den (figur 8).

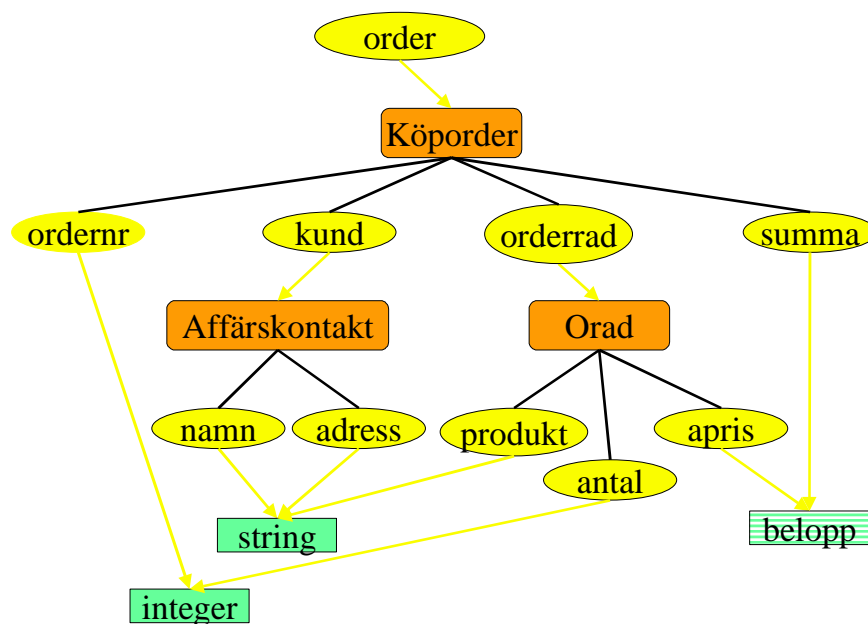


Figur 8

Strukturen visar namnen på alla förekommande element. Av strukturen framgår att elementen order, kund och orderrad vardera representerar eller pekar på en understruktur. Var och en av dessa tre understrukturer beskrivs

med sin uppsättning element. I XML Schema kallas dessa understrukturer för *Complex Types*. (Även typer som visserligen inte har en explicit understruktur men väl beskrivs av ett eller flera attribut klassas som *Complex Types*.)

De typer som uttrycker explicita värden (heltal, text, datum, ...) sägs vara på elementär nivå ("löv"). De kallas *Simple Types*. Varje beståndsdel i ett dokument kallas *Element* oavsett om det är enkelt eller sammansatt. Varje Element uttrycks med hjälp av antingen en *Complex* eller *Simple Type* beroende på om den är sammansatt eller ej. Exempelvis uttrycks elementet *antal* i ordern rimligtvis med hjälp av Simple Type *Integer*. Element kund uttrycks med hjälp av en *Complex Type* som i sin tur omfattar elementen namn och adress. Osv. Vi ritar om figur 8 med användande av de nyintroducerade begreppen (figur 9).



Figur 9

Vi hittar alla begreppen från figur 8 som Element, visade som ovaler. Rektanglar med rundade hörn är *Complex Types* som vi nu gett namn. Rektanglarna med spetsiga hörn är *Single Types*. En streckmönstrad Simple Type betyder att den är användardefinierad. Fördefinierade Simple Types har jämn färg. Alla Element utom order finns grupperade under en *Complex Type*. Elementet order på översta nivå – det Element som pekar på ordern som helhet - tänks i XML Schema utgå ifrån eller tillhöra en generell över-

ordnad dokumentnivå som av någon anledning går under benämningen *ur-type*.

Dags börja föra in figur 9 i XML Schemasyntax. Både Complex Types och Simple Types behöver definieras. Ett av flera sätt att göra det är i form av en explicit Complex Type respektive Simple Type definition. En explicit Complex Type definition uttrycker en namngiven intern struktur. I en Simple Type definition definieras de värden som är giltiga för typen ifråga. Som en service finns i XML Schema ett antal fördefinierade Simple Types. Dessa behöver inte definieras. Vi återkommer i avsnitt 4 till hur egendefinierade Simple Types, t.ex. belopp, kan formuleras.

Över till schemadefinitionen för ordern.

Först ut är definition av Complex Type på översta nivå. I DTD noteras en understruktur genom att den omges av parenteser. En explicit Complex Type definition uttrycks istället i XML-syntax samt ges ett explicit namn. Vi väljer namnet *Köporder*. Se figur 10.

```
<complexType name="Köporder">
  <element name="ordernr" type="integer"/>
  <element name="kund" type="Affärskontakt"/>
  <element name="orderrad" type="Orad"/>
  <element name="summa" type="belopp"/>
</complexType>
```

Figur 10

Som synes talar vi om att Köporder är en Complex Type med hjälp av begreppet/märkordet *complexType*. Därefter följer elementen på närmaste nivå under. Varje element ges ett namn. Ett element refererar i sin tur till vilken typ det pekar på eller uttrycks med hjälp av. Hur man formulerar villkor för hur många gånger respektive element får/måste anges återkommer vi till i avsnitt 3.3 nedan. Om kravet är att uppgifterna i dokumenten ska anges i samma ordning som de deklarerats i schemat (vilket i det här fallet verkar naturligt) ska elementen omges med märkordet *<sequence>* enligt figur 11.

```

<complexType name="Köporder">
  <sequence>
    <element name="ordernr" type="integer"/>
    <element name="kund" type="Affärskontakt"/>
    <element name="orderrad" type="Orad"/>
    <element name="summa" type="belopp"/>
  </sequence>
</complexType>

```

Figur 11

Vore valfri ordning acceptabel skulle <choice> angetts istället. Med <all> gäller att elementen må anges i valfri ordning men med restriktionen att varje element får förekomma högst en gång.

För att inte onödigtvis tynga ner kommande exempel har vi valt att i fortsättningen utelämna specifikation av önskad ordning.

Typen "integer" är fördefinierad. Däremot behöver de övriga typerna definieras, "Affärskontakt" och "Orad" som Complex types och "belopp" som Simple Type. Se figur 12. Hur belopp definieras återkommer vi till under avsnitt 4.

```

<complexType name="Affärskontakt">
  <element name="namn" type="string"/>
  <element name="adress" type="string"/>
</complexType>
<complexType name="Orad">
  <element name="produkt" type="string"/>
  <element name="antal" type="integer"/>
  <element name="apris" type="belopp"/>
</complexType>
<simpleType name="belopp">
  .....
</simpleType>

```

Figur 12

Observera att vi valt olika namn på elementet och den Complex Type elementet pekar på. Elementnamnet anger primärt den komplexa typens roll just i det aktuella sammanhanget, d v s som komponent under order. Samma komplexa typ kan förekomma i andra roller, inom andra understrukturer. Kanske vill vi i en dokumenttyp *offert* referera till Affärskontakt med hjälp av elementet *leverantör*.

Observera att det är elementen som sedan kommer att användas som märkord (taggar) i respektive orderdokument.

XML Schemat i sin helhet så här långt visas i figur 13.

```
<schema>
  <element name="order" type="Köporder"/>
  <complexType name="Köporder">
    <element name="ordernr" type="integer"/>
    <element name="kund" type="Affärskontakt"/>
    <element name="orderrad" type="Orad"/>
    <element name="summa" type="belopp"/>
  </complexType>
  <complexType name="Affärskontakt">
    <element name="namn" type="string"/>
    <element name="adress" type="string"/>
  </complexType>
  <complexType name="Orad">
    <element name="produkt" type="string"/>
    <element name="antal" type="integer"/>
    <element name="apris" type="belopp"/>
  </complexType>
  <simpleType name="belopp">
    .....
  </simpleType>
</schema>
```

Figur 13

Förutom de inledande och avslutande märkorden för schemat inleder definitionen med elementdeklarationen för elementet på den översta nivån nämligen order.

Ett alternativt sätt att definiera samma sak visar figur 14.

```

<schema>
  <element name="order">
    <complexType>
      <element name="ordernr" type="integer"/>
      <element name="kund">
        <complexType>
          <element name="namn" type="string"/>
          <element name="adress" type="string"/>
        </complexType>
      </element>
      <element name="orderrad">
        <complexType>
          <element name="produkt" type="string"/>
          <element name="antal" type="integer"/>
          <element name="apris">
            <simpleType>
              .....
            </simpleType>
          </element>
        </complexType>
      </element>
      <element name="summa">
        <simpleType>
          .....
        </simpleType>
      </element>
    </complexType>
  </element>
</schema>

```

Figur 14

Skillnaden gentemot figur 13 är att vi undvikit explicita namn på egendefinierade typer. Istället "bakas de in" på de platser varifrån de tidigare refererades. Kan vara praktiskt om en Type bara refereras från ett ställe. Den fulla definitionen finns "på plats" samtidigt som vi slipper hitta på ett explicit namn för den. Betalningen sker i form av sämre flexibilitet. I vårt fall innebär det att beloppsdefinitionen måste göras på två ställen, se de båda SimpleType-deklamationerna.

Köpordern innehåller inga attribut nånstans men i likhet med DTD-språket kan attributdeklamationer hanteras. Attribut är med andra ord ett begrepp med i stort sett likalydande innebörd i båda ansatserna även om de hanteras i olika språksyntaxer.

Antag att någon valt att formulera vår kända order enligt figur 15 nedan istället för den tidigare versionen enligt figur 4 ovan.

```

<order ordernr="123" summa="3500">
  <kund namn="Stina" adress="Storgatan"/>
  <orderrad produkt="Vilstol" antal="4" apris="500"/>
  <orderrad produkt="Lampa" antal="6" apris="250"/>
</order>

```

Figur 15

Båda är helt korrekta XML-dokument men i figur 15 anges de enkla värdena ("löven") som attribut istället för som element. Attribut uttrycks alltid med ett värde, d v s alltid som en Simple Type. Schemadeklarationen måste anpassas därefter.

Figur 16 visar hur det blir om vi väljer att, som i figur 13 ovan, återvända till principen med explicita namn för egendeklarerade typer.

```

<schema>
  <element name="order" type="Köporder"/>
  <complexType name="Köporder">
    <element name="kund" type="Affärskontakt"/>
    <element name="orderrad" type="Orad"/>
    <attribute name="ordernr" type="integer"/>
    <attribute name="summa" type="belopp"/>
  </complexType>
  <complexType name="Affärskontakt">
    <attribute name="namn" type="string"/>
    <attribute name="adress" type="string"/>
  </complexType>
  <complexType name="Orad">
    <attribute name="produkt" type="string"/>
    <attribute name="antal" type="integer"/>
    <attribute name="apris" type="belopp"/>
  </complexType>
  <simpleType name="belopp">
    .....
  </simpleType>
</schema>

```

Figur 16

Den som så önskar kan välja att gruppera ett antal attribut under ett gemensamt gruppnamn. I figur 17 är *kunddata* en *Attribute Group* sammanförande namn och adress. "..." inom Affärskontakt-deklarationen är till för att indikera att kund mycket väl kan beskrivas av ett antal olika element/attribut varav kunddata bara är en delmängd.

```
<schema>
...
  <complexType name="Affärskontakt">
    ...
    <attributeGroup=ref "kunddata"/>
  </complexType>
...
  <attributeGroup name="kunddata">
    <attribute name="namn" type="string"/>
    <attribute name="adress" type="string"/>
  </attributeGroup>
...
</schema>
```

Figur 17

Vad är då vitsen med Attribute Groups? Dels kan uppdelningen upplevas vara mer lättläst, dels – och betydligt viktigare – kan nu samma grupp attribut genom sitt namn refereras från flera typdeklarationer som har behov av samma uppsättning attribut. Praktiskt och än mer lättläst. En annan sak är hur det rent semantiskt ska uppfattas. Namn och adress för Affärskontakt kanske inte har riktigt samma semantiska valör som namn och adress för exempelvis den säljande butiken. Dock en fråga vi här väljer att inte ta ställning till.

Överlag inga större sensationer. I princip kan den som definierar dokumenttyper fritt välja att deklarerar en Simple Type som element eller Attribute så länge det bara är fråga om att antingen ge det ett eller inget värde i dokumentet. XML-syntaxen tillåter ju inte flera attributvärden av samma typ. Ett Attribute kan av naturliga skäl inte heller referera till en Complex Type.

Valfriheten mellan Element och Attribute är snarare en belastning än en tillgång för envärdiga uppgifter eftersom den inte ger definieraren någon väg-

ledning om när vilken är att föredra. Samma dokumentinnehåll kan paketeras på onödigt många alternativa sätt som inte var för sig har några unika egenskaper. Rekommendationen måste bli att i normalfallet nyttja elementdeklarationen eftersom den ger större flexibilitet, bland annat när det gäller antal uppgifter per elementtyp.

Kompletterande namespace-deklarationer

För att bli en komplett definition behöver ytterligare ett par deklarationer tillföras schemat. Motiveringen till dessa är lättare att förstå om vi till att börja med kompletterar orderdokumentet i figur 5 med vad som hittills saknats där, nämligen med en Namespace-deklaration. Den läsare som inte är familjär med Namespaces i XML hänvisas till en introduktion i xxxxxxxxxxxxxxxx. Se figur 18.

```
<order xmlns="http://www.postordersammanslutningen.org/Orderhantering">
  <ordernr> 123 </ordernr>
  ...
  <summa> 3500 </summa>
</order>
```

Figur 18

Elementet "order" har kompletterats med attributdeklarationen

```
xmlns="http://www.postsammanslutningen.org/Orderhantering"
```

Den talar om att begreppet "order" och alla de begrepp som befinner sig i understrukturen till "order" hämtar sin tolkning från angiven Internetadress. I det här fallet är det tydligen begrepp som *postordersammanslutningen* definerat i och för användning inom *Orderhantering*. Deklarationen underlättar entydig tolkning av innehållet i dokumentet.

Tillbaka till schemat i figur 13. Schemat är också ett dokument med sina begrepp/märkord. Det är visserligen inte fråga om orderdokument utan om schemadokument, men för övrigt att se som två dokument vart och ett uttryckt i XML-syntax. Att det senare beskriver villkoren för hur dokument får vara formulerade är i sammanhanget att se som vilken tillämpning som helst.

Även schemadokumentet behöver referens till ett namespace där "schema", "complexType", "element", med flera begrepp finns förklarade. Detta namespace är definitionsmässigt en av W3C underhållen fil

<http://www.w3.org/2000/10/XMLSchema> som avsatts för ändamålet. Deklarationen görs enligt samma princip som i orderdokumentet i form av ett attribut under den inledande schema-deklarationen:

```
<schema xmlns = "http://www.w3.org/2000/10/XMLSchema">
```

varefter alla dokumenttypens märkord underförstått har sin definition i deklarerat namespace.

I schemadeklarationen noteras för fullständighets skull även i attributet targetNamespace det namespace som gäller för de element- och attributtyper som deklarerats i schemat, t.ex. order, Affärskontakt, namn, Detta namespace är naturligen samma som det vi just deklarerat på översta elementnivå i orderdokument. Se figur 19.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.postordersammanslutningen.org/Orderhantering">
  <element name="order" type="Köporder"/>
  <complexType name="Köporder">
  ...
</schema>
```

Figur 19

Användningen av Namespaces i schemadefinitioner och dess implikationer på dokumentnivå kräver sitt eget omfattande kapitel. I sak går det att förstå XML Schemas huvudsakliga egenskaper utan att komplicera framställningen med detta varför vi väljer att inte vidare beröra temat i denna introduktion. För en uttömmande beskrivning hänvisas läsaren till XML Schema-specifikationen.

Villkor på element- och attributtyper

Hur deklarerar man då de villkor som ska gälla antalsuppgift för en viss typ av element? Jo med *minOccurs* och *maxOccurs*. Först och främst kan konstateras att om man inte säger någonting gäller alltid en uppgift, d v s *minOccurs*="1" och *maxOccurs*="1". Från DTD-deklarationen i figur 7 ovan minns vi ett frågetecken efter adress vilket ska tolkas som att högst en adressuppgift får förekomma men att den får utelämnas. Elementraden för adress i figur 13 ändras till

```
<element name="adress" type="string" minOccurs="0"
maxOccurs="1"/>
```

Notera att *maxOccurs* skulle kunna utelämnas eftersom dess defaultvärde är 1.

Antalsuppgiften för orderrad måste också ändras. Plus-markeringen i figur 7 betyder en eller flera orderrader. Vi uttrycker motsvarande genom att ge *maxOccurs* värdet "unbound", d v s valfritt antal rader. Elementet orderrad under Köporder omdeklarerar enligt

```
<element name="orderrad" maxOccurs="unbound">
```

Eftersom *minOccurs* utelämnats gäller minst 1 orderrad.

Antag att vi istället för en sammanhängande adresstext önskar redovisa den i ett antal fristående rader, dock max fyra stycken. Vi döper om adress till adressrad för att bättre spegla innehållet.

```
<element name="adressrad" type="string" minOccurs="0"
maxOccurs="4"/>
```

Om villkoret vore högst en adress, men med bivillkoret att om den förekommer så måste den bestå av minst två rader, måste vi återinföra adresselementet för helheten (figur 20).

```
<element name="adress" minOccurs="0" maxOccurs="1">
  <complexType>
    <element name="adressrad" type="string" minOccurs="2" maxOccurs="4"/>
  </complexType>
</element>
```

Figur 20

Som synes valde vi här alternativet att "baka in" typdeklarationen för adressraderna istället för att deklarera den separat med explicit namn.

Förutom `minOccurs` och `maxOccurs` kan attributet `fixed` användas när man kräver ett visst värde på uppgiften och attributet `default` användas för att deklarera att om inget annat värde angivits så ska defaultvärdet användas.

Affären säljer enbart sin egentillverkade fotogenlampa. Alltså kan endast order med produktnamnet "fotogenlampa" accepteras (lite långsökt men ändå). Har man sedan den lite luriga regeln som säger att om antalsuppgiften skulle utebli så betyder det två exemplar, blir deklarationerna

```
<element name="produkt" type="string" fixed="fotogenlampa"/>
<element name="antal" type="integer" default="2"/>
```

För attribut gäller andra uttryck. Ingen eller en förekomst uttrycks med hjälp av värdet `optional` respektive `required` vid beskrivningsattributet `use`. Önskar man ett fixt värde som alltid ska förekomma sätts `use` till `required` och beskrivningsattributet `value` till värdet ifråga. Gäller villkoret att om värdet är angivet så måste det vara ett visst värde anges `use="fixed"` och `value="xxxx"`. Byter vi nu ut `fixed` mot `default` betyder det att om inget värde anges så ska det sättas till `xxxx`.

Extension och restriction

Extension

Nu närmar vi oss något mer komplicerat. I datamodeller har det närmast sin likhet i specialiseringssamband eller arvsdeklarationer. Antag att vissa av våra kunder inte bara klassas som Affärskontakt utan även som så kallade VIPkund, en kund med en speciell förmån i form av generell rabattsats på köpesumman. För dessa kunder tillkommer med andra ord elementet *Rabattsats* utöver elementen *Namn* och *Adress* som gäller för alla affärskontakter inklusive normala kunder. Istället för att deklarera Complex Type *VIPkund* med *namn*, *adress* och *rabattsats* specificerar vi endast det som gäller unikt för den kategorin, d v s *rabattsats*. *Namn* och *adress* "ärver" *VIPkund* från *Affärskontakt* genom att ange att den är en viss kategori eller specialisering av *Affärskontakt*. Helt rimligt eftersom en *VIPkund* ju trots allt också är en vanlig *affärskontakt*. Inom data- och objektmodellering har specialiseringar en lång tradition som en effektiv, preciserande uttrycksform. XML Schema

har alltså begåvats med en motsvarighet, uttryckt som en extension. Figur 21 visar deklARATIONEN i en något förenklad form. (Egentligen borde mellan märkorden *complexType* och *extension* även märkordet *complexContent* finnas med av skäl som gott kan utelämnas i denna introduktion.)

```
<complexType name="VIPkund">
  <extension base="Affarskontakt">
    <element name="rabattsats" type="integer"/>
  </extension>
</complexType>
```

Figur 21

Gott och väl men hur kan vi utnyttja de båda typerna i orderdokument så att de kunder som är VIPkunder utöver namn och adress också tillåts ange rabattsats medan de som inte är VIPkunder inte får göra det? Som det nu står gäller att kund ska anges som märkord. Ingenstans framgår om det är en vanlig kund (Affarskontakt) eller en VIPkund. Är alternativet måne att definiera ett speciellt VIPorder-schema som ska nyttjas när det är fråga om VIPkunder? Låter klumpigt, speciellt som det inte alls är otroligt att nya kategorier kan tillkomma efterhand var och en med krav på eget order-schema.

XML Schema har löst problematiken på så sätt att schemat lämnas oförändrat med angivande av den mest generella typen, d v s kund. Vilken typ som i realiteten gäller anges i respektive dokument. Den som genererar dokumentet vet förhoppningsvis vilken typ som gäller för den aktuella kunden. varefter Den kontrollerande rutinen hos mottagaren (parser) kan nu anpassa kontrollen efter den angivna typen. Jämför med objektorienteringens snarlika polymorphism-begrepp. Sofia från figur 5 är faktiskt VIPkund. Dokumentet ändras därför enligt figur 22 där rabattsats nu helt korrekt kan accepteras i enlighet med vad som deklarerats i schemat för VIPkund.

```

<order>
  <ordernr> 124 </ordernr>
  <kund type="VIPKund">
    <namn> Sofia </namn>
    <adress> Drottninggatan </adress>
    <rabattsats> 15 </rabattsats>
  </kund>
  ...

```

Figur 22

Att som i XML Schema kalla detta för "Extension" kan lätt föra tankarna fel. Visserligen är det en utökning med ytterligare en eller flera elementrader i dokumenten men konceptuellt snarare att se som en specialisering, en avgränsning av en generellare population objekt.

Restriction

Extension ovan uttrycker kompletterande alternativ/villkor för giltigt innehåll, nämligen möjlighet/krav på ytterligare element och/eller attribut endast en viss namngiven delmängd av den ursprungliga populationen svarar upp emot. Ett annat sätt att komplettera med strängare villkor är att visserligen hålla fast vid samma innehåll som i en redan deklarerad Complex Type men att för övrigt ställa hårdare krav på ett eller flera av elementen i form av antal förekomster, mm. På så vis sker också en avgränsning till den delmängd av den ursprungliga populationen som uppfyller kraven. Kraven deklarerar i en separat namngiven Complex Type med referens till den Complex Type man baserar sig på.

Ett exempel: Vår aktuella verksamhet använder sig av hävd av begreppet Stororder för att indikera sådana order som omfattar minst 11 orderrader. Stororder innehåller för övrigt exakt samma element som Köporder. Deklarationen blir som i figur 23. Detta är i praktiken också en form av specialisering. En viss Stororder är ju samtidigt en Köporder.

```

<complexType name="Stororder">
  <restriction base="Köporder">
    ....
    <element name="orderrad" type="Orad" minOccurs="11" maxOccurs="unbound"/>
    ....
  </restriction>
</complexType>

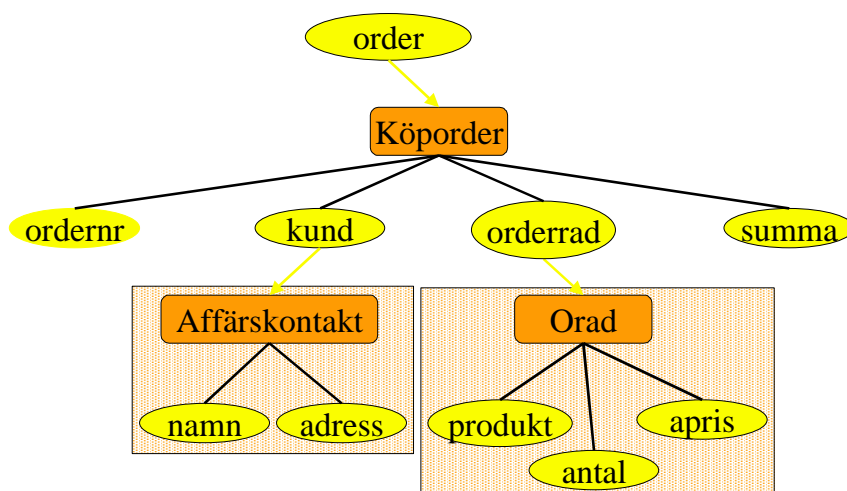
```

Figur 23

Restriction-faciliteten kommer också väl till pass när vi önskar definiera egna Simple Types baserade på redan existerade. Se vidare avsnitt 4.

Fragmenterat schema

Ett schema omfattande många element och attribut, dessutom uppdelade i en flernivå hierarkisk struktur kan bli tung att läsa, svår att underhålla på ett korrekt sätt. En facilitet som tillåter ett schema att vara uppbyggt av flera fristående delar skulle avsevärt kunna underlätta överblickbarhet. XML Schema tillåter sådan uppdelning. Exempelvis skulle vi kunna bryta ut Affärskontakt- och Orderraddeklarationerna som två separata delar (underdokumenttyper) om vi så önskar.



Figur 24

Varje del utgör ett subschema, d v s innehåller valda delar av det fulla schemat.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.postordersammanslutningen.org/Orderhantering">
  <complexType name="Affärskontakt">
    <element name="namn" type="string"/>
    <element name="adress" type="string"/>
  </complexType>
</schema>
```

Figur 25

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.postordersammanslutningen.org/Orderhantering">
  <complexType name="Orad">
    <element name="produkt" type="string"/>
    <element name="antal" type="integer"/>
    <element name="apris" type="belopp"/>
  </complexType>
</schema>
```

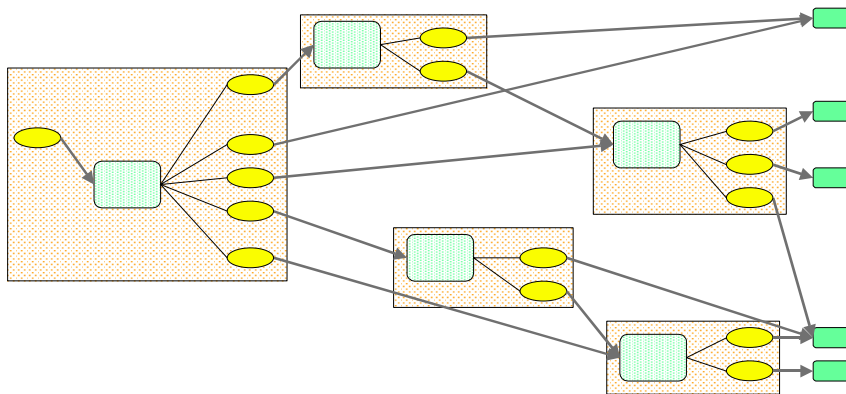
Figur 26

Dessa delar infogas i det övergripande schemat med hjälp av märkordet `<include>` vars attribut `schemaLocation` ges adressen till respektive subschema.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.postordersammanslutningen.org/Orderhantering">
  <include schemaLocation="http://www.postordersammanslutningen.org/kund.xsd"/>
  <include schemaLocation="http://www.postordersammanslutningen.org/orderrad.xsd"/>
  <element name="order" type="Köporder"/>
  <complexType name="Order">
    <element name="ordernr" type="integer"/>
    <element name="kund" type="Affärskontakt"/>
    <element name="orderrad" type="Orad"/>
    <element name="summa" type="belopp"/>
  </complexType>
  <simpleType name="belopp">
    .....
  </simpleType>
</schema>
```

Figur 27

Ett konsekvent nyttjande av denna facilitet kan resultera i en större uppsättning scheman som i olika kombinationer inordnas som delbeskrivningar under andra scheman över ett antal nivåer. Figur 28 visar fem olika subscheman som alla i detta fall direkt eller indirekt utnyttjas för att definiera den längst till vänster. Dock gäller restriktionen att samtliga dessa scheman måste referera till samma target namespace, d v s representera samma ämnesområde, syfte e.dyl.



Figur 28

Notera att include inte erbjuder någon ny typ av datamodellingsfacilitet utan endast inkopiering av delar av ett schema som av olika anledningar beskrivits separat. De dokument som har att svara mot schemat ser exakt likadana ut oavsett om schemat definierats som en homogen enhet eller uppdelat i ett antal fristående subscheman. Frågan är om den som skapar dokument upplever fragmenteringen som ett stöd eller som ett hinder. För den som läser figur 27 ger knappast include-instruktionen någon vidare vägledning om vad som döljer sig bakom webbadressen. Såvida inte något stödjande dokumentgenereringsverktyg används förstås.

Datatyper

Version 1.0 erbjöd endast en datatyp nämligen PCDATA (Parsed Character Data), vilket står för alla upptänkliga teckenkombinationer. Inte minst inom e-affärer behövs större precision för att, om inte garantera, så ändå kraftigt öka sannolikheten för korrekta uppgifter. Bland användbara datatyper kan nämnas Datum med formatet åå-mm-dd. En möjlig ytterligare precision (specialisering) på Datum kan vara Startdatum som måste ligga inom visst intervall eller kanske tidigast 99-07-30. En annan datatyp kan vara Veckodag med de endast giltiga värdena "må", "ti", "on", "to", "fr", "lö", "sö". Ytterligare en kan vara Dagnr med giltiga värden 1-31 (dagarna i en månad). Och så vidare i all oändlighet.

XML Schema erbjuder ett 40-tal fördefinierade atomära Simple Types allt ifrån string, integer över date, time till IDREF, NMTOKEN. Därutöver finns tre Simple Types som är listor av atomära värden. Dessa tre är NMTOKENS, IDREFS och ENTITIES.

Förutom denna rikhaltiga uppsättning fördefinierade datatyper kan tillämpningsspecifika behov ställa krav på ytterligare typer, exempelvis läsårsintervall eller prisnivå eller varumärkesskod eller skonummer eller Eller varför inte *belopp* i vårt orderexempel? Villkor för ett korrekt belopp är enligt överenskommelse mellan de samverkande parterna att det är ett positivt heltal i intervallet 10-10000. Definitionen innebär en restriktion på den fördefinierade typen `positiveInteger` enligt följande:

```
<simpleType name="belopp">
  <restriction base="positiveInteger">
    <minInclusive value="10">
    <maxInclusive value="10000">
  </restriction>
</simpleType>
```

Figur 29

Ville vi definiera Veckodag enligt ovan blir definitionen

```
<simpleType name="veckodag">
  <restriction base="string">
    <enumeration value="må">
    <enumeration value="ti">
    ....
  </restriction>
</simpleType>
```

Figur 30

Därutöver finns ett språk för att beskriva krav på ett specifikt utseendemönster. Antag exempelvis ett värde som måste börja med någon av alfabetets fem första bokstäver följt av siffror följt av ett bindstreck följt av X, Y eller Z. Mallen blir (tror jag)

```
[a-e]\d+-[XYZ]
```

Exemplen är bara ett smakprov. Det skulle föra för långt att här penetrera alla möjliga alternativa konstruktioner. Förhoppningsvis ger de en god fingervisning om den kapacitet som står till buds för att deklarerar egna datatyper.

Några avslutande synpunkter

XML Schema innehåller många fler uttrycksmöjligheter än vad denna introduktion förmår redovisa. Vi har bara skrapat på ytan. Är det tecken på XML Schemas styrka eller svaghet? Redan det vi tangerat ställer krav på en gedigen uppfattning om de olika syntaktiska konstruktionernas innebörd och på en strikt, konsekvent användning av dem. Blir uttryckskraften användbar för gemene man eller kommer vi återigen att behöva vända oss till experter, behöva uppleva dimridåerna bakom teknikens mysterier? Det vore olyckligt. Samtidigt ställer förstås alltmer avancerat informationsutbyte allt större krav på precision. Har man hittat den rätta balansen? Har man hittat den rätta infallsvinkeln? Knappast. Här följer några argument.

XML Schema består av två normativa delar, Part 1: Structures som omfattar ca 150 sidor och Part 2: Datatypes som omfattar ca 125 sidor. Framförallt Part 1 är ett alldeles utmärkt exempel på hur i grunden sunda principer och vettiga faciliteter kan krossas under en massiv lavin av ogenomtränglig text. Fullständigt kompakt och otolkbart. Ibland hjälper det inte ens att läsa om samma stycke två-tre gånger. Hur många personer kan med rimlig

trovärdighet påstå sig ha gedigen kunskap och överblick idag? Det skulle inte förvåna om till och med arbetsgruppens egna medlemmar tolkar vissa avsnitt olika.

Blir det så här när en mindre grupp människor sitter och filar och lappar och putsar och reviderar och kompromissar alltför länge? De är säkerligen sällsynt väl insatta i XML, SGML, mm. Men däri ligger återigen både en styrka och en fara. Risken är att de ser standarden som ett skötebarn att ges bästa tänkbara vård genom att inkludera allt, skapa det perfekta. Förmodligen har de därutöver att väga in och ta hänsyn till de syften det egna företaget ser med standarden. Kanske orkar gruppen inte med den nödvändiga förankringen hos användargrupper. Risken för att en sådan grupp går in i självsvängning och slår knut på sig själv är uppenbar. Har det måne hänt här?

Frågan är om man ens sökt någon förankring i de discipliner som under många år arbetat med närliggande frågeställningar. Dit hör data- och affärsmodelleringsområdena. Varför har man inte tagit fasta på de begreppsmodeller som där finns att tillgå, genomarbetade, standardiserade modeller med bred förankring och lång tradition. Ta exempelvis Unified Modeling Language (UML). Varför inte använda UMLs begrepp (Class, Attribute, Association, Generalization, ...) där de passar istället för att hitta på egna som inte har någon stabil tradition och följdriktigt knappast ger läsaren de stödjande associationer eller aha-tolkningar som är så viktig för entydigare förståelse?

Om lösningen i sig är komplex, vilken den kanske måste vara, är en "kompenserande" välgenomtänkt pedagogisk specifikation inte bara efterlängtdad utan helt avgörande för specifikationens entydiga tolkning och spridning. I detta fall har vi ett komplext tema i kombination med en rörig presentation. Knappast en bra plattform för en framgångsrik standard. Med sannolikt endast ett fåtal rimligt väl insatta personer är det knappast riskfyllt att påstå att specifikationen saknar konsensus. Snarare är den en tidsinställd bomb. Missförstånd, inkompatibla produkter kommer som ett "brev på posten". Andra konkurrerande ansatser dyker snabbt upp. Arbeta på en revision startar i nästa sekund. Vad blir kvar av den stabila, universellt accepterade plattform för informationsutbyte vi så intensivt hoppats kunna realisera med XMLs hjälp? Kommer visionen om ett intensivt, globalt, finmaskigt flöde av information mellan alla och envar att ske på en motorväg eller över en tjälskottsfylld grusväg?

Eller; kan det vara så väl att specifikationen bara behöver en ordentlig pedagogisk putsning? Part 0: Primer är ett lovvärt initiativ, dessutom väl genomarbetat, men det är icke förty de normativa delarna som entydigt och fullständigt ska kunna tolkas med rimlig ansträngning.

Vem är förresten redo för användning av XML Schema? Antagligen Part 2 – den del som varit mest efterfrågad. Men Part 1? Säkert ytterst få just nu. Förmodligen inte heller de som nu ska försöka åstadkomma mjukvaror som stödjer standarden. De flesta som närmar sig informationsutbytesområdet har fortfarande ett tungt jobb med att ta till sig grunderna i XML och applicera dessa på sin verksamhet. Att intresset då är svalt för XML Schema borde inte överraska. Uteblir den nödvändiga breda uppslutningen riskerar vi hamna i två läger, de som anammar XML Schema och de som håller fast vid DTDer i enlighet med XML 1.0. Ett inte speciellt lockande utfall som till och med kan komma att störa XMLs allmänna acceptans och spridning.

Det finns för övrigt en falang som gärna skulle vilja se en bantning av specifikationen till att endast omfatta de viktigaste datatyperna i kombination med möjlighet till egendefinierade datatyper. Givetvis uttryckta med nyttjande av XML-syntax. En okontroversiell lösning som snabbt skulle kunna bli en accepterad standard (Recommendation). Dessutom lite av samma filosofi som fått XML att slå (i förhållande till SGML) nämligen genom att klara 80% av behoven till 20% av komplexiteten. Övriga modelleringsfinesser kan i lugn och ro vänta till nästa version.

XML Schema har tagit lång tid att ta fram, betydligt längre tid än någon initialt räknade med. Man är ännu inte i mål. Är detta partintressens kamp om herraväldet eller en seriös bearbetning av ett komplicerat problemområde? Är detta månne ytterligare ett orostecken när det gäller den kommande uppslutningen bakom XML Schema?

Oops – blev det månne en alltför stor portion högst personligt färgade negativa synpunkter? Är de orättvisa? Till viss del är det syftet. Att irritera lite grann, att locka till egna personliga ställningstaganden, kanske till vidareläsning av specifikationen. I bästa fall genererar de ett aktiverat intresse för XML Schemas fortsatta öde. Vilket inte är så dumt eftersom, om alla tecken slår in, XML Schema kommer att bli en central ingrediens i nästa generations XML-tillämpningar.