Subproject: *Supporting facilities*

XML Schema – en översikt[1]

Version 0.9
(2004-03-30)

Stig Berild

(Santa Anna IT Research Institute AB)

[1] Rapport inom Serviam-projektet. Projektet har en egen webbplats: www.serviam.se.

1 SYFTE 3

2 INLEDNING 4

2.1 Bakgrund 4

2.2 Dokumentstrukturer generellt 5

2.3 Dokumentstrukturer beskrivna i XML 1.0 8

3 XML SCHEMA, EN ÖVERSIKT 10

3.1 Inledning 10

3.2 Grundläggande struktureringsregler 10

3.3 Strukturen uttryckt i XML Schema 14
3.3.1 Specifikation av complex Types 15
3.3.2 Specifikation av simple Type 17
3.3.3 Avslutande komplettering 18
3.3.4 Alternativ specifikation 18

3.4 Några villkorsspecifikationer 19
3.4.1 Ordningsföljd mellan element 20
3.4.2 Villkor för viss elementtyp 20

4 XML SCHEMA, LITE MER 22

4.1 Attributdeklarationer 22

4.2 Kompletterande namespace-deklarationer 24

4.3 Extension och restriction 25
4.3.1 Extension 25
4.3.2 Restriction 27

4.4 Fragmenterat schema 27

5 NÅGRA AVSLUTANDE SYNPUNKTER 30

1 Syfte

XML Schema är en standard (Recommendation) framtagen av World Wide Web Consortium (W3C). Syftet är att åstadkomma ett betydligt kraftfullare språk för dokumenttypsspecifikation än vad XML 1.0-standarden erbjuder med sitt DTD-språk (DTD=Document Type Definition).

Syftet med denna rapport är att ge en introduktion till de viktigaste syntaktiska konstruktionerna i XML Schema. Kapitel 2 diskuterar det övergripande syftet med standarden 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 kapitel. Kapitel 3 beskriver några av de mer centrala faciliteterna för dokumentstrukturbeskrivning medan kapitel 4 exemplifierar några fler uttrycksmöjligheter. I kapitel 5 slutligen vågar vi oss på några kritiska synpunkter.

2 Inledning

2.1 Bakgrund

Alla tycks vara överens om att XMLs DTD-språk enligt standarden 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. I XML 1.0 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 deklarera 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 tillgång till standardiserade datatyper bör egna datatyper kunna specificeras för unika behov, exempelvis olika typer av tidsintervall och symbolsekvenser.

Att DTDer uttrycks med en egen syntax istället för i XML är en annan besvärande barlast.

Väl medveten om bristerna initierade W3C redan1998 en aktivitet som skulle utmynna i ett språk för dokumenttypsspecifikationer inkluderande de identifierade behoven. Dessutom var kravet att dessa specifikationer skulle uttryckas i XML och inte i enlighet med någon specialkonstruerad syntax. Arbetet och standarden 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)

Efter mycket kompromissande kunde W3C i maj 2001 anta den då genomrabetade specifikationen som en Recommendation, d.v.s. som en W3C- standard.

Specifikationen för XML Schema är uppdelad i två separata dokument, *XML Schema Part 1: Structures* (http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/) och *XML Schema Part 2: Datatypes* (http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/). Därutöver finns en introducerande skrift *XML Schema Part 0: Primer* (http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/). Observera att den senare inte är någon del av den officiella specifikationen, men en väl så angelägen komplettering för att öka förståelsen av de två första, mycket ”tunga” och kompakta dokumenten.

Specifikationerna finns nerladdningsbara på W3Cs webbplats www.w3c.org. Klicka på ”XML Schema” i vänster kolumn. På webbplatsen finns även referenser till en stor mängd verktyg som på olika sätt säger sig stödja standarden.

De flesta experter tycks vara överens om att XML Schema successivt kommer att ersätta den nu tillgängliga DTD-syntaxen i XML 1.0, även om det kan ta avsevärd tid.

2.2 Dokumentstrukturer generellt

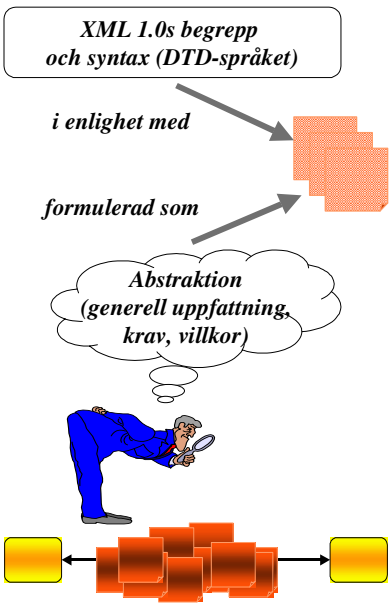
Den som är insatt i XML och vad DTDer står för kan titta på figurerna i detta kapitel och därefter i princip hoppa direkt till kapitel 3.

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 uttryckskraft, kan precisera villkoren mer exakt. DTD och XML Schema ser på XML-dokument med olika ögon. Av den anledningen har de också valt att använda delvis olika termer för att uttrycka de generella egenskaper som ska gälla för varje enskilt XML-dokument av viss typ.

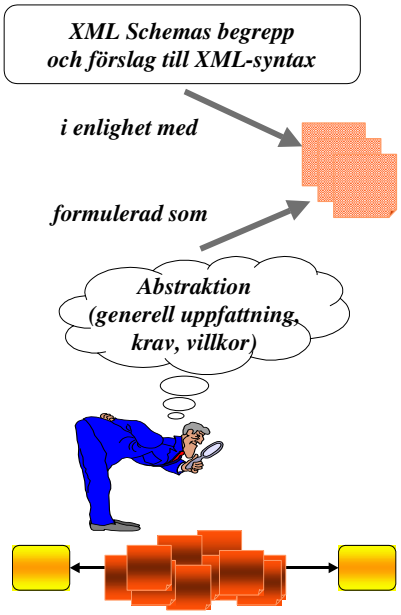
Anta att ett stort antal dokument flödar mellan två (eller flera) parter. Se figur 1.

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).



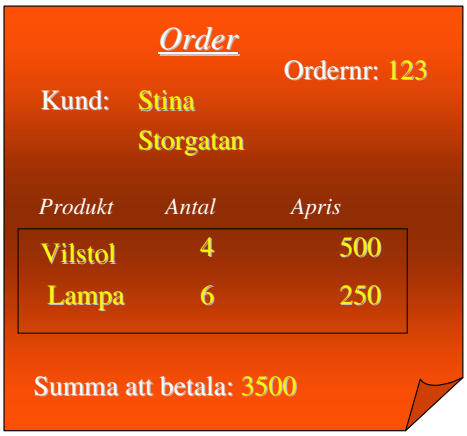
Figur 2a



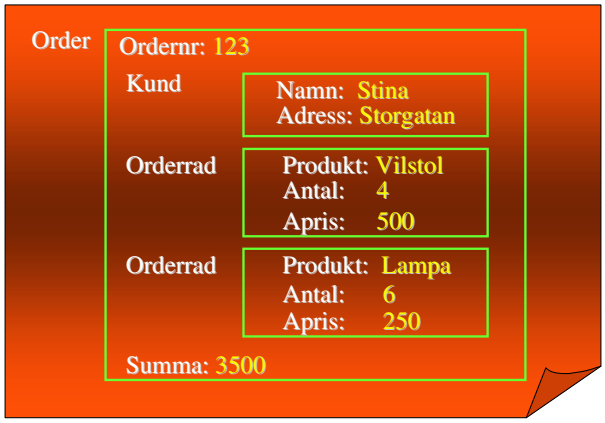
Figur 2b

Anta 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. Anta 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 hierarkiska uppbyggnad eller struktur visas i figur 3b.

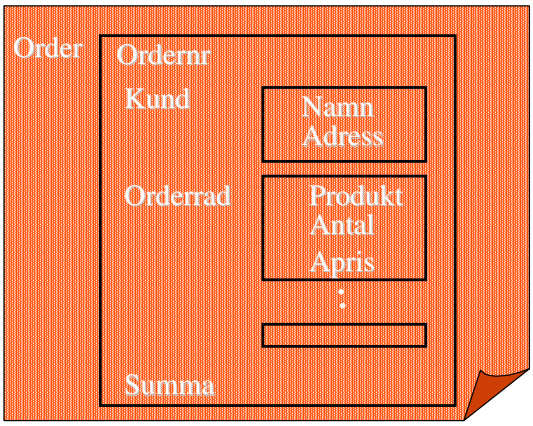


Figur 3a



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 ledord som de samverkande parterna valt att använda för de olika elementen. Figur 4 är en abstraktion av alla orderdokument som visar deras generella uppbyggnad.



Figur 4

Ledorden 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 ledord som ska användas, speciellt om det är parter som känner varandra 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 regler för innehåll på utväxlade dokument. Här kommer dokumenttypsspecifikationen in i bilden.

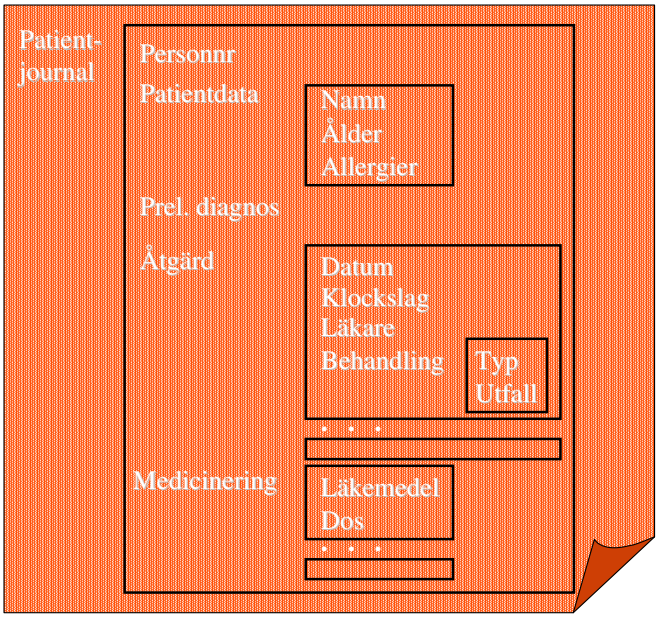
Ta exempelvis dokumenttypen för order. Där gäller det att definiera den struktur och det innehåll som visas i figur 4. Observera att vi här helt bortser från layout på ordern, d v s var i dokumentet de olika delarna ska placeras, vilka teckensnitt som ska användas mm. Layouten är något för mottagaren 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 vidare. 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 termer för att prata om de olika komponenterna i orderdokument behöver vi generella termer 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 termer för nämligen *kund*, *orderrad*, *adress*, mfl. Därigenom kunde parterna prata med varandra på ett generellt plan med hjälp av dessa begrepp, bland annat för att kunna komma överens om dokumenttypsspecifikationen.

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 omfattar ett antal olika 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.

2.3 Dokumentstrukturer beskrivna i XML 1.0

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 uppsättning struktureringsbegrepp, 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 beroende på syfte, konventioner, mm. Istället för ELEMENT kunde vi lika gärna ha använt begreppet KOMPONENT, INGREDIENS, OBJEKT, INFORMATIONTYP, …. 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. Förmodligen specificeras de allra flesta idag existerande tillämpningars dokumenttyper med hjälp av DTD-syntaxen.

3 XML Schema, en översikt

3.1 Inledning

In på arenan träder XML Schema. Den tidigare nämnda arbetsgruppen hade till uppgift att ta fram en begreppsmodell som erbjuder bättre uttryckskraft än vad begreppen i XML 1.0 klarar av för att uttrycka dokumenttyper. Som snart kommer att framgå valde gruppen att uppfatta dokumenttyper på ett i vissa stycken helt annat sätt än XML 1.0.

Intressant att notera är att termen ”dokument” fortfarande används för att referera till en avgränsad informationsmängd för utbyte, trots att dessa numer nästan uteslutande skickas mellan dataprogram utan mänsklig inblandning.

Notera också att man för XML Schema 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 i XML Schema så att ett schema 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.

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

3.2 Grundläggande struktureringsregler

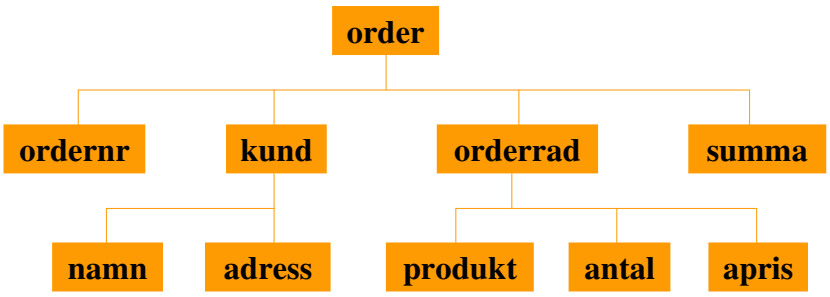
Låt oss nu i ett antal steg diskutera hur order som dokumenttyp skulle definieras om det uttryckes med hjälp av XML Schema-syntax istället för med XML V1.0-syntax. Notera återigen att XML-syntaxen för enskilda dokument är helt oförändrad. Det betyder att XML-dokument ser ut som de alltid har gjort med märkord, element, attribut, hierarkisk uppbyggnad, mm. Till exempel som i figur 5 ovan.

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. En markering av skillnaderna är att en dokumenttyp uttryckt enligt XML Schema-syntaxen kallas för *schema* till skillnad från XML 1.0 som kallar samma sak för DTD (Document Type Definition).

De följande graferna syftar till att introducera den filosofi som XML Schema tillämpar vid formuleringen av dokumenttyper. De olika symbolerna i graferna är författarens egna påhitt. Däremot motsvarar förhoppningsvis dessa symboler syntaktiska konstruktioner i XML

Schema. Tanken är att ett introducerande resonemang baserat på grafiska modeller är lättare att ta till sig än en start direkt i XML Schemas textbaserade syntax.

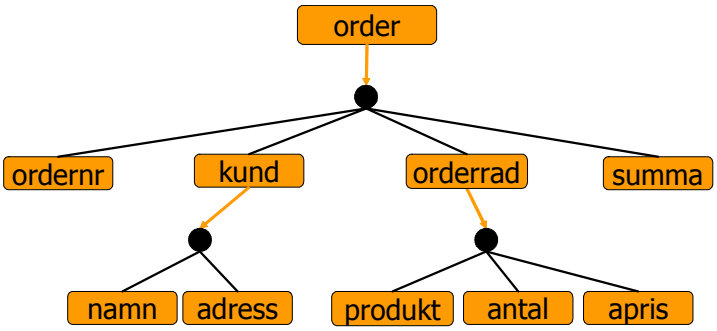
Vi startar med en syntaxneutral hierarkisk bild av orderdokumentets principiella struktur så som XML 1.0 ser på den (figur 8).



Figur 8

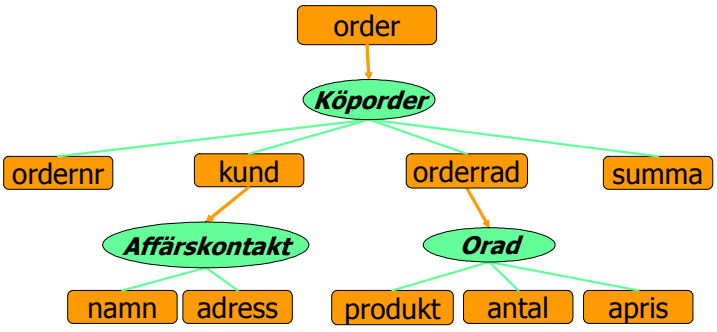
Strukturen visar namnen på alla förekommande element. Av strukturen framgår att elementen *order*, *kund* och *orderrad* vardera representerar eller pekar på en understruktur. Var och en av dessa tre understrukturer beskrivs med sin uppsättning ingående element.

I figur 9 visas samma struktur men på ett något annorlunda sätt.



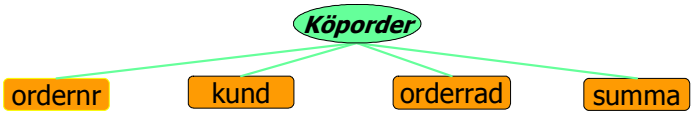
Figur 9

Varje prick i figur 9 svarar mot en parentesomgiven elementgrupp i DTDn i figur 7. I XML Schema kan varje delstruktur ges ett eget namn enligt figur 10. Finessen med det kommer snart att framgå.

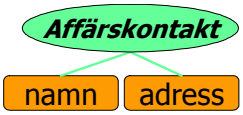


Figur 10

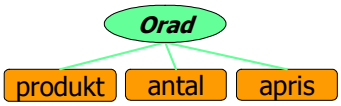
Varje namngiven delstruktur kan upplevas som en egen självständig entitet. Se figur 11 a – c.



Figur 11 a

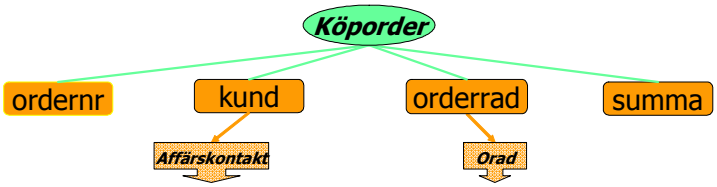


Figur 11 b



Figur 11 c

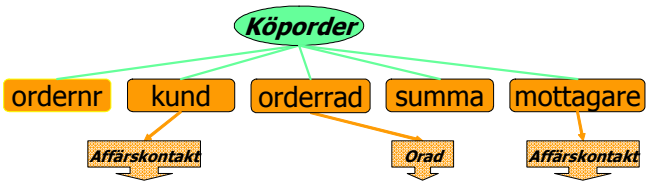
Men därmed håller inte längre ordertypens struktur ihop. Alltså måste det tillföras en möjlighet att referera till en delstruktur från någon annan plats, i vårt exempel från *kund* respektive *orderrad*.



Figur 12

Nu hänger strukturen som helhet entydigt ihop genom referenserna, även om det rent rittekniskt inte fullt ut framgår.

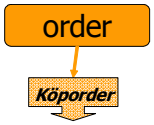
Värt att notera i sammanhanget är att om det att referera till en namngiven delstrukturer från ett ställe är det heller inget som hindrar att också göra det från något annat ställe. Namnet står ju till förfogande som referens. Se figur 13.



Figur 13

Strukturen är inte längre rent hierarkisk. Både den som beställt (*kund*) och den som så småningom ska ta emot varorna (*mottagare*) är affärskontakter som behöver dokumenteras med namn och adress. Det är ju inte säkert att det är samma personer.

Kvarstår att även koppla in märkordet på översta nivå, nämligen *order*. Ordern i sin helhet beskrivs genom strukturen Köporder. Alltså sätts order att referera till Köporder (figur 14).



Figur 14

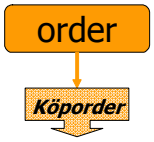
Varje referens talar om vilken typ av information som ska omfattas av det aktuella elementet. *Order* avser *Köporder*, *orderrad* avser *Orad*, o.s.v. Jämför med strukturen och märkordsanvändningen i orderdokumentet i figur 5, ovan.

Hittills har vi endast definierat färdigt de element vars typ av information är en sammansatt delstruktur. Men även för enkla element måste typen anges. Eftersom dessa uttrycks genom text, heltal, symbol, datum, webbadress, … snarare än en delstruktur faller det sig naturligt att referera till just den datatyp som ska gälla för elementet ifråga. Datatypen formulerar villkor för hur värdet får uttryckas.

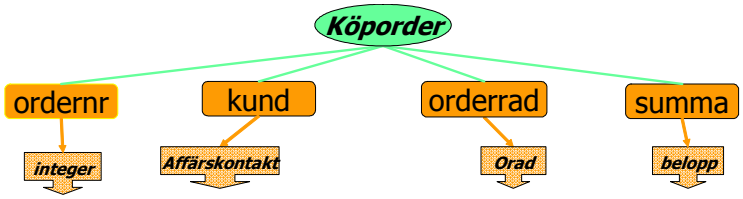
Skillnaden är inte så stor mot de element som refererar till en delstruktur. I det ena fallet specificeras acceptabel datatyp, i det andra fallet acceptabel elementstruktur. I båda fallen fungerar referensen som en slags villkorsspecifikation.

Dokumenttypen för order får i sitt fullständiga skick (så här långt) ett innehåll motsvarande det som grafiskt visas i figur 15.

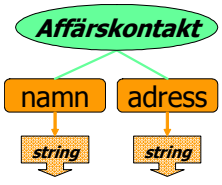
Typer som är sammansatta (har delstruktur) anges som gröna ovaler (15 b – d). Typer som är enkla datatyper anges som gröna rektanglar (15 e).



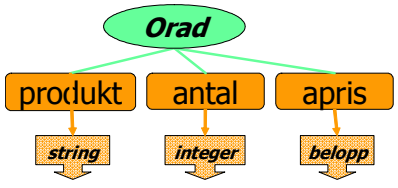
Figur 15 a



Figur 15 b



Figur 15 c



Figur 15 d

Figur 15 e

Observera att detta är ett av flera möjliga sätt att specificera orderstrukturen. Att vi valt att börja med att visa den här principen beror på att den är mest generell och flexibel. Sannolikt

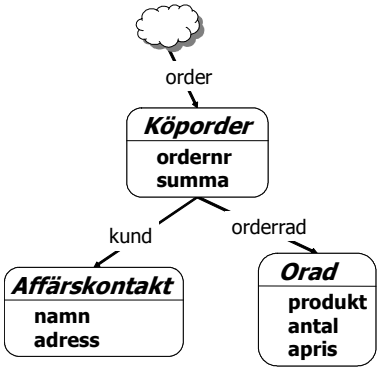
också den mest tillämpade principen. Ytterligare ett sätt kommer att visas senare i detta kapitel.

Vad tillför egentligen Order, kan man undra. Räcker det inte med Köporder? Vore det inte allra bäst om Köporder istället kallas Order för att anpassa till det märkord som används i de enskilda dokumenten?

Figur 13 ovan visar två referenser till samma Affärskontraktsstruktur, en för varje typ av användning av strukturen. I ena fallet gäller det strukturen för kunduppgifter, i den andra strukturen för mottagareuppgifter. Även Köporder kan visa sig bli refererat från flera håll. Varje behov anges med sin typ av referens. I vårt exempel anges behovet av Köporder genom *order*. Andra referenser till samma Köporderstruktur skulle kunna vara *restorder*, *kompletteringsorder*, eller liknande. Det kan ju finnas behov av att samla alla dessa i samma dokument för något syfte.

Alltså finns det behov av att skilja på en struktur som sådan och dess användning i olika sammanhang.

Skillnaden mot vanlig informationsmodellering är vid första påseende inte så stor. Jämför figur 16. Molnet finns med för att just öppna för möjligheten att ange olika syften för den gemensamma strukturen.



Figur 16

Vid närmare granskning finns en hel del att säga om både likheter och olikheter, men den diskussionen får anstå till en separat rapport.

3.3 Strukturen uttryckt i XML Schema

Vi har redan nämnt att XML Schema är baserad på XML, uttrycker sina specifikationer med hjälp av XML-syntax. Alltså måste vi formulera om den grafiska specifikationen i figur 15 till något motsvarande, uttryckt i XML.

Först några ord om tre grundläggande begrepp i syntaxen. Varje beståndsdel i ett dokument kallas *element* (orange rektangel med mjuka hörn i figur 15). Ett element refererar antingen till en enkel eller sammansatt typ (de gröna rektanglarna respektive ovalerna i figur 15). En enkel typ kallas i XML Schema för ”*simple Type*”. En sammansatt typ kallas för ”*complex Type*”. Delarna i en sammansatt typ består av element precis som delarna av ett fullt dokument består av element.

3.3.1 Specifikation av complex Types

Låt oss börja med Affärskontakt. Den är en sammansatt typ bestående av två element, *namn* och *adress*. Elementet *namn* anser vi ska uttryckas med hjälp av den enkla typen *string*. Detta framgår av den nedåtriktade pilen under rektangeln *namn* i figur 15 c. Samma gäller adress. *String* finns som en fördefinierad datatyp i XML Schema tillsammans med 30-40 stycken andra fördefinierade datatyper.

Formuleringen blir med hjälp av XML Schemasyntaxen enligt figur 17. *Element* används som märkord för elementtyper samt kompletteras med värden för attributen *name* och *type*. Varje värde omges av citationstecken. Attributet *name* anger det namn vi vill använda som märkord i enskilda dokument för de elementet som svarar mot elementtypen ifråga. Vid *type* anges därutöver den datatyp som ska gälla för elementtypens värden i enskilda dokument. Allt uttryckt i vanlig XML-syntax.

```
<element name="namn"     type="string"/>
<element name="adress"   type="string"/>
```

Figur 17

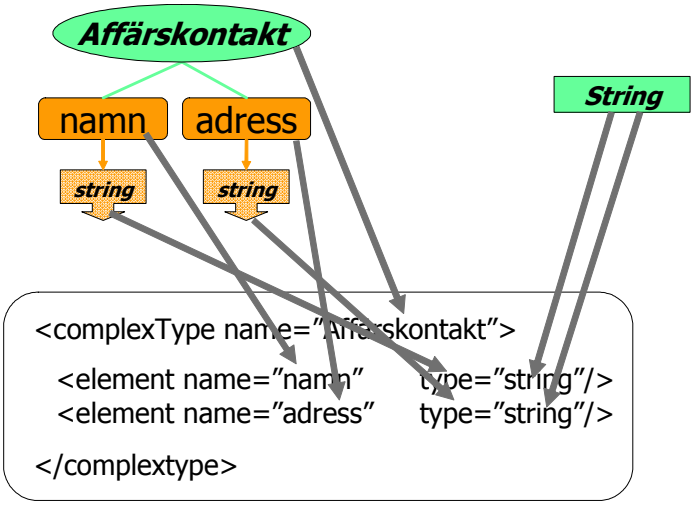
I figur 5 ovan återfinns bland annat ”<namn> Stina </namn>” och ”<adress> Storgatan </adress>”. Alltså helt korrekt formulerade i enlighet med schemat så här långt. Både ”Stina” och ”Storgatan” är ju korrekta string-värden.

Affärskontakt är en complex Type bestående av de två elementen i figur 17. Affärskontakt deklareras enligt figur 18.

```
<complexType name="Affärskontakt">
  <element name="namn"     type="string"/>
  <element name="adress"   type="string"/>
</complextype>
```

Figur 18

Figur 19 sammanför figur 15 c med hur och var motsvarande konstruktioner uttrycks enligt XML Schemasyntaxen.



Figur 19

Vi kompletterar på motsvarande sätt med Orad i figur 20.

```
<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>
```

Figur 20

Egentligen är det lika enkelt att även inkludera Köporder. Den enda skillnaden är att två av de fyra ingående elementtyperna refererar till delstrukturer och inte till enkla datatyper. Men det är lätt ordnat. Type sätts i dessa fall helt enkelt att referera till aktuell complex Type på samma sätt som de övriga refererar till simple Types.

```
<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>
```

Figur 21

3.3.2 Specifikation av simple Type

Integer och *string* är fördefinierade datatyper. Det är däremot inte *belopp*. Man kan ju inte begära att XML Schema ska ha tänkt på alla möjliga behov av datatyper som kan finnas överallt och för alla behov. Om vi ansåg att apris och summa kunde uttryckas som heltal skulle vi valt att deklarera type=”integer” för dem. Eftersom vi i vår tillämpning har som krav att apris och summa lägst får vara 10 och högst 100000, vill vi att denna regel ska gälla och kontrolleras när dokumenten så småningom kommer strömmande. Därför behövs en speciell datatyp som formulerar regeln ifråga.

Lyckligtvis erbjuder XML Schema möjlighet att specificera egna enkla datatyper efter egna behov. Vår speciella simple Type har vi valt att kalla ”belopp”. Specifikationen framgår av figur 22. Där anges att vi önskar formulera en restriktion av de regler som gäller för integer. Vilka restriktioner som ska gälla anges genom minInclusive och maxInclusive.

```
<simpleType name=”belopp">
   <restriction base="integer">
       <minInclusive value="10"/>
       <maxInclusive value=”100000"/>
   </restriction>
</simpleType>
```

Figur 22

Restriction, minInclusive och maxInclusive är fördefinierade elementtyper i XML Schema. Utöver dem finns en rikhaltig uppsättning andra, allt i avsikt att erbjuda en rik specificeringskapacitet för egendefinierade datatyper.

3.3.3 Avslutande komplettering

Återstår att inkludera *order* (figur 15 a) i specifikationen. Order är ett element bestående av en delstruktur av typ Köporder. Specifikationen sker på principiellt samma sätt som elementet kund, d.v.s. här med referens till complex Type Köporder. Avslutningsvis behöver vi tala om att denna kompletta specifikation utgör ett schema genom att omge den med märkordet <schema>. Figur 23 visar resultatet.

```
<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">
        <restriction base="integer">
              <minInclusive value="10"/>
              <maxInclusive value=”100000"/>
        </restriction>
    </simpleType>
</schema>
```

Figur 23

3.3.4 Alternativ specifikation

Hittills har varje delstruktur formulerats under ett självständigt namn. Detta öppnar för möjlighet att med hjälp av namnet referera till samma delstruktur från flera ställen i schemat. Finns inte det behovet kan respektive delstruktur alternativt bakas in på sin rätta plats i helheten och då utan något explicit namn. Vi är tillbaka till en specifikation motsvarande trädet i figur 9 ovan.

```
<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 name=”Orad”>
                    <element name=”produkt”       type=”string”/>
                    <element name=”antal”         type=”integer”/>
                    <element name=”apris”>
                        <simpleType >
                            <restriction base="integer">
                                <minInclusive value="10"/>
                                <maxInclusive value=”100000"/>
                            </restriction>
                        </simpleType>
                    </element>
                </complexType>
            <element name=”summa”>
                <simpleType >
                    <restriction base="integer">
                        <minInclusive value="10"/>
                        <maxInclusive value=”100000"/>
                    </restriction>
                </simpleType>
            </element>

        </complextype>

    </element>
</schema>
```

Figur 24

Priset för denna smidighet är sämre flexibilitet för eventuella framtida förändringsbehov. I vårt fall innebär det också att beloppsspecifikationen måste göras på två ställen, se de båda simpleType-deklarationerna.

De fortsatta schemaexemplen har vi valt att bygga på schemat i figur 23.

Både figur 23 och figur 24 är i princip kompletta specifikationer av ett schema (om vi bortser från diverse formalia som inte direkt berör den aktuella strukturen). Men det finns ytterligare saker av tillämpningsorienterad karaktär som vi önskar komplettera schemat med. Låt oss gå vidare.

3.4 Några villkorsspecifikationer

Hittills har vi endast berört hur dokumentstrukturer i stort specificeras. I många fall behöver, förutom strukturen, även kompletterande villkor kunna anges för att mer i detalj reglera hur enskilda dokument får och inte får vara uppbyggda.

3.4.1 Ordningsföljd mellan element

För angivna element under en complex Type kan anges om de i ett enskilt dokument måste förekomma i samma ordning som de specificerats i dokumenttypen. Detta sker genom att omgärda gruppen elementdeklarationer med <sequence>. För elementen i Affärskontakt är detta ett rimligt antagande.

```
<complexType name="Affärskontakt">
   <sequence>
      <element name="namn"      type="string"/>
      <element name="adress"    type="string"/>
   </sequence>
</complextype>
```

Figur 25

Om endast en av de uppräknade elementtyperna får förekomma, valfritt vilket, anges detta med <choice> istället. <all> används om elementtyperna må anges i valfri ordning, dock med restriktionen att ett element i varje elementtyp får förekomma högst en gång.

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

3.4.2 Villkor för viss elementtyp

Element av elementtypen namn inom Affärskontakt måste alltid anges där Affärskontakt förekommer i ett enskilt dokument. Däremot får adress i den aktuella tillämpningen utelämnas om man så finner lämpligt. Denna frihet för adress att förekomma 0 eller 1 gång måste kunna specificeras. Anger man inga villkor i det avseendet gäller att elementet ska förekomma en och endast en gång (1:1). Som det nu står i figur 23 ovan gäller alltså att både namn och adress måste finnas med.

XML 1.0-syntaxen anger rätten att utelämna ett värde med ett frågetecken efter elementtypsnamnet. Se i figur 7 ovan hur adress specificerades. XML Schema erbjuder betydligt rikare villkorsspecifikationer än XML 1.0s frågetecken (0:1), asterisk (0:M) och plustecken (1:M).

Märkordet element kan i XML Schema kompletteras med olika attribut som vart och ett reglerar något villkor. Två av dessa är *minOccurs* och *maxOccurs*. Elementdeklarationen för adress i figur 23 ändras till

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

Anges inte någonting gäller alltså alltid en uppgift, d v s minOccurs="1" och maxOccurs="1". Notera att maxOccurs för adress 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. Elementtypen orderrad under Köporder omdeklareras enligt

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

Eftersom minOccurs utelämnats gäller minst 1 orderrad.

Anta 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 26).

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

Figur 26

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 ett visst specifikt värde alltid ska gälla för elementet. Attributet *default* används för att deklarera att om inget annat värde angivits så ska defaultvärdet användas.

Tillbaka till tillämpningen. Affären säljer enbart sin egentillverkade fotogenlampa. Alltså kan endast order med produktnamnet ”fotogenlampa” accepteras (lite långsökt men ändå). Man tillämpar också den lite luriga regeln som säger att om antalsuppgiften skulle utebli så betyder det två exemplar. Specifikationerna blir som följer:

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

Ökas produktsortimentet till att även innefatta gungstol, cykel och kratta (detta är en diverseaffär) räcker inte fixed längre. Vi får specificera en ny simple Type, exempelvis kallad *produktnamn*, under vilken vi begränsar den fördefinierade datatypen string till att endast acceptera de fyra produktnamnen.

```
<simpleType name=”produktnamn">
   <restriction base=”string">
        <enumeration value=”fotogenlampa"/>
         <enumeration value=”gungstol"/>
        <enumeration value=”cykel"/>
        <enumeration value=”kratta"/>
   </restriction>
</simpleType>
```

Figur 27

Den som tycker det nu räcker med kunskaper om XML Schema må stanna här. Övriga får information om lite fler konstruktioner i nästa kapitel.

4 XML Schema, lite mer

4.1 Attributdeklarationer

Köpordern innehåller inga attribut någonstans men i likhet med DTD-språket kan attributdeklarationer också hanteras i XML Schema. Vilket ställer sig naturligt eftersom de kan förekomma i enskilda XML-definierade dokument. Attribut är med andra ord ett begrepp med i stort sett likalydande innebörd i båda ansatserna även om de hanteras i olika språksyntaxer när det gäller typspecifikationer.

Anta att någon valt att formulera vår kända order enligt figur 28 nedan istället för den tidigare versionen enligt figur 5 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 28

Båda är helt korrekta XML-dokument men i figur 28 anges de enkla värdena (”löven”) som attribut istället för som element. Attribut måste alltid uttryckas med ett värde, d v s alltid som en simple Type. Schemadeklarationen måste anpassas därefter.

Figur 29 visar specifikationen av det mot de nya förutsättningarna anpassade schemat.

```
<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">
        <restriction base="integer">
              <minInclusive value="10"/>
              <maxInclusive value=”100000"/>
        </restriction>
   </simpleType>
</schema>
```

Figur 29

Skillnaden i schemat är egentligen bara att <element> ersatts med <attribute> och att attributtypsdeklarationerna placerats efter elementtypsdeklarationerna inom samma complex Type.

Den som så önskar kan välja att gruppera ett antal attribut under ett gemensamt gruppnamn. I figur 30 är *kunddata* en *attributeGroup* sammanförande namn och adress.

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

Figur 30

Vad är då vitsen med attributeGroup? Dels kan uppdelningen upplevas vara mer lättläst, dels – och betydligt viktigare – är gruppen självständigt deklarerad och därmed refererbar från flera ställen med 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.

I princip kan den som definierar dokumenttyper fritt välja att deklarera 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, som ju finns i den grundläggande XML-syntaxen, är snarare en belastning än en tillgång för envärdiga uppgifter eftersom den inte ger definieraren någon vägledning 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 element-deklarationen eftersom den ger större flexibilitet, bland annat när det gäller antal uppgifter per elementtyp. Till attribute-deklaration hänvisas lämpligen uppgifter som berör informationselementet snarare än den värld vi vill hantera information om. Ett exempel på ett sådant attribut kan vara uppgift om informationselementets trovärdighet, uppgiftslämnare, tidpunkt för registrering och andra liknande uppgifter som vanligtvis kallas metadata.

4.2 Kompletterande namespace-deklarationer

För att bli en fullödig definition behöver ytterligare att par deklarationer tillföras schemat. Namespace-deklarationer är ett sådant komplement. 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. Se figur 31.

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

Figur 31

Elementet ”order” har kompletterats med attributdeklarationen

xmlns=”http://www.postordersammanslutningen.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* definierat i och för användning inom *Orderhantering*. Deklarationen underlättar entydig tolkning av märkorden i dokumentet.

Tillbaka till schemat i figur 23. 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/2001/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/2001/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 deklareras i schemat, t.ex. order, Affärskontakt, namn, ….. Detta namespace är naturligen samma som det vi just deklarerat på översta elementnivå i orderdokument. Se figur 31.

```
<schema xmlns=”http://www.w3.org/2001/XMLSchema”
        targetNamespace=”http://www.postordersammanslutningen.org/Orderhantering”>
    <element name=”order”   type=”Köporder”/>
    <complexType name=”Köporder”>
      …
</schema>
```

Figur 31

Användningen av namespace-deklarationer 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-standarden och till W3C-standarden ”Namespaces in XML 1.1” (http://www.w3.org/TR/2004/REC-xml-names11-20040204).

4.3 Extension och restriction

4.3.1 Extension

Nu närmar vi oss något mer komplicerat. I datamodeller har det närmast sin likhet i specialiseringssamband eller arvsdeklarationer. Anta att vissa av våra kunder inte bara klassas som Affärskontakt utan även som så kallad 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. Elementen namn och adress ”ärver” VIPkund från Affärskontakt genom att ange att VIPkund 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 32 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=”Affärskontakt”>
        <element name=”rabattsats”  type=”integer”/>
    </extension>
</complexType>
```

Figur 32

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 (Affärskontakt) eller en VIPkund.

Är alternativet månne 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. Den kontrollerande rutinen hos mottagaren (parser) kan nu anpassa kontrollen efter den angivna typen. Jämför med objektorienteringens snarlika polymorphism-begrepp.

Sofia är faktiskt VIPkund till skillnad från Stina i figur 5. Stinas orderdokument får därför utseende enligt figur 33 där rabattsats 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 33

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.

4.3.2 Restriction

Extension uttrycker kompletterande alternativ/villkor för giltigt innehåll, nämligen möjlighet/krav på ytterligare element och/eller attribut som 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 därutöver ställa hårdare krav på ett eller flera av elementen i form av antal tillåtna förekomster, mm. På så vis sker också en avgränsning till den delmängd av den ursprungliga populationen som uppfyller kraven. Kraven deklareras i en separat namngiven complex Type med referens till den complex Type man utgår ifrån.

Ett exempel: Vår aktuella verksamhet använder sig av hävd av begreppet Storорder för att indikera sådana order som omfattar minst 11 orderrader. Storорder innehåller för övrigt exakt samma element som Köporder. Deklarationen blir som i figur 34. Detta är i praktiken också en form av specialisering. En viss Storорder är ju samtidigt en Köporder.

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

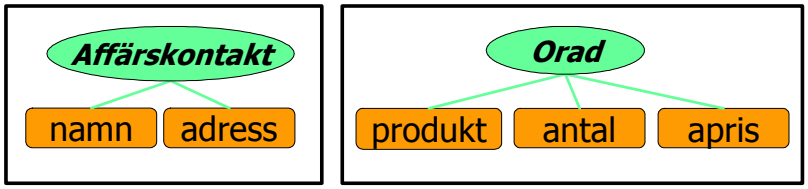
Figur 34

Restriction-faciliteten kommer för övrigt också väl till pass när vi önskar definiera egna simple Types baserade på redan existerade.

4.4 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 helt separata delar (underdokumenttyper) om vi så önskar.



Figur 35

Varje del utgör ett subschema, d v s innehåller valda delar av det fulla schemat definierat i en separat fil refererbar genom en egen webbadress.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.postordersammanslutningen.org/Orderhantering">
        <complexType name="Affärskontakt">
            <element name="namn"      type="string"/>
            <element name="adress"    type="string"/>
        </complexType>
</schema>
```

Figur 36

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

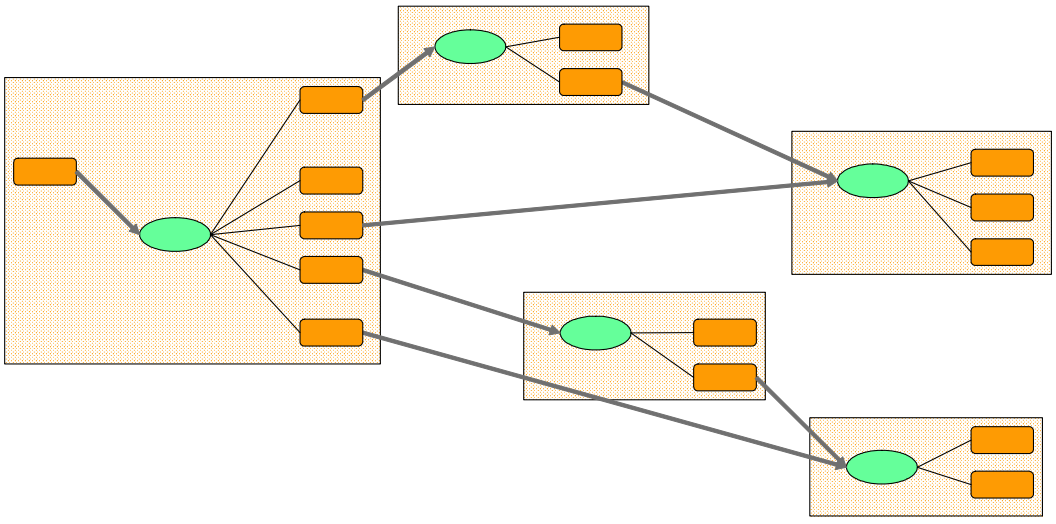
Figur 37

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/2001/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 38

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 39 visar fyra olika subscheman som alla i detta fall direkt eller indirekt utnyttjas för att definiera schemat längst till vänster. Självfallet kan även andra scheman begära include på någon lämplig delmängd av dessa. Fördelen med XML Schemas stringenta uppdelning av struktur (grön) och användning av en struktur (orange) framstår återigen tydligt. Dock gäller den begränsande restriktionen att samtliga dessa scheman måste referera till samma target namespace, d v s representera samma ämnesområde, syfte e.dyl.



Figur 39

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 38 ger knappast include-instruktionen någon vidare vägledning om vad som döljer sig bakom webbadressen. Såvida inte något stödjande dokumentgenereringsverktyg används förstås.

5 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 vid 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? Tveksamt. Här följer några synpunkter. De är medvetet något spetsigt formulerade för att förhoppningsvis inbjuda till egna funderingar, ståndpunkter och värderingar.

XML Schema består av två normativa delar, Part 1: Structures som omfattar ca 190 sidor och Part 2: Datatypes som omfattar ca 145 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 tolkat 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 möjligtvis 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 bygga på 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? Eller beror det helt enkelt på att man har ett fokus på dokument och dess struktur snarare än på en kombination av dokumentstrukturering och begreppsmodellering?

Intressant att uppmärksamma i sammanhanget är XML Schemas två olika ”personligheter”. Den ena är användartillvänd och lockande. Att specificera ett schema i en grundstruktur är både roligt och relativt enkelt. Att därifrån tillföra alla erforderliga kompletteringar och för

varje komplettering vara klar över syfte och konsekvenser är att möta den andra betydligt mer svårfångade personligheten.

Om lösningen i sig är komplex, vilken den kanske måste vara, är en ”kompenserande” välgenomtänkt pedagogisk specifikation inte bara efterlängtad 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. Arbete 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 trots allt de normativa delarna som entydigt och fullständigt ska kunna tolkas med rimlig ansträngning.

Mot detta kan förstås hävdas att de flesta av oss aldrig behöver se scheman uttryckta i XML Schemasyntax, än mindre behöva förstå den. Det finns ju verktyg att använda som döljer komplexiteten. Visst, verktyg underlättar, men även de måste på olika sätt exponera de olika uttrycksvarianter som språket erbjuder. Även ett grafiskt språk har en syntax en verktygsanvändare måste förstå innebörden av. Till detta kommer förståelsen för syftet med ett schema. Lägg till detta känslan för orsak, verkan, konsekvenser med de formuleringar som görs. Visst kan de mer intrikata detaljerna överlåtas till mer flyhänta schemaspecificerare men ska en produktiv dialog kunna föras mellan denne och kravställarna måste den grundas på en ömsesidig förståelse kring det som diskuteras.

Vem är förresten redo för användning av XML Schema? Antagligen Part 2 – den del som varit mest efterfrågad. Men Part 1? Förmodligen ganska få just nu. Hur många använder sig av alla de verktyg som exponeras på W3Cs webbplats? 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. Om intresset då är svalt för XML Schema borde det 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 tog lång tid att ta fram, betydligt längre tid än någon initialt räknade med. Berodde detta på partintressens kamp om herraväldet eller på en seriös bearbetning av ett

komplicerat problemområde? Är detta möjligtvis ytterligare ett orostecken när det gäller den kommande uppslutningen bakom XML Schema?

Oops – kanske blev det 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 de allt tydligare tecknen slår in, XML Schema kommer att bli en central ingrediens i nästa generations XML-tillämpningar.