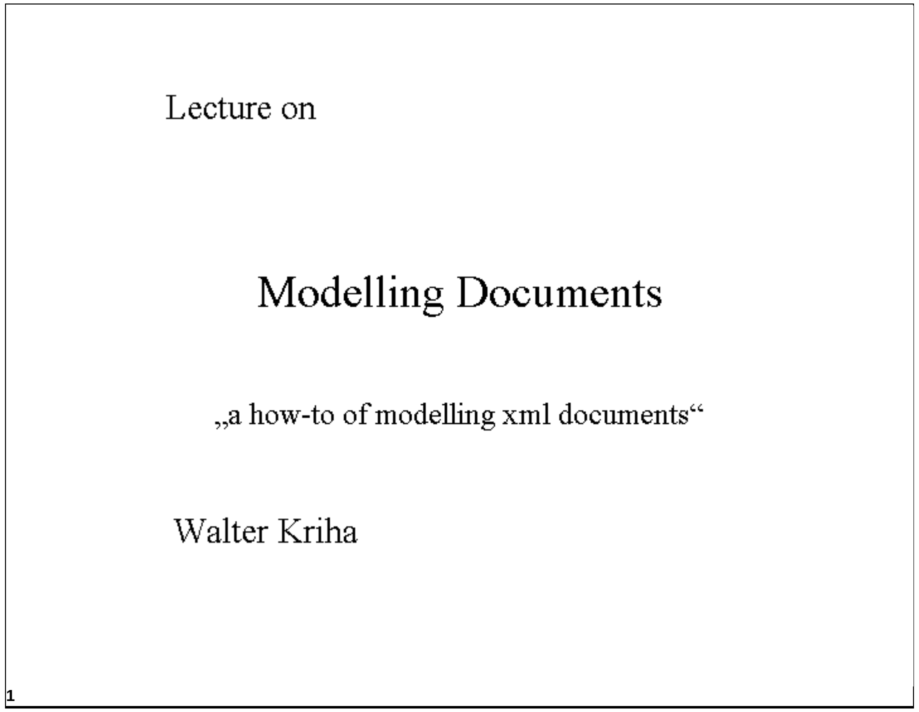
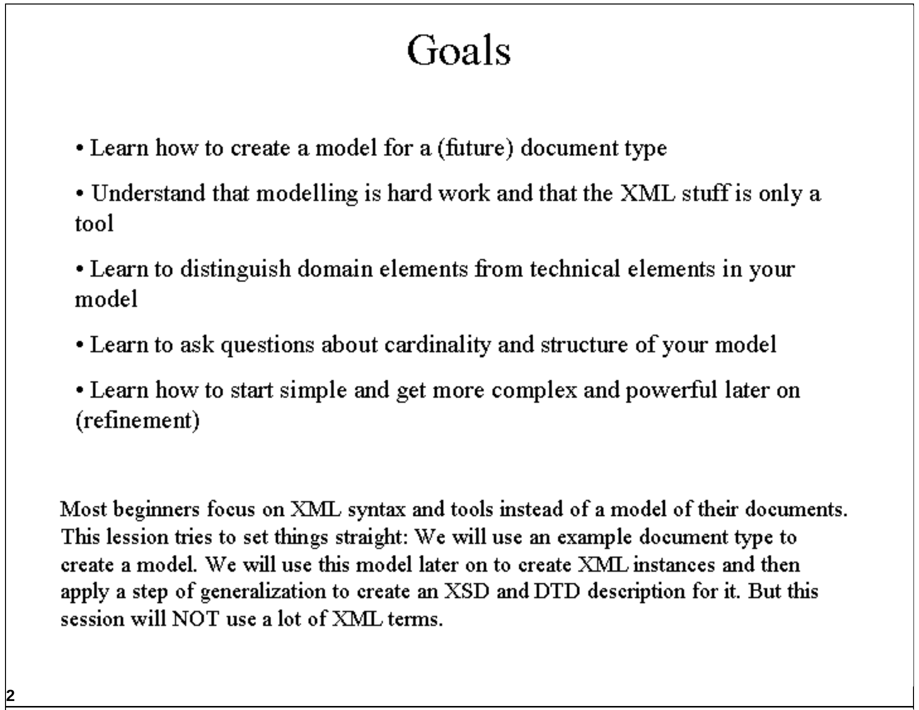
Document Construction


Lecture on

Modelling Documents

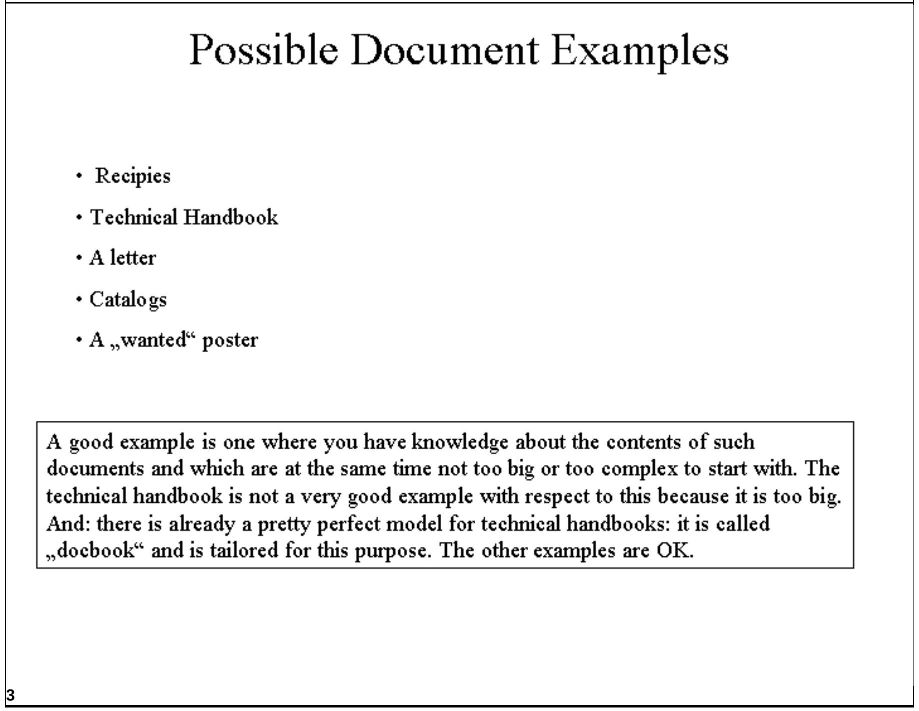
„a how-to of modelling xml documents“

Walter Kriha


Goals

- Learn how to create a model for a (future) document type
- Understand that modelling is hard work and that the XML stuff is only a tool
- Learn to distinguish domain elements from technical elements in your model
- Learn to ask questions about cardinality and structure of your model
- Learn how to start simple and get more complex and powerful later on (refinement)

Most beginners focus on XML syntax and tools instead of a model of their documents. This lession tries to set things straight: We will use an example document type to create a model. We will use this model later on to create XML instances and then apply a step of generalization to create an XSD and DTD description for it. But this session will NOT use a lot of XML terms.


Possible Document Examples

- Recipies
- Technical Handbook
- A letter
- Catalogs
- A „wanted“ poster

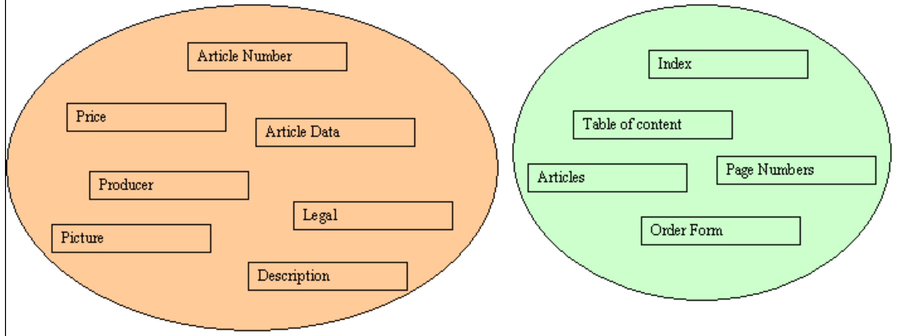
A good example is one where you have knowledge about the contents of such documents and which are at the same time not too big or too complex to start with. The technical handbook is not a very good example with respect to this because it is too big. And: there is already a pretty perfect model for technical handbooks: it is called „docbook“ and is tailored for this purpose. The other examples are OK.

Step one: Brainstorming What makes a catalog?

- Article number
- Description
- Price
- Picture(s)
- Index
- Legal
- Order Forms
- Article data (technical data)
- Producer
- Page numbers
- Table of content

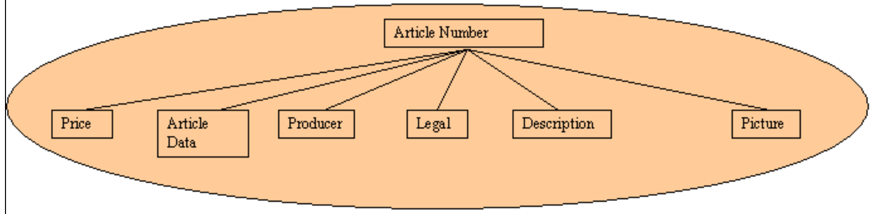
In a first step we collect things that appear in a catalog. We just write down what we know in list form without getting into the details yet.

Step two: group things



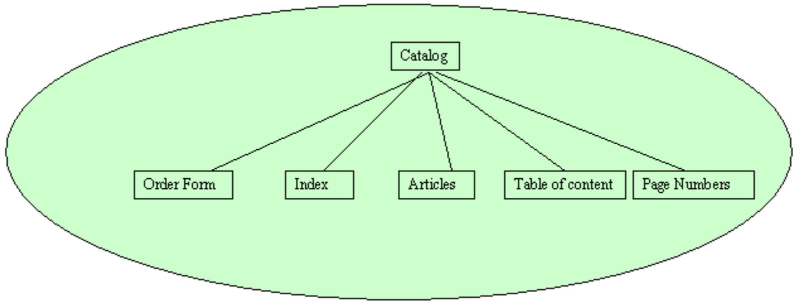
It looks like our things can be grouped in 2 main categories: one has to do with the articles themselves, like a row in a database with article number as the primary key. The other one has things which belong to the catalog as well. We don't have a top element yet for the right side. We can chose „catalog". The left side groups around „article number" or „article" quite naturally.

Step three: build trees (1)



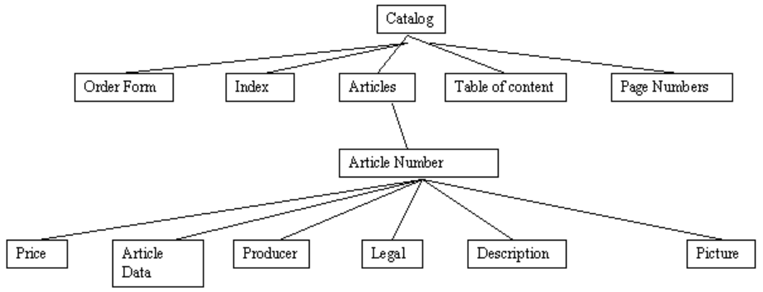
We can form a flat tree with „article number" as the root element. We do the same for our other catalog elements on the next slide.

Step three: build trees (2)



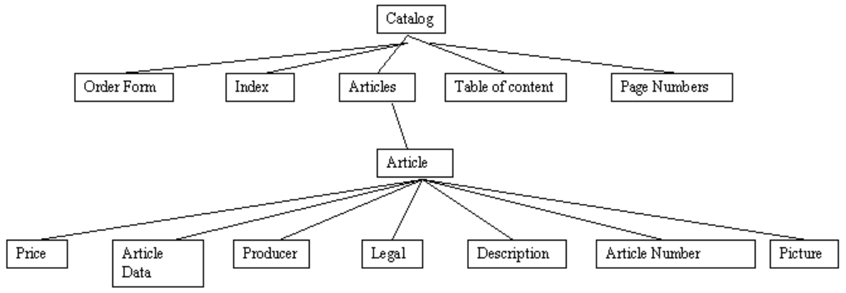
While we build the catalog tree we notice a couple of things: first, we want our articles show up in the catalog but we do not have an „article" thing yet. All we have is „article number". We can introduce „article" easily. Second: Some of our things look a bit dubious: Shouldn't page numbers and table of content be generated automatically? How do we create an index automatically? What information do we need to do this? We will come back to those problems later on.

Step four: combine trees (1)



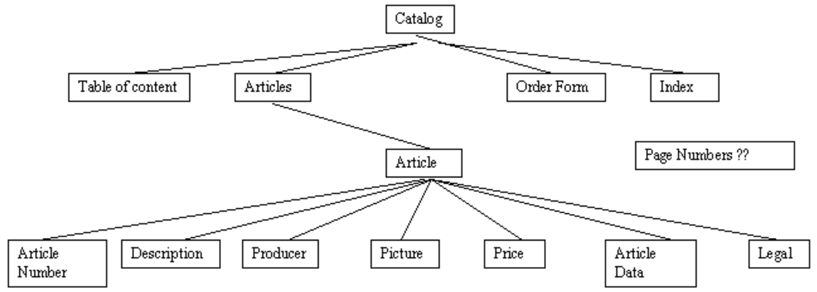
We recognize that we have a little problem with „articles“ and „article number“ which we solve by pushing „article number“ back on the same level as „price“ etc. and introduce „article“ as the new parent element. We also notice that we need to distinguish between many things (articles) and only one thing (article)

Step four: combine trees (2)



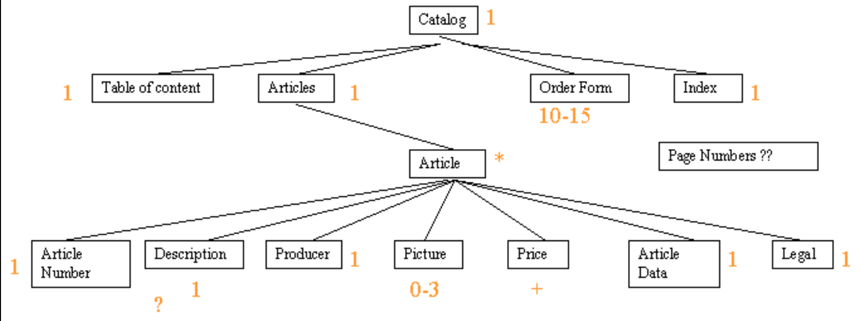
Now we can merge the two trees of our catalog easily. But it still does not look right. Istn‘t there some kind of ORDER in a catalog? Meaning TOC comes first, index last etc.? We need to get some order into our catalog model.

Step five: Create order



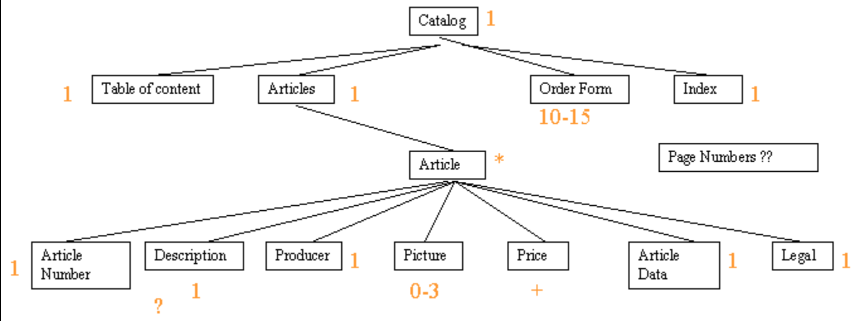
Again we discover some problems: Page numbers need to go everywhere in our catalog. And we want them to be generated. Let‘s exclude them for now. Order seems to be more important in the catalog main tree than in the article sub-tree. But it looks ok to start with article number there and end with the legal stuff.

Step six: Create cardinalities



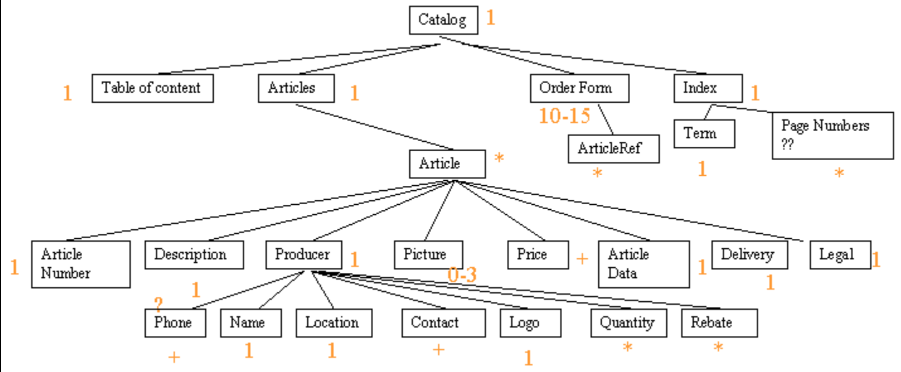
How many of each thing do we have in the catalog? We decide to use the following simple notation: 1 for one and only one, + for at least one, * for 0 or many, ? for 0 or 1 and a number or range for a specific number or range of numbers (15, 2-12 etc.). We notice that by doing this we also distinguish between MANDATORY and OPTIONAL things in our catalog. „Picture“ e.g. is an optional thing because there can be 0 pictures for an article. And every article now must have a description.

Step seven: refine structure (1)



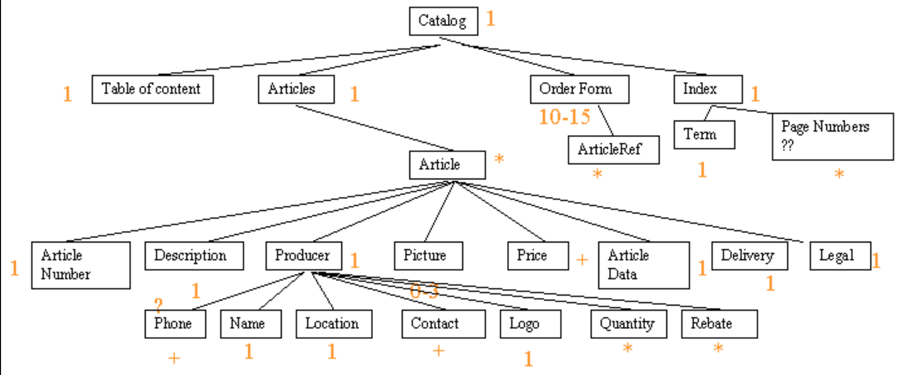
Some of our things in the catalog look a little coarse: „Producer“ e.g. could have some more internal structure like address, contact person, telephone/fax, location etc. And what about „Order Form“. There is certainly more structure in this but we would probably repeat elements from the main tree there. We decide to skip order form for now and treat it as a separate tree whose contents „refers“ to elements in our main tree. BTW: we cannot „refer“ to anyting yet but we write it down as a requirement.

Step seven: refine structure (2)



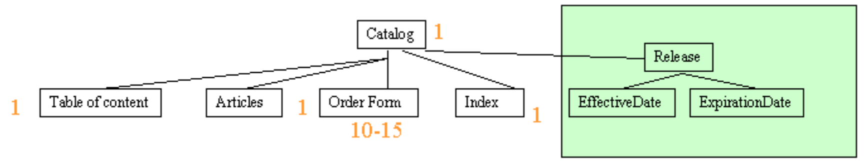
Things are getting more complex now. We still feel that e.g. picture would need more structure (what if somebody cannot see?). We still miss a delivery element saying how long you have to wait for an article to be shipped. „Picture“ is troublesome for other reasons as well: we don‘t want the pictures in our database, we only want REFERENCES to pictures there (We‘ve also heard that XML does not embed pictures and instead uses references as well.) How do we reference pictures? And last but not least: how detailed should we/could we be in our model?

Step seven: refine structure (3)



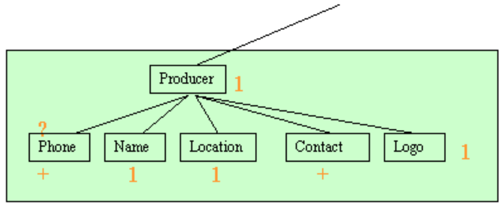
Things are getting more complex now. We still feel that e.g. picture would need more structure (what if somebody cannot see?). We still miss a delivery element saying how long you have to wait for an article to be shipped. „Picture“ is troublesome for other reasons as well: we don‘t want the pictures in our database, we only want REFERENCES to pictures there (We‘ve also heard that XML does not embed pictures and instead uses references as well.) How do we reference pictures? And last but not least: how detailed should we/could we be in our model?

Step eight: Define Housekeeping Info (1)



Some meta-information is needed to distinguish different versions of our catalog. A change log element could be added as well. Possibly also the names of our authors with an unique ID each.

Step eight: Identify Re-usable Things (2)



„Producer“ is a often required type of information. We should make our schema so modular that we can re-use parts of it in other contexts as well. „Address“ may be part of Producer but could be used standalone as well.

16

How much structure and detail?

• The more structure the better you can later on do post-processing of the content.

• More structure also leaves less opportunity for misunderstandings.

• More structure is also a guideline for authors

• The more structure you define the more will authors feel „forced“ into following it. Authors may feel like filling out forms instead of being creative.

• More structure slows down the process of creating an instance

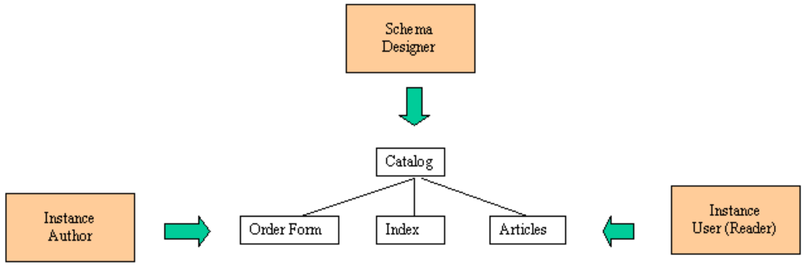
• More structure can lead to a point where not all necessary information is available and authors can no longer create complete instances of your model.

• More structure forces you to define more elements as optional because not every instance will need all elements.

There is clearly a trade-off between structure and free-form. You will need to think about how authors will create instances of your model. Don‘t make this too cumbersome or authors will no longer accept your model. But also force them to fill in those elements which are necessary for a complete document instance and which you need later on for post-processing. Sit down and create some instances of your model so you feel how it works.

17

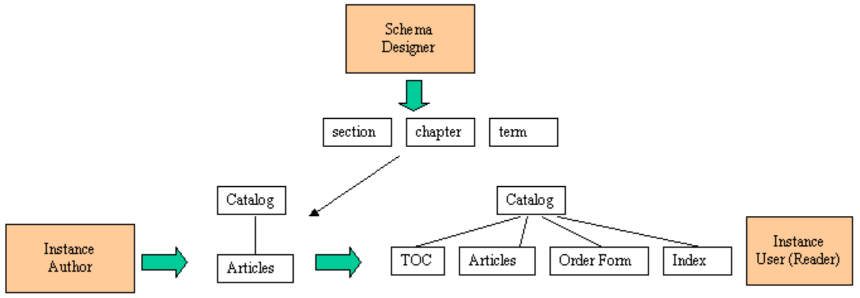
Three views on your model



The schema designer creates a model of a document by defining structure and types of elements. Instance authors create instances which comply with the schema and fill in content. End-users get a rendering of those instances (e.g. a complete catalog) in different media types (online, printed etc.) The art of document design lies in understanding both the authoring side (creation) as well as the user side (viewing) of the instances. A Schema designer needs to make a model so that it allows many different uses (renderings) but still be understandable for the authors. Some elements will need attributes used in post-processing only. There is NO modelling without respecting authors AND users (post-processing) needs.

18

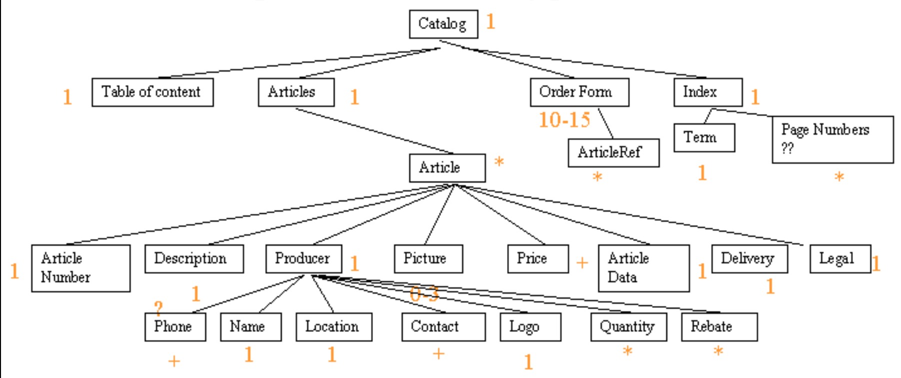
Generated Content



Authoring of our catalog basically means filling in instances of article elements. This is what the authors are interested in. The schema designer added other elements to the model which serve as anchors for the post-processing step. Table-of-content, order form and index are parts of a catalog which will be generated automaticall IFF the proper structure exists in our catalog model. A TOC without chapters/sections with titles is hard to generate. An index without <term> marks in the articles cannot be generated automatically.

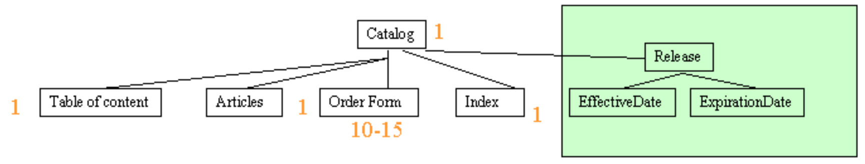
19

Step nine: define types (1)



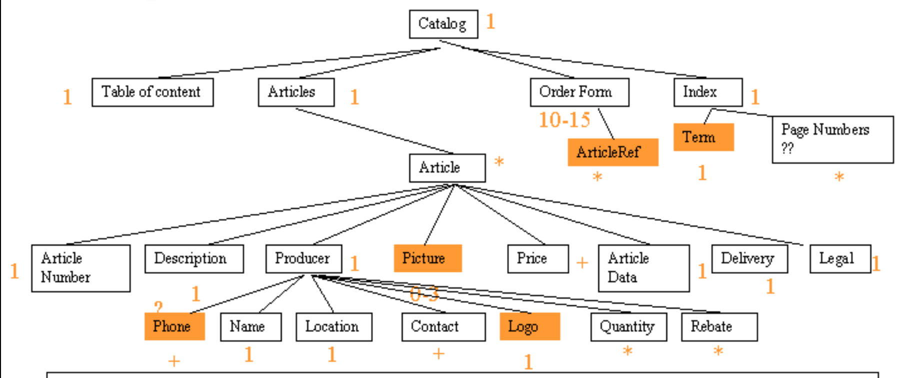
Things are getting more complex now. We still feel that e.g. picture would need more structure (what if somebody cannot see?). We still miss a delivery element saying how long you have to wait for an article to be shipped. „Picture" is troublesome for other reasons as well: we don't want the pictures in our database, we only want REFERENCES to pictures there (We've also heard that XML does not embed pictures and instead uses references as well.) How do we reference pictures? And last but not least: how detailed should we/could we be in our model?

Step nine: what becomes an attribute (2)



EffectiveDate and ExpirationDate are usually atomic values. They won't be extended with further child elements and can therefore easily become attributes of Catalog. This is a step toward modelling the catalog in XML. Don't worry about elements vs. attributes too early: leave room for everything to be extended during the modelling process and only then determine atomic values (attributes)

Step ten: define technical elements (1)



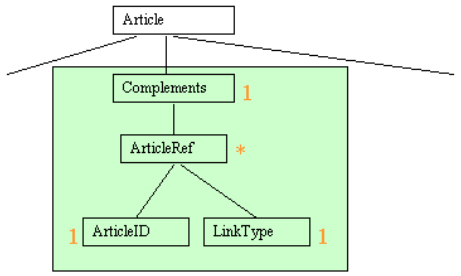
Some things in our model seem to be of more general nature and have little to do with a catalog itself – these things are needed in other document types as well. Examples are: References to pictures, term definitions to create an index, possibly a „person" or „address" element, an „author" element. A means to create a reference or „link". You can decide to model all these technical elements from scratch or decide to steal them (I meant re-use them) from popular models like docbook. I recommend looking at docbook, especially at „mediaobject", „ulink", „index" etc. and use the elements there.

Step ten: define technical elements (2)

```
<figure><title>Some Title Text</title>
  <mediaobject>
     <imageobject>
          <imagedata fileref=„somepicture.png" format=„PNG" />
     </imageobject>
     <textobject> <para> This pictures describes xxxxxx </para>
     </textobject>
  </mediaobject>
</figure>
```

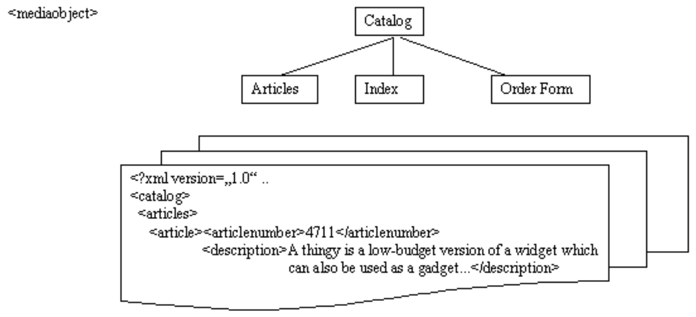
The mediaobject model allows to refer to arbitrary media (video, audio, images) in all kinds of formats. It also allows to enter alternative text e.g. to support disabled people. The easiest way to use those elements is to copy their definitions into your own model definition (which we will create later on as a DTD or XSD schema). XSD schema will allow you to re-use existing definitions. Name clashes can be solved through the use of name space definitions. The example is from the docbook dtd.

Step ten: define link elements



Some articles in our catalog might be related with each other. The Complements element allows us to express those dependencies which could have several possible types: e.g. increases use, necessary, practical extension, recommended add-on etc. This makes browsing our catalog much easier because we can show a user which things belong together or what optional articles might be useful.

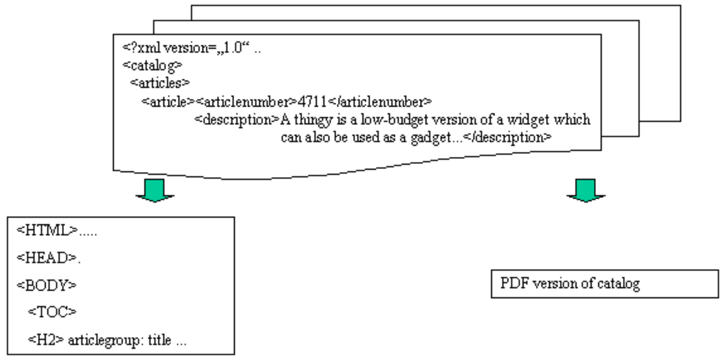
Step eleven: test your model (1)



```
<?xml version=„1.0“ ..
<catalog>
  <articles>
    <article><articlenumber>4711</articlenumber>
             <description>A thingy is a low-budget version of a widget which
                    can also be used as a gadget...</description>
```

It is important to create real instances of your model early on in the process. You will immediately see what is good and bad in your model and also how easy it is to work with. A side effect of this is also that you will detect new elements for your catalog: In this case it would be nice if a catalog had a title and if you could group articles into articlegroups with titles. Go back to your model and fix it.

Step eleven: test your model (2)

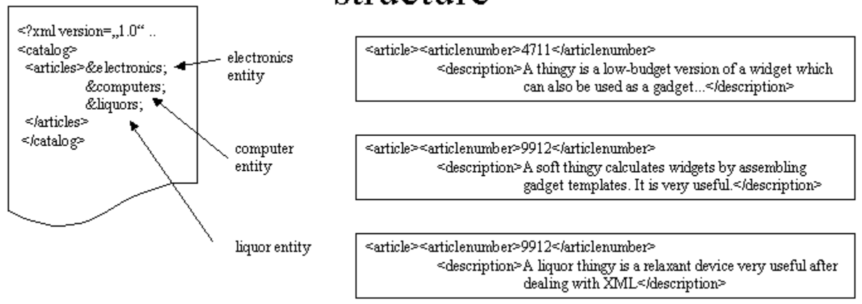


```
<?xml version=„1.0“ ..
<catalog>
  <articles>
    <article><articlenumber>4711</articlenumber>
             <description>A thingy is a low-budget version of a widget which
                    can also be used as a gadget...</description>
```

```
<HTML>.....
<HEAD>.
<BODY>
  <TOC>
  <H2> articlegroup: title ...
```

The next step in testing is to perform some renderings of your instances. The most important question: does your model contain everything so that you can support different media types and create different views on your catalog? You may find some elements missing, e.g. an element to emphasize some text in your description elements (an <emphasize> tag. You could make these parts blink in the online version and bold in the printed version. Add them to the model.

Step twelve: define the physical (entity) structure



```
<?xml version=„1.0“ ..
<catalog>
  <articles>&electronics;
            &computers;
            &liquors;
  </articles>
</catalog>
```

```
<article><articlenumber>4711</articlenumber>
         <description>A thingy is a low-budget version of a widget which
                can also be used as a gadget...</description>
```

```
<article><articlenumber>9912</articlenumber>
         <description>A soft thingy calculates widgets by assembling
                gadget templates. It is very useful.</description>
```

```
<article><articlenumber>9912</articlenumber>
         <description>A liquor thingy is a relaxant device very useful after
                dealing with XML</description>
```

The model defines your „element structure“. You will find out quickly that you want to partition your instances into smaller parts, e.g. to split work between colleagues or to insert fixed blocks of text (your „legal“ comments might be re-used for different products of your companies). You can use so-called „entities“ to pull in those parts. Please note that the parser will pull in the entities and create the final structure of your instance. This means the content contained in the entities need to fit into the element structure expected by the parser (which it „learns“ from your model definition)

Next Steps

- Create a number of instances of your catalog. Use a xml editor (xml-edit, morphon, XML-Spy) or just a text editor (emacs, wordpad etc.)
- Define your schema using DTD or XML-Schema notation. Tools like xml-authority (demo version available) help you write your schema or dtd.
- Use a parser to validate your catalog instance against the schema. We will use the xerces java parser to do this.

Next time we will meet in the lab to do exactly this.

Resources (1)

- Eve Mahler, Jeanne El-Andaloussi, Designing Document Type Definitions
- Charles Goldfarb, XML Handbook 4th edition: read about XML Authority and TurboXML in Chapter 36, "Building a schema for a product catalog".