Practical Reuse in XML

A little something on being as lazy as humanly possible

condesign

Here's what I want...

- Write only once and reuse whenever needed
- I want to be as specific as possible...
- ...yet reuse as much as possible

So this is what I need to talk about

- Naming and uniqueness (names *in addition to* addresses)
- Linking (why I prefer one linking system instead of several)
- Markup (targets, linking, profiling, and other tricks for the lazy)
- Practicalities (actually using it all in an editor)
- Publishing
- Translations
- Odds and ends

I wanted to find a phone book...

- ”Second floor, room 3, by the window, top shelf, right, yellow covers”
- ...the 1979 Yellow Pages in a room full of Yellow Pages...
- ...in Swedish...
- ...and a specific company's phone number, on a specific page

It’s easier if you have a name...

- You know what you’re looking for...
  - ...in whatever language...
  - ...and version
- Unique names, therefore, result in unique and up-to-date addresses
- If the 1979 Yellow Pages is a db object, there is a (unique) string from the librarian (your search facility) to the book (the object)

A good name...

- ...is not about handling a file in the file system
- It's not an object in a database, either

- A name should identify resources according to their use in an authoring system, so it should identify *semantic documents*
  - Documents (manuals), document fragments (sections, paras, warnings...), images

- The semantic document (the URN scheme) is an abstraction layer

What's in a name?

- An identifier for the semantic resource
- Language information
- Version information

`urn:x-cassis:r1:condesign:0123456:en-GB:0.1`

Namespace **Document ID** **Language & version**

In which case the semantic document is...

`urn:x-cassis:r1:condesign:0123456:*:*`

condesign:0123456

In other words...

- Translated versions are *renditions* of the original document...
- ...while the various versions indicate *progress*

How I do links (the cool stuff)

- Me, I've always been partial to XLink
  - It's a single spec
  - It's a simple spec

```
<ref xlink:href="target.xml#target-id"/>
```

  - XLink covers it all, from cross-references to document insets to image references
  - There are processing hints but no real processing model

  - ..furthermore, I love Extended XLink

On mechanism, not several

- Insets

  <inset xlink:href="urn:x-cassis:condesign:01234:en-GB:0.1#id-inset1"/>

- Images

  <image xlink:href="urn:x-cassis:condesign:9876:en-GB:0.14"/>

- Cross-references

  <ref xlink:href="urn:x-cassis:condesign:1357:en-GB:0.99#id-xref1"/>

- (etc)

- In other words, let the element type decide

Targets

```
<inset xlink:href="warning.xml"/>
```

```
<warning id="warning-id">
  <p>Horrible things will happen if you push the button!</p>
</warning>
```

- Crude, I know, so better is...

Fragment identifiers in use

```
<inset xlink:href="warning.xml#warning-id"/>

<doc>
  <title>My warnings</title>
  ...
  <warning id="warning-id">
    <p>Horrible things will happen if you push the button!</p>
  </warning>
  ...
</doc>
```

- You can point at a fragment within a document (and yes, it's like @xpointer in XInclude)
- Anything with an ID is a valid target

Size matters, or...?

- Large documents are harder to reuse
  - Too few reusable contexts
  - Too product-specific
  - Too complex
  - Too specific to a product or description...
- Smaller documents are harder to find
  - With 12,000 individual paragraphs, how will you find the one you wrote 8 months ago?
  - Reusable "phrases" are often duplicated
  - It takes (far) longer to find a reusable component than to write it
  - The language is stilted
- We need a middle ground

Well, yes and no (and no, it's not too small)

- A special document collection, a ”warnings document” (or "phrases", or "notes", or...), offers advantages:
  - Easier to group similar reusable content
  - Easier to include comments and surrounding information
  - Easier to find the reusable content
  - Easier to implement
- You don’t need a special case for including the whole document, meaning that you can use the same linking software

- But also, it becomes easier to *profile the document...*

Profiling: setting filters on content

- The basic idea is incredibly simple:

```
<doc applic="A">
  <p>Information common to products A and B.</p>
  <p applic="A">Information about product A.</p>
  <p applic="B">Information about product B.</p>
</doc>
```

- Which results in...

> Information common to products A and B.
> Information about product A.

Profiling conditions

- The root node sets the filter conditions
- Nodes without applics are always included
- Normal processing is "OR" (A | B)...

```
<doc applic="A C D">
  ...
  <p applic="A B">...</p>
  ...
</doc>
```

- ...but it's doable to do an "AND" (A & B)

How to implement profiling

- *Don't* define the filters in the DTD, use a database instead—much easier to update and define *relations* ("A *if* B", "C *if not* A", etc)
- Consider using an abstraction layer instead of descriptive applics (i.e. URNs)
  - A change in naming could result in a very large conversion project
  - URNs could be used to represent *semantic groups* of filters
  - Besides, it would be really cool
- Consider variables:

```
<p>The <phrase applic="XC60"/> is an exciting new model.</p>
```

Profiling in the XSLT

- It's relatively easy, using XSLT

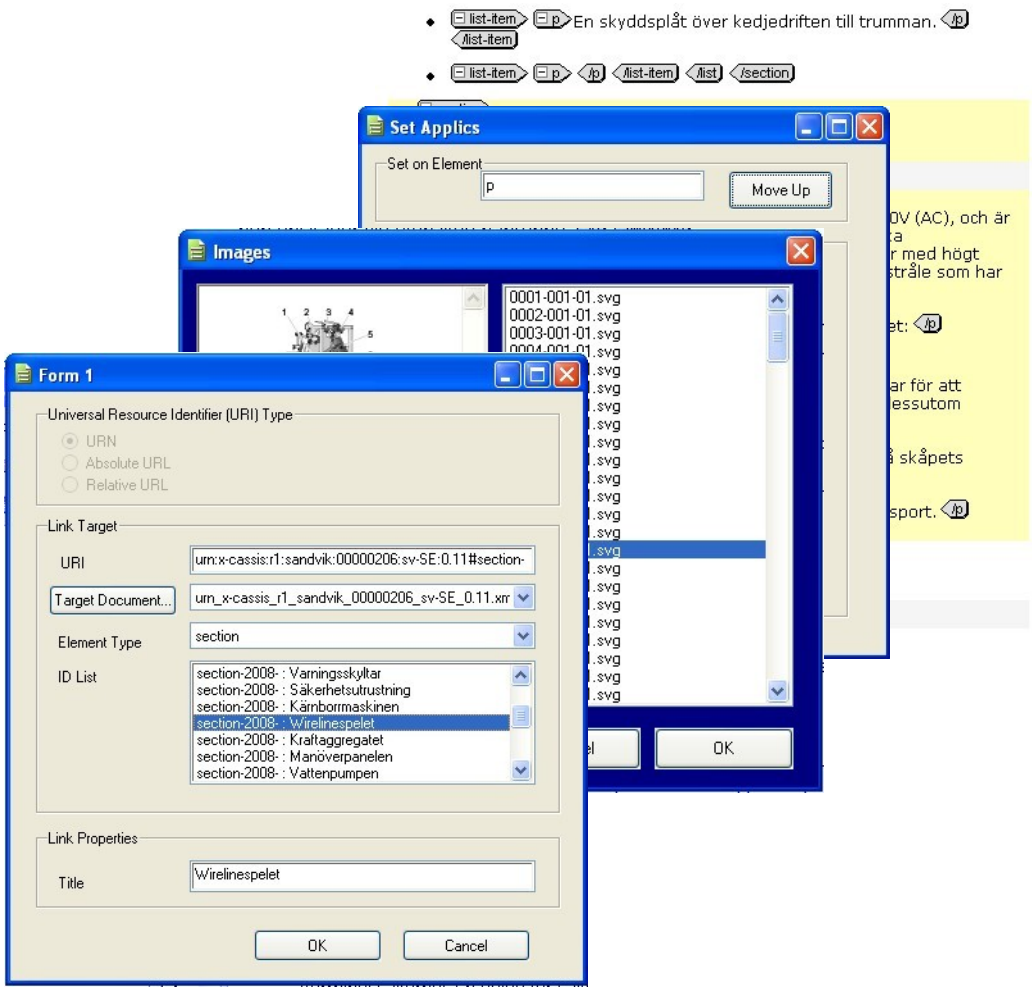
```
<xsl:template match="*">
  ...
  <xsl:if test="not(@applic) or $print='yes'">
    <xsl:element name="{name(.)}">
      <xsl:call-template name="attribute-copy"/>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:if>
</xsl:template>
```

```
<doc applic="S80">
  ...
  <p applic="V70 S80 XC90">
    ...
  </p>
  ...
</doc>
```

- Depending on the complexity of the applics filtering, producing the `$print` value can be more or less complicated

A truly useful *editing environment* needs to...

- Generate ID values
- Generate URNs
- Map URNs to current URLs (and back)
- Provide linking software
- Help with profiling



Mapping URNs and URLs locally


```
<map>
  <pair>
    <urn>urn:x-cassis:r1:sandvik:00000206:sv-SE:0.15</urn>
    <url>C:\Temp\XMetaLCache\urn_x-cassis_r1_sandvik_00000206_sv-SE_0.15.xml</url>
    <status>RO</status>
  </pair>
  <pair>
    <urn>urn:x-cassis:r1:sandvik:00000249:sv-SE:0.38</urn>
    <url>C:\Temp\XMetaLCache\urn_x-cassis_r1_sandvik_00000249_sv-SE_0.38.xml</url>
    <status>RO</status>
  </pair>
</map>
```

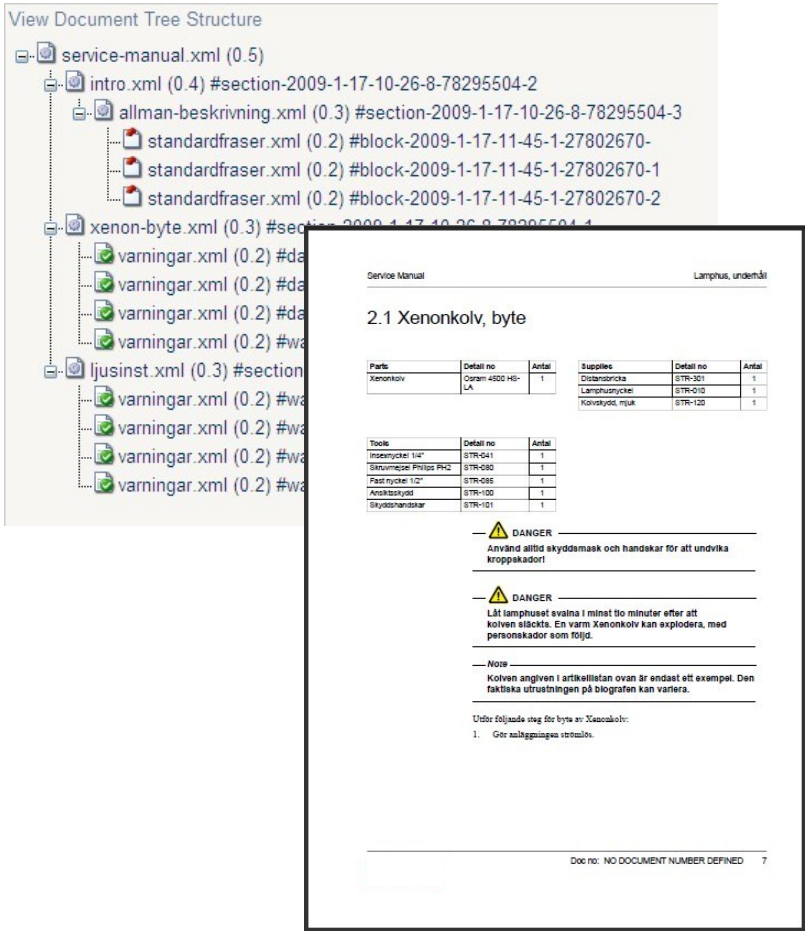

Condition	URN	URL
Checked out	YES	YES
Linked (opened) but not checked out	YES	NO
Checked in	NO	NO
Unregistered in CMS but NEW in editor	NO	YES

Keeping Track of it All

- Reusing is a presentational problem, a visualisation problem

Publishing

1. Parse the first (root) document for URNs pointing at other docs
2. Open those in a temp area
3. Replace URNs with URLs in document
4. Repeat 1-3 with every opened document until done
5. Normalise and filter/process applics
6. Feed the normalised document to the publishing engine



condesign

Handling translations is extremely easy...

```
<doc>
  <inset xlink:href="urn:x-paper:r1:doc0002:en-GB:1.0"/>
  <inset xlink:href="urn:x-paper:r1:doc0003:en-GB:1.0"/>
</doc>
```

```
<doc>
  <inset xlink:href="urn:x-paper:r1:doc0002:sv-SE:1.0"/>
  <inset xlink:href="urn:x-paper:r1:doc0003:sv-SE:1.0"/>
</doc>
```

...because the translated document is really a copy of the original

Manage Translations

English (United Kingdom)

- service-manual.xml (0.5)
  - intro.xml (0.4) #section-2009-1-17-10-26-8-78295504-2
    - allman-beskrivning.xml (0.3) #section-2009-1-17-10-26-8-78295504-3
      - standardfraser.xml (0.2) #block-2009-1-17-11-45-1-27802670-
      - standardfraser.xml (0.2) #block-2009-1-17-11-45-1-27802670-1
      - standardfraser.xml (0.2) #block-2009-1-17-11-45-1-27802670-2
  - xenon-byte.xml (0.3) #section-2009-1-17-10-26-8-78295504-1
    - varningar.xml (0.2) #danger-2009-1-17-10-24-15-97687159-
    - varningar.xml (0.2) #danger-2009-1-17-10-24-15-97687159-1
    - varningar.xml (0.2) #danger-2009-1-17-10-24-15-97687159-1
    - varningar.xml (0.2) #warning-2009-1-17-10-24-15-97687159-2
  - ljusinst.xml (0.3) #section-2009-1-17-10-26-8-78295504-4
    - varningar.xml (0.2) #warning-2009-1-17-10-24-15-97687159-3
    - varningar.xml (0.2) #warning-2009-1-17-10-24-15-97687159-4
    - varningar.xml (0.2) #warning-2009-1-17-10-24-15-97687159-4
    - varningar.xml (0.2) #warning-2009-1-17-10-24-15-97687159-4

Create Package

Profiling according to market or country...

- ...is NOT something we should do with what essentially is the xml:lang mechanism
  - What if we have two (or more) market customisations for the same language and country?
  - xml:lang is meant to handle translated content *after* filtering
- Market- or country-specific profiles are just that, profiles
  - Translations are renditions, not variants
  - Define profiles (applics) instead!

In conclusion...

- ***Use an abstraction layer to name your resources!!!***
- URN schemes are a Good Thing!
- Translations are renditions of the originals
- Use one linking system instead of many
- Use profiling
- Generate IDs, URNs and offer GUIs for linking/profiling

Questions?