OBJECTIVITY
SYSTEMS INTEGRATORS

# Transforming Applications


## Peter den Haan, Steve Mudford

## Objectivity Ltd.



*Over the last few years XML has become the format of choice for representing structured data. As a result a number of technologies have arisen to simplify the manipulation of XML. XSL Transformations (XSLT) in particular will be covered by this paper.*

*After a brief, high level introduction, we will explore how XSL Transformations can be used to simplify development tasks. The main areas of interest are data presentation, application generation, process management and configuration file generation. Case studies and examples of actual projects are given throughout in order to expand on the ideas and concepts that are presented.*

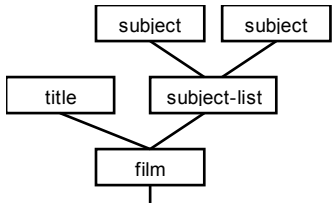

## 1. Introduction

This paper is perhaps best introduced by describing what it is not. For a start, even though we will give a brief overview, this is not an introduction to XML or XSLT. Neither is it an in-depth treatment; if you have used XSLT in anger, you are unlikely to learn anything new about it here. XSL Transformations are a uniquely flexible tool in the software developer's toolset, and this paper aims to be a loose but thought-provoking discussion of some of the conventional and less conventional uses it can be put to. As such it will be neither systematic nor exhaustive, but an opportunistic mix of techniques, tools, and case studies entirely in the spirit of postmodern programming [Post].

### 1.1. XML

An XML file [XML] is essentially a textual representation of tree-structured data:

```
<?xml version="1.0"?>
<film id="1411609">
  <title>Beautiful Girls</title>
  <subject-list>
    <subject>COMEDY</subject>
    <subject>DRAMA</subject>
  </subject-list>
</film>
```



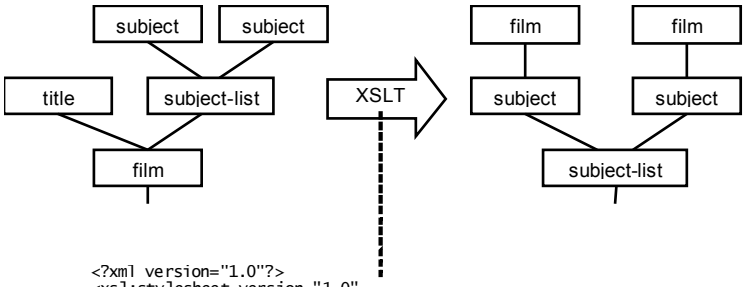
Of course, there is a lot more to say about XML tags, attributes and other nodes, DTDs, and XML schemas, but for our purposes they are just implementation details.

XML being a textual representation of tree-structured data means two things. Firstly, it means that it is generally quite useless for data which isn't readily represented as text; a JPEG image, for instance. Secondly, it means that you have to hammer your data into a tree structure, although you can establish cross-links by using ID attributes. Fortunately, a surprisingly large amount of data maps more or less naturally to a tree.

## 1.2. XSLT

XSL (XML Stylesheet Language) Transformations [XSLT] is a language to transform an input XML tree into an arbitrary output XML tree. XSLT is an XML-based template-matching language of considerable complexity and power.

By way of a simple example, we could turn above XML tree which associates subjects with a film into one which associates films with a subject:




```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <subject-list>
      <xsl:apply-templates select="/film/subject-list/subject"/>
    </subject-list>
  </xsl:template>

  <xsl:template match ="subject">
    <subject name="{text()}" >
      <film id="{../../film/@id}"/>
    </subject>
  </xsl:template>

</xsl:stylesheet>
```

This example would be more realistic if we had a film list – and if we would merge the inevitable duplicated subjects – but such detail would merely obscure the key point that XSL Transformations allow you to transform between different tree representations of your data with a minimum of fuss and a maximum of productivity.

The growing number of possible applications of XSLT derives from the fact that an increasing amount of data, messages, scripts, configuration files, remote procedure invocations, and other electronic information is either stored in XML format, or at least available in XML format. In the following sections we will examine a number of different ways to capitalise on the information available in enterprise systems and enterprise Java development and the fact that we have a lightweight, quick way of transforming it into something completely different.

## 2. Data Transformation

Transformation of data into a representation of that data in a different vocabulary, often without information loss, is the most obvious application of XSLT. The transformation from the previous section would be an example.

- You may wish to transform from one XML vocabulary to another. A common example would be when you need database data in some predefined XML output format. You can use the Oracle XML SQL utility or XSQL servlet to select database data in XML format, but the resultant XML is unlikely to be of the required format. XSLT can realise the required transformation quickly and effectively.

- The vocabulary you transform to need not be an XML-based one, though. For example, one possible way to feed XML data into a relational database is to transform the XML into SQL statements. This is not necessarily the most robust way to go about it, but if your data source is trusted it can be quick and effective. Alternatives would be to use an XML-relational mapping tool or go the XML database route using Oracle XML DB or another RDBMS-backed XML database.

- Conversion into just about any other text format is possible, for example comma-separated values or Java properties files.

Note that you are not restricted to a single input XML file as a data source. The `document()` function allows access to other XML documents, and most XSLT processors support extension functions that allow querying of JDBC, JNDI and other data sources. If these do not fit the bill, you can write your own extension functions in Java, which is not nearly as hard as you might think.

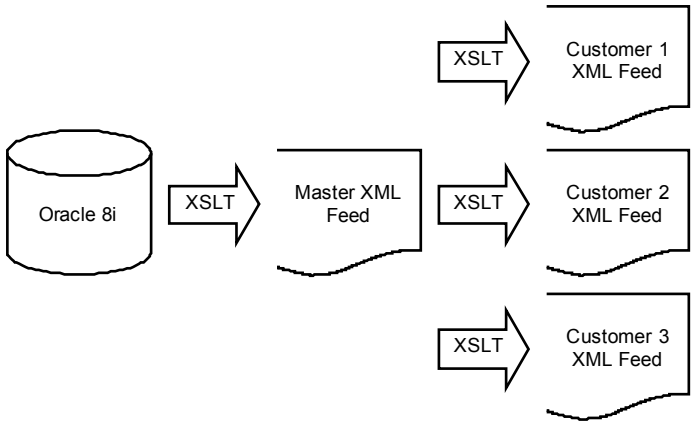
### 2.1. Case Study: IPC Tx Schedule Data Feeds

IPC Magazines, the dominant TV magazine publisher in the UK with titles such as *TV Times* and *What's on TV* in its portfolio, wanted to expand their business by providing TV schedules, plus value-adds such as reviews, to third parties. The data was held in an ordinary, relational Oracle database.

The obvious choice for the feed data formats was XML. Each customer would require different information in a different XML format. Rather than extract each feed from the database individually, the customer feeds are all transformed from a master XML feed. This master feed contains all the information held in the database and runs on a daily basis.

Of course, the master XML feed could have been the raw output of a couple of queries to the Oracle XML SQL utility, but adding an XSLT step to produce the master feed has two advantages: first, it allows the master feed format to resemble the customer feeds very closely so that the customer's XSLT stylesheets remain as simple

as possible; second, it eases maintenance by isolating the master feed from changes in the underlying relational database model.




This system is currently in successful production and serves three different customers with daily feeds; the incremental cost of setting up a new customer has turned out to be very low thanks to the architecture and the simplicity of writing XSLT stylesheets. We initially used Apache Xalan-J but ultimately chose Saxon for its superior speed on the very large XML documents involved. An XSLT extension function written in Java adds a digital watermark to the feed contents to prevent unauthorised syndication. It is perhaps interesting to note that the entire process is controlled by Apache Ant [AAnt], a tool usually associated with compiling and building Java-based software but which is actually a great way to control *any* kind of batch process.

When XML processing pipelines get a bit more complicated than this, it can be a good idea to use a dedicated XML document processing product which will manage an entire complicated pipeline; if you want to experiment with a simple tool, try NekoStyle [Neko].

## 3. Data Presentation

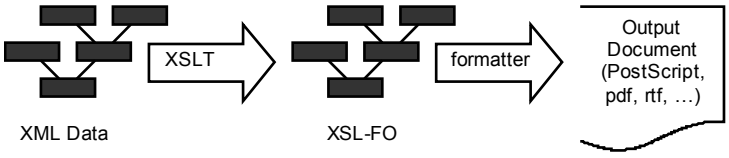
XSLT is even more commonly used for presentation purposes, using the stylesheet to add presentation information to the raw data contained in the source XML file.

- The presentation language may be XML-based, such as XHTML or WML, or almost so in the case of HTML. This is the bread and butter of almost all portal applications, for example Oracle Portal, and of XML publishing frameworks such as Apache Cocoon [ACo].

- As of version 1.2, Java Server Pages (JSPs) support a fully XML compliant syntax in addition to the traditional JSP syntax. This makes it easier to generate JSPs from XML information and, conversely, to transform such JSPs using XSLT.

- With a bit more effort and perhaps a stylesheet library, XSLT can also be used to produce plain text output. The language is complete and expressive enough to perform tasks such as word wrapping and table-type layouts.

- The most sophisticated possibilities for data presentation are offered by XSL [XSL], of which XSLT is only one component. It consists of an XSL Transformation stage which outputs an intermediate tree. This is then processed by a *formatter*. The intermediate tree contains formatting objects (XSL-FO) which describe the desired layout.




If you have an academic background in science you may be familiar with Donald Knuth's TeX system [TeX]; in some ways, XSL-FO is a bit like an XML version of TeX.

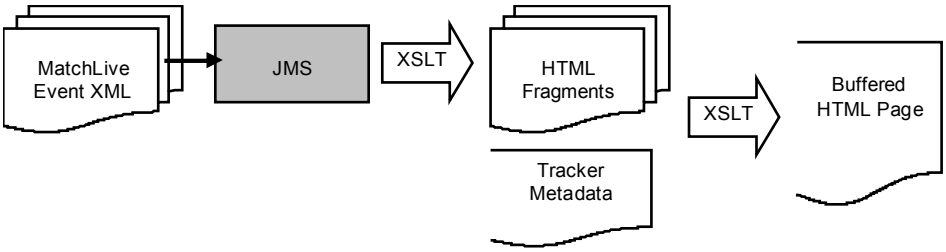
As most readers will be familiar with this type of application, we will not dwell on it for too long.

## 3.1. Case Study: F.A. Premier League Live Match Tracker

The new F.A. Premier League web site (http://www.premierleague.com) is supplied with a number of XML data feeds by the Press Association (PA). During Premier League matches, XML files describing interesting events such as goals, fouls and substitutions are continually fed into the system. This information needs to be displayed in a Live Match Tracker frame which is part of a football statistics console.


As essentially the task of this subsystem is to convert XML information into HTML (and possibly later other formats such as WML), XSLT was a natural candidate. The challenges were mostly in the areas of communication (there can be any number of web servers in the cluster) and sheer volume of traffic.

The communication issues are solved by publishing the XML event document to a JMS topic (essentially a simple version of XML messaging). On the web servers, this XML is transformed into to an HTML fragment using XSLT. A further XSLT stylesheet is used to transform all current fragments into a complete HTML page which is buffered in memory. As a consequence, the HTML is rendered only when a new event arrives through JMS and the contents of the tracker need to change. As the tracker refreshes every 30 seconds, the site experiences a load of at least 333 hits a second for every 10,000 users during live matches. It would be impossible to perform the transformation for every HTML request.




This solution scales well and changes in the way events are listed, and which are reported on in the first place, are almost trivial to make.

# 4. User Interface and Content Management

Another area where XML is becoming ubiquitous is that of user interfaces and, to a slightly lesser extent, content management. This opens interesting opportunities, such as the ability to automatically generate GUIs or indeed complete content management functionality from metadata in XML format.

In particular, you can either query the Oracle data dictionary using XSQL/XMLSQL or interrogate the JDBC driver programmatically to generate an XML document describing the tables, fields, keys and relationships in your database. This document can be further processed using XSLT.

## 4.1. XML GUIs

Although a Java Swing user interface still has to be constructed programmatically, usually with a builder tool, there are a number of third party tools that allow you to define a GUI using XML. Many combine this with delivery over the web, often using Java WebStart (JNLP).

- The most interesting are standards-based ones. Mozilla set a standard of sorts with the XML User Interface Language (XUL); interesting Java implementations are Luxor XUL [Luxor] which is based on the Swing toolkit, and the XML Windowing Toolkit [XWT] which can use an ActiveX browser plug-in as well as a Java applet and uses JavaScript to automate the user interface.

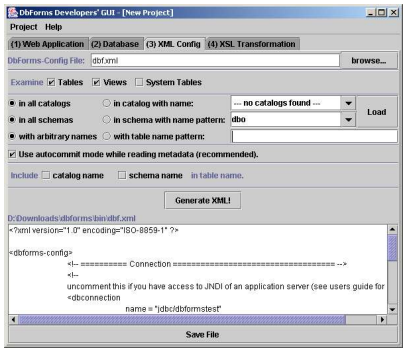
- A standard which looks to become increasingly important in the near future is W3C XForms [XFms]. Interesting aspects of XForms are that it can map to all kinds of user interfaces ranging from PDAs through Web pages to traditional GUIs, and that the underlying data model for a form is an XML document. This makes XForms especially suitable for XML-based content management. Java XForms implementations are currently few and far between; open source ones worth mentioning are Chiba [Chiba], which runs in a J2EE container and renders a traditional HTML browser interface, and X-Smiles [XSls] which targets exotic devices such as mobile phones.

- A proprietary, but remarkably lightweight example of an XML GUI toolkit is Thinlet [Thin]. It renders a polished GUI from an XML file, has a particularly simple way of integrating with your Java code, and weighs in at an amazing 27KB. As it is AWT-based it works on JDK 1.1 or better; it can be used both in applets and in standalone applications, and is open source (LGPL).

In all of these cases one could easily write an XSLT stylesheet which, given a database metadata XML file, would generate a boilerplate user interface to your corporate standard. The amount of work involved is modest enough to fit in most projects even as a one-off.

## 4.2. Generating Content Management

One can take the above one step further and generate entire content management applications from an XML (database) metadata document using XSLT, either on the fly or as a boilerplate for further development.




One interesting example of this, again from the open source arena, is a rapid web application development tool called DbForms [DbF]. This is a combination of a database-oriented tag library, an MVC back end controlled by an XML configuration file, and finally a simple GUI tool which will generate all necessary JSPs and configuration information for a basic content management application from database metadata using XSLT. The framework supports validation, user rights, internationalisation, and provides hooks for your own Java code when you need to go beyond the framework's limitations.

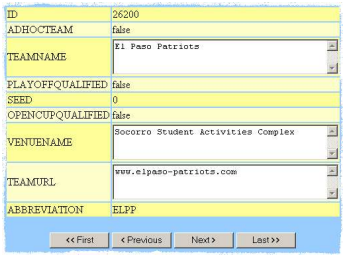
It is worth mentioning that the stylesheets used can be modified. Even if DbForms does not fit your needs, you can still add your own stylesheets to the GUI tool and use it to generate arbitrary code and configuration information from the database metadata XML or even, since it is open source, adapt the tool to your own needs.

## 4.3. Case Study: UI Generation using DbForms

For internal purposes, we needed a quick interface to edit the data in some tables that did not have or normally need content management. It took an hour to set up DbForms, read the documentation, point the generator tool at the database and generate basic JSPs for all relevant tables. We then manually modified the generated menu (which simply lists all JSPs) to provide the workflow we needed.

It may look functional rather than pretty, but it did a great job. Had we wanted to turn it into a production-worthy application, the two most important areas which would have needed attention were validation and look & feel.

For validation, DbForms uses the Apache Jakarta Commons [ACom] Validator, which is configured declaratively using an XML configuration file. The look and feel of the generated pages can be modified simply by modifying the stylesheets used. All in all developer productivity appears to be an order of magnitude better than anything we have used so far and we are currently investigating the tool for production purposes.




## 5. Application Generation and Transformation

XSLT is not just useful to generate XML files of all kinds – you can also use it to generate Java code or, for that matter, code in any other language. Of course, code generators and wizards are nothing new and quite common in today's development tools. Use them where you can. The key point is that with limited effort, *you* can write code generators for *your* problems that are specific to *your* environment, and boost your effectiveness significantly.

There are an endless number of sources for Java code generation:

- A database dictionary in XML format could be used to generate a JavaBeans object model, data access objects, validation classes or content maintenance code. Of course one would only do this where existing tools like Oracle TopLink [OIAS], Castor [Cast] or the plethora of other database mappers did not fit the job.

- An XML DTD is not easily processed using XSLT, but an XML Schema is an XML document which can be transformed like any other. The possibilities are similar to those mentioned above. Again tools such as the Oracle XML Class Generator [OXml] or Castor [Cast] will often do the job.

- If you use a CASE tool and its code generation facilities fall short, you might be able to generate Java code from its XMI model.

- You can use your Java code to generate further code or other information! This is the bread and butter of tools like XDoclet [XDoc] and of course `javadoc`, but much more interesting in the context of this paper is IBM's *Reengineering Toolkit for Java* [Ret4j]. This toolkit will convert Java code or class files to an XML representation and vice versa. Consequently, you can transform Java source and even bytecode in a lightweight way using XSLT.

  This can be used in a number of ways. XML Schemas and database designs can be generated from Java source (although again existing tools should be explored first). Documentation and code metrics can be generated even from class files. Where a number of related Java classes need to be written, as for example in Enterprise JavaBeans, some may be generated from the others.

It is impossible to be exhaustive in a list like this; all we can do is give examples that might inspire you to find applications relevant to your own environment.

## 5.1. Case Study: Transfer Object Generation

The Transfer Object J2EE design pattern [J2EEP] (also known as Value Object) is used to eliminate the overhead of multiple network calls to retrieve multiple EJB properties by wrapping these properties in a single Transfer Object which can be transferred over the network in a single call.

Such Transfer Objects can be generated automatically from the EJB remote interface using the reengineering toolkit discussed above; we used this as a simple test case to evaluate the technology. Although the stylesheet is quite simple, the XML representation of a Java source file is quite verbose and at 150 lines the stylesheet is unfortunately too large to include in this paper. The stylesheet took about an hour to put together.

# 6. Process and Configuration Management

XML is also today's number one choice for configuration files. This is in many ways an improvement over proprietary binary formats: they are now much more amenable to version control and tools like `diff`, you can actually see what is in them, and they can be processed or generated using XSLT. The latter can be profitably exploited in many ways.

- When you are integrating different systems or developing a system based on a number of different components, it is easy to end up with a dozen different configuration files in as many locations. The information in these files is often overlapping and interdependent. In a software development environment, you probably need to maintain sets of configuration information for several environments: development, testing, production, and usually more.

  With XSLT, it is straightforward to put all relevant information in a single master XML configuration file that you define, and generate all other configuration files from this as part of the build or deployment process. This greatly helps maintenance as well as development; with all important information gathered in a single file, changing configuration information is far less error-prone.

- Configuration information can be extracted, either from the master configuration file or from the individual configuration files, and processed into a summary document which can be e-mailed, printed, or made available from a web server.

- The de facto standard for Java software build processes is Apache Ant [AAnt]. Not only is Ant a powerful batch process tool which can be used to

automate many of the transformations discussed so far; its build file itself is an XML file which can be generated from high-level information.

- Many database tools, such as the Castor [Cast] object-relational mapping tool, have XML configuration files which can often be wholly or partly generated from database metadata.

In a project-based software development team, a well-defined project structure and a robust automated build process can really help productivity, quality and, eventually, ongoing maintenance. We will look at an example of project and build generation next.

## 1.1. Case Study: Build Process Generation

About a year ago, Objectivity had realised a number of Java projects where the build process was automated using the Apache Ant build tool. We experienced the following problems:

- Constructing and maintaining comprehensive Ant build files is hard work. Some things that could be automated never were because it was hard to justify the effort.

- Every project and build process worked a bit differently.

- In the course of several projects, new features had been introduced, such as better configuration management along the lines described above, automated unit testing using JUnit [JU], enforcement of company coding and documentation standards using Checkstyle [CSt], and support for continuous integration using Cruise Control [CC]. Many of these should have been retrofitted into existing projects, but changing the build process again was cumbersome enough that it never happened.

The solution was to create an XSLT-based *project generator*. At the start of a project, you write a `project.xml` file which describes the main project characteristics:

```
<?xml version="1.0"?>
<project buildgen-version="1.0">
  <module type="jar" name="feeder">
    <junit/>
    <audit excludes="com/fapl/feed/castor/**/*"/>
  </module>

  <module type="taglib" name="engine">
    <audit excludes="uk/co/objectivity/queryengine/config/*"/>
  </module>

  <module type="war" name="stats">
    <dependencies>
      <depends name="engine"/>
    </dependencies>
  </module>
```

```
    <documentation>
      <link>http://java.sun.com/j2se/1.3/docs/api</link>
      <link>http://java.sun.com/j2ee/sdk_1.3/techdocs/api</link>
    </documentation>
  </project>
```

Above configuration file describes the F.A. Premier League statistics project, consisting of three modules: a Java application packaged in a jar, called *feeder*; a Web application packaged in a war called *stats*, and an *engine* JSP tag library that is incorporated in the stats web application. The `documentation` section specifies external API documentation that the generated JavaDoc will be linked with.

The generator uses this XML file to generate the project directory structure, copy in default libraries and scripts, and generate a complete Ant build file for the modules and the project as a whole. This build process encompasses all the features and tools that we have come to use. Should we add new refinements in the future then they can be retrofitted into this project simply by regenerating the project from the existing `project.xml` file.

It is perhaps interesting to mention that the initial project directories are created using a two-stage process: first, the `project.xml` file above is transformed, using XSLT, into a temporary Ant build file containing the appropriate `mkdir` and `copy` instructions; then, this build file is executed and finally deleted. XSLT `imports` and template overrides are used to share as much code as possible between the different module types.

## 7. Conclusions

From the smörgåsbord of lists and examples we have seen so far, we can draw three conclusions.

First, a large amount and diversity of information is available in XML format and/or can be sourced from XML files, and the amount is growing day by day. Its very ubiquity is a great part of the power of XML, and networking effects seem only to accelerate its development.

Furthermore, once you have information available in XML format XSLT can be used to transform it; indirectly, this means that XSLT is available as a lightweight way to transform data that we would normally never think of as XML, such as database schemas or Java source files.

Finally, if you have not done so long ago, now is the time to download the Oracle XML Development Kit [OXml] and probably some of the other tools mentioned, and start familiarising yourself with XSLT, the tools and their applications. The learning curve can be steep for some, but from our own experiences we can promise that it will transform your XML files, your applications, and ultimately your productivity.

Peter den Haan (pdenhaan@objectivity.co.uk)
Steve Mudford (smudford@objectivity.co.uk)

http://www.objectivity.co.uk

## 8. References


[AAnt] *Apache Ant,* http://jakarta.apache.org/ant/

[ACom] *Apache Commons,* http://jakarta.apache.org/commons/

[AXml] *Apache XML Project*, http://xml.apache.org/

[Chiba] *Chiba* XForms implementation, http://chiba.sourceforge.net/

[Cast] Exolab *Castor* OR and XML mapping tool, http://castor.exolab.org/

[CC] ThoughtWorks' *Cruise Control* continuous integration tool has been open sourced on http://cruisecontrol.sourceforge.net/

[Cst] *Checkstyle* coding standards checker, http://checkstyle.sourceforge.net/

[DbF] *DbForms Rapid Application Development*, http://www.dbforms.org/

[Ret4j] IBM's *Reengineering Toolkit for Java,* http://www.alphaworks.ibm.com/tech/ret4j

[J2EEP] *Core J2EE Patterns*, Deepak Alur, John Crupi, Dan Malks, Prentice Hall, ISBN 0130648841; see also http://java.sun.com/blueprints/corej2eepatterns/ for a good overview of the 15 core patterns.

[JU] The home of the *JUnit* framework is http://www.junit.org. Apart from JUnit itself, you will find links to many extensions which enhance or build upon the framework.

[Luxor] The *Luxor XUL* Project, http://luxor-xul.sourceforge.net/

[Neko] Andy Clark's *NekoStyle* XML pipeline framework can be found at http://www.apache.org/~andyc/neko/doc/style/index.html

[OIAS] Oracle 9iAS, http://otn.oracle.com/products/ias/

[OXml] Oracle XML Center, http://otn.oracle.com/tech/xml/

[Post] *Notes on Postmodern Programming*, James Noble, Robert Biddle; http://www.mcs.vuw.ac.nz/comp/Publications/CS-TR-02-9.abs.html

[TeX] *The TeXbook,* Donald E Knuth, AWL, ISBN 0201134470; *TeX: The Program,* Donald E Knuth, AWL, ISBN 0201134373

[Thin] Robert Bajzat's *Thinlet* toolkit can be found at http://www.thinlet.com/

[XFms] *XForms 1.0*, Micah Dubinko, Leigh L. Klotz Jr, Roland Merrick, T.V. Raman (eds); http://www.w3.org/TR/xforms

[XDoc] *XDoclet*, http://xdoclet.sourceforge.net

[XML] *Extensible Markup Language (XML) 1.0* (Second Edition), Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler (eds); http://www.w3.org/TR/REC-xml

[XSL] *Extensible Stylesheet Language (XSL) 1.0*, Adler et al; http://www.w3.org/TR/xsl

[XSLT] *XSL Transformations (XSLT) 1.0*, James Clark (ed); http://www.w3.org/TR/xslt

[XSls] *X-Smiles, an open xml-browser for exotic devices,* http://www.xsmiles.org/

[XWT] Adam Megacz' *XML Windowing Toolkit* can be found at http://www.xwt.org/