Xml Technology Study
Current XML Technologies

by
Mika Maunumaa

Preparation for Sadie project
Summer 2002 – Spring 2003
Instructor: Tarja Systä
# Table of Contents

1. **INTRODUCTION** .......................................................................................................................... 5
   1.1 PURPOSE OF THE STUDY ........................................................................................................... 5
   1.2 PURPOSE OF THE DOCUMENT ............................................................................................... 5
   1.3 THE AUDIENCE .......................................................................................................................... 5
   1.4 ENVIRONMENT .......................................................................................................................... 5
   1.5 DEFINITION OF TERMS ............................................................................................................. 5

2. **BACKGROUND TECHNOLOGIES** ................................................................................................. 8
   2.1 DOMAIN OBJECT MODEL (DOM) ............................................................................................ 8
   2.2 TRAVERSING AND SEARCHING (XPATH) .............................................................................. 9
      2.2.1 Description ......................................................................................................................... 9
      2.2.2 Tool support ...................................................................................................................... 11
   2.3 BINDING RESOURCES TOGETHER (XLINK) ............................................................................ 11
      2.3.1 Description ....................................................................................................................... 11
      2.3.2 Tool support .................................................................................................................... 12
   2.4 POINTING RESOURCES (XPOINTER) ..................................................................................... 12
      2.4.1 Description ....................................................................................................................... 12
   2.5 DEPLOYMENT MODULARITY (XINCLUDE) ............................................................................. 12
      2.5.1 Description ....................................................................................................................... 12
   2.6 PARTIAL DOCUMENT (XFRAGMENT) ...................................................................................... 13
      2.6.1 Description ....................................................................................................................... 13
   2.7 GRAMMARS FOR DOCUMENT HIERARCHY ........................................................................... 14
      2.7.1 General ............................................................................................................................. 14
      2.7.2 Document Type Definition (DTD) .................................................................................. 14
      2.7.3 XML Schema ................................................................................................................... 15
      2.7.4 Other schema descriptions ............................................................................................... 15

3. **XML PROCESSING TECHNOLOGIES** ......................................................................................... 17
   3.1 GENERAL ................................................................................................................................. 17
   3.2 PARSING XML .......................................................................................................................... 17
      3.2.1 General ............................................................................................................................. 17
      3.2.2 Scanning ........................................................................................................................... 17
      3.2.3 Event based SAX (Simple API for XML) .................................................................. 18
      3.2.4 Tree based DOM (Document Object Model) ............................................................. 18
      3.2.5 Comparing SAX and DOM ......................................................................................... 19
   3.3 VALIDATION ............................................................................................................................. 19
   3.4 TRANSFORMATIONS ................................................................................................................ 20
      3.4.1 General ............................................................................................................................. 20
      3.4.2 Cascading Style Sheets, level 2 (CSS2) .................................................................... 20
      3.4.3 XSL / XSLT ..................................................................................................................... 21

4. **XML SCHEMA (HIERARCHICAL SCHEMAS)** ............................................................................. 23
   4.1 BASIC ELEMENTS .................................................................................................................... 23
   4.2 SIMPLE DATA TYPES ............................................................................................................. 23
   4.3 OBJECT ORIENTATION ......................................................................................................... 24
      4.3.1 Inheritance ....................................................................................................................... 24
      4.3.2 Abstract document type ................................................................................................. 25

5. **XML STYLES AND STYLE GUIDELINES** ................................................................................... 27
5.1 XML STYLE GUIDELINES ........................................................................................................27
5.2 XSLT ......................................................................................................................................27
5.3 XML SCHEMA ......................................................................................................................27
5.4 XPATH ....................................................................................................................................27

APPENDIX A REFERENCES ........................................................................................................28
   A-I BOOKS AND STANDARDS .................................................................................................28
   A-II WEB SITES .......................................................................................................................28

APPENDIX B TOOLS ..................................................................................................................29
   B-I XML PARSING ...................................................................................................................29
      B-I.1 XML Tools ..................................................................................................................29
      B-I.2 Other ..........................................................................................................................29
   B-II TRANSFORMATION .........................................................................................................29
   B-III SERIALIZATION ..............................................................................................................30
1. INTRODUCTION

This chapter explains what this document covers. In addition, a short terminology list defines the most common terms used in this document. The chapter contains material that could be familiar for advanced reader and thus could be skipped.

1.1 Purpose of the study

This study is for studying and evaluating XML technologies eventually available for efficient XML document processing and an essential focus is in parsing and processing of XML documents. Presently there is a growing number of very large XML documents and finding efficient and/or fastest way to process them is important. The fast translation of XML based information from one structure to another (XML to HTML) has also become important. Because in the present it is common that the information is combined from various sources, the web of those relations can grow beyond human comprehension. Thus finding ways to seek those relations between XML documents is also major issue in the study.

1.2 Purpose of the document

This document has three major goals: to check the status of XML processing techniques, to explore XML processor’s capabilities to handle hierarchical schemas, and to collect guidelines for efficient XML.

The first goal is the largest part of the study. It collects all commonly available XML parsing and processing technologies. The focus is in XML data processing and translation techniques. Other subjects are abstract schemas and their support in current tools, and object serialization with XML.

The second goal is set to determine how hierarchical schemas are supported in current parsing components and how they can be implemented using tools available.

The last goal is set to find some tip-and-tricks how one can design efficient structural XML documents. In addition, there is a need to find some kind of a style guide for producing efficient XML.

1.3 The audience

The reader of this document is supposed to have some prior knowledge of XML and XML processing technologies. The knowledge of programming languages like Java or C++ is optional, but it is recommended to have knowledge either of them.

1.4 Environment

Throughout the study, we’ll use Java Development Kit 1.4 and runtime java engine 1.4. C++ environment is Microsoft Visual C++ 6.0 SP 6. Operating systems are Microsoft Windows 4.0/2000 and Sun OS Unix.

1.5 Definition of Terms

<p>| Term | Definition |</p>
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>DOM</td>
<td>Domain Object Model</td>
</tr>
<tr>
<td>DSSSL</td>
<td>Document Style Semantics and Specification Language: transformation and formatting language for SGML based documents</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition is SGML’s structural definition for the SGML document</td>
</tr>
<tr>
<td>Normalization</td>
<td>Normalization is a process where serialized characters in XML document are transformed to application specific characters (e.g &lt; ⇒ '&lt;') or white spaces are replaced or collapsed.</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group. An organization is responsible for standardization of several object oriented technologies: UML, MOF, XML, and Corba.</td>
</tr>
<tr>
<td>Parser</td>
<td>A tool that converts information to other form using specified grammar and lexical rules. The information can be either textual (like in Java-compiler) or binary information.</td>
</tr>
<tr>
<td>Processor</td>
<td>On the other hand, processor means the same as the parser, but XSLT has clear distinction between parser and processor. In XSLT, parser provides the source data to be transformed and it also parses the XSLT script, but the XSLT processor actually makes the transformation from data source to target as the script orders.</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Format</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML; de facto standard. SAX is event based API for XML parsing and validation. <a href="http://www.saxproject.org/">http://www.saxproject.org/</a> (06/2002)</td>
</tr>
<tr>
<td>Schema</td>
<td>Schema is an XML-based format for describing DTD. In addition, schema includes several enhancements like an element type definition other than just text (for example integer and date).</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium; develops recommendations, which are considered as standards, for the web. In addition, they develop guidelines, specifications, and reference tools.</td>
</tr>
<tr>
<td>White space collapse</td>
<td>Depending on the data type duplicated space characters may or may not be replaced with a single space character. Also trailing and leading spaces are removed.</td>
</tr>
<tr>
<td>White space replacement</td>
<td>Process where all white space characters (new-line, tab etc) are converted to a space character. This supersedes the white space collapse, and collapsing white spaces might be done if and only if replacement is done.</td>
</tr>
<tr>
<td>XDF</td>
<td>XML</td>
</tr>
<tr>
<td>XDR</td>
<td>XML Data Reduced An early language used to create a schema, which identifies the structure and constraints of a particular XML document. XML-Data Reduced refers to the subset of the XML-Data schema specification that was made available in MSXML 3.0 and later. It carries out the same basic tasks as DTD, but with more power and flexibility. Unlike DTD, which requires its own language and syntax, XML-Data Reduced uses XML syntax for its language. Unlike XSD, which has only recently been recommended as a standard, XML-Data Reduced was implemented and made available by Microsoft well ahead of the existence of XSD as a recommended standard by the W3C XML Schema Working Group.</td>
</tr>
<tr>
<td>XFragment</td>
<td>Partial XML data interchange mechanism</td>
</tr>
<tr>
<td>XInclude</td>
<td>File inclusion mechanism for XML.</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Model Interchange; an OMG specification integrating three industrial standards: MOF (OMG), UML (OMG), and XML (W3C). In practice, XMI is used as an XML-based interchange format for UML models (or other MOF-based data).</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language; W3C standard</td>
</tr>
<tr>
<td>XML schema; Lexical space</td>
<td>Before any validation, the second white space processing is done for the parsed space data. Now processing is targeted to all values defined in the schema. Depending on the data type the white space processing may or may not occur. All information that has gone through lexical processing is lexically correct for the data type in question.</td>
</tr>
<tr>
<td>XML schema; Parsed space</td>
<td>Parsed space contains information that actually reaches the application. Before information can be sent to the application, XML parser must transform data from serialization space to parsed space. This transformation includes Unicode conversions and white space etc. normalizations (in text nodes and attributes).</td>
</tr>
<tr>
<td>XML schema; Serialization</td>
<td>Serialization space defines that stream of bytes that the document stores. This is the payload of the document. [XMLSCH]</td>
</tr>
<tr>
<td>space</td>
<td></td>
</tr>
<tr>
<td>XML schema; Value space</td>
<td>This is the final step of value validation. Each data type has a value space, which determines the correctness of the value. For example, strings “3.1415” and “.31415E1” are the same if type is xs:float but they are different if type is xs:string.</td>
</tr>
<tr>
<td>XML: Valid</td>
<td>'Valid' document is &quot;well-formed&quot; and conforms to a specified set of production rules</td>
</tr>
<tr>
<td>XML: Well formed</td>
<td>'Well formed' means that there is exactly one root element, and every sub-element (and recursive sub-elements) have delimiting start- and end-tags, and that they are properly nested within each other. Well-formed document contains only one root element that contains all other elements. All tags has ending tag (&lt;tag&gt;) or if tag is empty tag it ends with /&gt; (&lt;emptyTag /&gt;)</td>
</tr>
<tr>
<td>XPath</td>
<td><a href="http://www.w3.org/TR/WD-xptr#XPath">http://www.w3.org/TR/WD-xptr#XPath</a></td>
</tr>
<tr>
<td>XPointer</td>
<td><a href="http://www.w3.org/TR/WD-xptr">http://www.w3.org/TR/WD-xptr</a></td>
</tr>
<tr>
<td>XSL</td>
<td>Extended Style Language. Extended version of CSS</td>
</tr>
<tr>
<td>XSL-FO</td>
<td>XSL Formatting objects. Formatting objects that handles the correct presentation of information in some printing media (display, printer, PDF-file etc.).</td>
</tr>
<tr>
<td>XSLT</td>
<td>XSL Translation</td>
</tr>
</tbody>
</table>
2. BACKGROUND TECHNOLOGIES

This chapter collects background information for several technologies that are involved to the subject in hand but are not key interests for the study.

2.1 Domain Object Model (DOM)

Domain Object Model presents the structure of a XML document as a family tree. DOM has one root element (the grand parent who has no parent) and all other elements are descendants of it.

Element has a single parent and element can have ancestors, child elements, and siblings (descendants of the element’s parent).

The Example 2-1 shows a simple XML document called The Book. The Book contains three chapters and each chapter contains zero or more paragraphs. Each ‘Chapter’ element has a chapter sequence number. The same document is depicted graphically in Figure 2-1.

```
<?xml version="1.0"?>
<Book>
  <Chapter num="1">
    <Para>Text 1</Para>
  </Chapter>
  <Chapter num="2">
    <Para>Text 2</Para>
    <Chapter num="3">
      <Para>Text 3</Para>
      <Para>Text 4</Para>
      <Para>Text 5</Para>
    </Chapter>
  </Chapter>
</Book>
```

Example 2-1: theBook.xml

Figure 2-1 also shows basic relationships in a DOM tree. The root of the tree is the ‘Book’ element. It has three child nodes and eight descendants.

The three ‘Chapter’ nodes are siblings to each other, so are ‘Para’ nodes. Each ‘Para’ node has a ‘Chapter’ node as a parent, while the ‘Book’ node is the parent for all ‘Chapter’ nodes. Ancestors for the ‘Para’ {Node 3.3} node are ‘Chapter’ {Node 3} and ‘Book’ {Root}. The ‘Chapter’ {Node 1} node has two descendants: ‘Para’ {Node 1.1} and ‘Para’ {Node1.2}.
There are also two other relationships in a DOM tree: predecessor and successor. A predecessor node is the rightmost child of preceding sibling of the context node. A successor node is the leftmost child of the succeeding sibling of the context node. For example, if the context node is ‘Chapter’ {Node 2}, then the predecessor is ‘Para’ {Node 1.2} and the successor is the ‘Para’ {Node 3.1}.

Current DOM implementations know only predecessor and successor sets, which contain all applicable nodes. For example, Predecessors for ‘Chapter’ {Node 2} are ‘Book’ {Root}, ‘Chapter’ {Node 1}, ‘Para’ {Node 1.1}, and ‘Para’ {Node 1.2}. Successors for the same node are ‘Book’ {Root}, ‘Chapter’ {Node 3}, and ‘Para’ {Node 3.1}, ‘Para’ {Node 3.2}, and ‘Para’ {Node 3.3}.

In addition to structural nodes described above, the DOM tree contains also other nodes, which are fully described in [XML 1.0]. For example, there are two non-structural content nodes: a text node and an attribute node. The text node is used to carry content data and attribute is used to carry parameters and sometimes content data.

Presently implementations give a little different definition for the DOM tree root. They have one additional element representing the root. The root has one structural child element (in our case it would be the ‘Book’), but it also could have additional elements called processing instructions. For simplicity, we dropped that additional root element to clarify the principle of DOM.

### 2.2 Traversing and searching (XPath)

The XPath is one of the most important background technologies and is applied widely. Almost every other technology uses XPath as a foundation or a tool.

#### 2.2.1 Description

The primary purpose of XPath is to address various parts in an XML document and especially in its DOM representation. It also provides tools for manipulating strings, numbers, and Booleans. There are also tools for searching and matching XML element nodes.

XPath has two important concepts: a location path and an expression. The location path is a way to show direct path from some node to some other node. Location path evaluation uses expression to have some flexibility and dynamics in it. The value of evaluated expression can be a node-set, a Boolean value, a string, or a number. For each expression, there is a certain evaluation context, which defines the intent of the expression.

The context has a node (called context node), a pair containing context nodes position and the size of the context, a set of variables and library functions visible to the context, and namespace declarations. XPath declares a standard function set, but an application using XPath language can extend the function set.

**Location path**

Location path is the most important construct in the XPath language. Location path is an expression that identifies a node or a group of nodes. Location path can identify a node element, an attribute of node element, or the text of a node element. Location path is either relative or absolute (see Example 2-2).
Example 2-2: Location path examples

<table>
<thead>
<tr>
<th>Location path examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Book/Chapter[1]</td>
<td>Absolute path to Chapter 1</td>
</tr>
<tr>
<td>/child::Book/child::Chapter[1]</td>
<td>Same as above (non-abbreviated)</td>
</tr>
<tr>
<td>//Chapter[1]/attribute:num</td>
<td>Selects num attribute from the first Chapter node</td>
</tr>
<tr>
<td>child::*/@num</td>
<td>Same as above (abbreviated)</td>
</tr>
<tr>
<td>/Book</td>
<td>Selects all child nodes of the context node</td>
</tr>
<tr>
<td></td>
<td>Selects the root element</td>
</tr>
<tr>
<td></td>
<td>Selects the child Book of the root element</td>
</tr>
</tbody>
</table>

In Example 2-2, we can see how location path contains path separators ‘/’ and the name of the node, which is along the path. When two path separators are combined, we shall have a path wildcard: all paths containing given node will do. For example, if we have a document having structures `/book/chapter/paragraph` and `/book/chapter/verse`, we can select all paragraphs and verses using location path `//chapter/*`.

Context node is the current node that is used as a base of the relative location path execution. Absolute paths are executed without the knowledge about the context node.

Location path is constructed from node names, axes or both. The node name in DOM tree is the same as the name of an XML element in the XML document. Axes are names for relationships between nodes in the tree (see Table 2-1).

Table 2-1: Axes

<table>
<thead>
<tr>
<th>Axes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>Axis contains the children of the context node</td>
</tr>
<tr>
<td>descendant</td>
<td>Axis contains the descendants of the context node; thus the descendant axis never contains attribute or namespace nodes</td>
</tr>
<tr>
<td>parent</td>
<td>Axis contains the parent of the context node, if there is one (root node)</td>
</tr>
<tr>
<td>ancestor</td>
<td>Axis contains the ancestors of the context node; thus, the ancestor axis will always include the root node, unless the context node is the root node</td>
</tr>
<tr>
<td>Following-sibling</td>
<td>Axis contains all the following siblings of the context node; if the context node is an attribute node or namespace node, the following-sibling axis is empty</td>
</tr>
<tr>
<td>Preceding-sibling</td>
<td>Axis contains all the preceding siblings of the context node; if the context node is an attribute node or namespace node, the preceding-sibling axis is empty</td>
</tr>
<tr>
<td>following</td>
<td>Axis contains all nodes in the same document as the context node that are after the context node in document order, excluding any descendants and excluding attribute nodes and namespace nodes</td>
</tr>
<tr>
<td>preceding</td>
<td>Axis contains all nodes in the same document as the context node that are before the context node in document order, excluding any ancestors and excluding attribute nodes and namespace nodes</td>
</tr>
<tr>
<td>attribute</td>
<td>Axis contains the attributes of the context node; the axis will be empty unless the context node is an element</td>
</tr>
<tr>
<td>namespace</td>
<td>Axis contains the namespace nodes of the context node; the axis will be empty unless the context node is an element</td>
</tr>
<tr>
<td>self</td>
<td>Axis contains just the context node itself</td>
</tr>
<tr>
<td>descendant-or-self</td>
<td>Axis contains the context node and the descendants of the context node</td>
</tr>
<tr>
<td>ancestor-or-self</td>
<td>Axis contains the context node and the ancestors of the context node; thus, the ancestor axis will always include the root node</td>
</tr>
</tbody>
</table>

Node test is a concept where a principal node type is selected for qualifying the node. Principal node type can be either element, namespace, or attribute. For example, the principal node type is element if ‘child::*’ or ‘/book/para::para’ is used, while it is attribute if ‘attribute::*’ is used.
Predicates are filtering qualifiers that filter out unwanted nodes. For example, the fourth paragraph in the book would be called '/book/paras::para[4]'. Predicate is always expression surrounded by square brackets.

Location step is partial location path used to select subset of nodes. Location step is constructed from axis, node test and predicate.

**Expressions**

An expression gives additional processing capability to the XPath. The XPath expression can contain variable references, string literals, numbers, function calls, paths, and node sets. These are bound together with arithmetic, logical, Boolean, and relational operators.

There is a small set of functions available for the user. They can be divided in three categories: node/node set, string and numeric functions. Node/node set functions mainly returns node’s position information or textual information that is located into the node. String functions provide basic set of string manipulation functions like substring, catenation and find. In addition, there are several numeric function, Boolean operations and basic conversion functions (ceiling, floor, etc.).

**Data Model**

XPath assumes that the XML document is modeled as a tree. This is not implementation constraint but conceptual. There are seven types of nodes in that tree:

1. Root nodes
2. Element nodes
3. Text nodes
4. Attribute nodes
5. Namespace nodes
6. Processing instruction nodes
7. Comment nodes

There is only one root node in XML document. All the other nodes are its children. The root node has only one element node (the document element). In addition, it can have comment and processing instructions but nothing else.

Element nodes construct the structure of the XML document. Text and attribute nodes contain the data content. The namespace node contains the URI for the namespace that is defined.

**2.2.2 Tool support**

Most XML parsers and XSLT translators are able to execute XPath expressions. In addition, XPath is used in Apache’s Xindice XML database as a query language.

**2.3 Binding resources together (XLink)**

**2.3.1 Description**

XLink is a resource pointer that shows to the where the resource is locate and it can be retrieved. XLink provides means to bind two or more resources together with unidirectional links or even more complex linking structures. A simplest example of XLink is the hyperlink in HTML namely the A element. It provides a way to travel from one place to another, but you are not able to tell from where you have come there.
XLink provides more complex ways to link resources than is provided in HTML. One can define a keyword where the user can select what kind of meaning he is looking for. For example with keyword ‘car’ one can make a selection of resources like ‘Manufacturers’, ‘Repairs shops’, ‘Car Dealers’, ‘Parts’, and ‘Museum’. From that selection list the user can select what he wants to know.

2.3.2 Tool support

There are some partial implementations that conforms at least simple links and in some cases complex link structures.

2.4 Pointing resources (XPointer)

2.4.1 Description

XPointer expresses an XML resource fragment identifier. With XPointer, it is possible to address internal structures of the XML document or external parsed entities (other XML documents). XPointer is an extension to XPath language, which embeds XPath expression to URI. XPointer allows the following:

- Addressing points and ranges as well as whole nodes
- Locating information by string matching
- Using addressing expressions in URI references as fragment identifiers

As its name suggests, the XPointer application handles collections of resource locations. With XPointer, it is possible to locate start and/or end tags of an element, partial data inside an element, etc. XPointer differs from XLink in one way. XLink is capable of pointing one single resource entity whereas XPointer can point to a collection of resource locations. XLink can be used to travel from document to another while XPointer can be used to locate resources available for use in applications.

As an example, the XPointer expression http://www.foobar.org/Books/theBook.xml#/1/3/2 points to /Books/Chapter[3]/Para[2] element, which can be expressed also as http://www.foobar.org/Books/theBook.xml#xpointer(/*[1]/*[3]/*[2]).

2.5 Deployment modularity (XInclude)

2.5.1 Description

XInclude provides generic mechanism to facilitate modularity, where user can merge larger document from smaller document parts. Even though XInclude has similarities with XLink and XFragment, it has its own role. The XInclude operates at parser level whereas XLink and XFragment operate at application level. When parser sees XInclude element in the XML-stream, the parser replaces the element with the content of the XML-document indicated by the element. Thus, the inclusion is not visible to the application.

XInclude introduces two commands: include and fallback. Include element introduces the document that is supposed to include into the original document and fallback is used to report for error during the inclusion.

Example 2-3: XInclude example

<table>
<thead>
<tr>
<th>Original</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2-3 shows simple inclusion examples with and without error. The first example shows what happens when the original document tries to include a file that does not exist and what is the result for it. The second example shows successful inclusion.

### 2.6 Partial Document (XFragment)

#### 2.6.1 Description

Because the XML document can easily grow very large and the only need is to edit a small piece of it at the time, the W3C has made XML application for interchanging document fragments. The recommendation is called XFragment [XFragment]. XFragment is a mechanism that gives a possibility to split a large document into smaller pieces, which should be easier for client to handle.

XFragment has two parts. The first part is Fragment Context Specification (FCS) and the second part is the fragment body itself. The FCS describes the full context (location and path) where the fragment body can be found. XPointer is used as a fragment resource locator. Depending on locating style (i.e. how precise it is) there may be a need to describe also all siblings of the fragment’s root element. If the locator is precise as it is in Example 2-4 the FCS should only contain direct descendant tree from root element (Book) to the fragment body root element (Chapter).

The fragment body is an obligatory part of document fragment. It can be delivered along the FCS but it can also be retrieved separately from it. The fragment body contains the element described in FCS and all its descendants.
Example 2-4: Fragment package with FCS and fragment body

```xml
<?xml version="1.0"?>
<p:package xmlns:p="http://www.w3c.org/2001/02/xml-package">
  <p:fcs xmlns:f="http://www.w3c.org/2001/02/xml-fragment"
        sourcelocn="http://www.foobar.org/Books/theBook.xml#root().child(3,Chapter)">
    <Book>
      <Chapter/>
      <p:fragbody/>
    </Book>
  </p:fcs>
  <p:body>
    <Chapter num="3">
      <Para>Text 3</Para>
      <Para>Text 4</Para>
      <Para>Text 5</Para>
    </Chapter>
  </p:body>
</p:package>
```

Here the Example 2-4 shows document fragment package containing both FCS and the fragment body. There is no need to distribute the body part along FCS. The application can retrieve the body using `sourcelocn` attribute. To construct the original document structure, the element above the `fragbody` element shall be replaced with the fragment body.

### 2.7 Grammars for document hierarchy

#### 2.7.1 General

The XML core specification [XML10] defines rules, which determine a syntactically correct XML document, namely a well-formed document. It will not tell if each element in the document is correct. It rather tells that all elements in the document look like correct elements.

To determine if the element is correct, a set of rules defining the semantics of the document must be created. The rule set is called grammar. This grammar in XML is called a schema or a document type declaration.

There are two W3C recommendations to declare semantics. The older one is called DTD (Document Type Definition) and the newer is called XML Schema. The DTD comes from SGML world from where the XML itself is originated. One should not call DTD as document type declaration because it is a different concept. Document type declaration is an element in XML document that defines the DTD used for validating that XML document (`<!DOCTYPE>` element).

#### 2.7.2 Document Type Definition (DTD)

Document Type Definition declares a class of documents. DTD is not syntactically XML. DTD has several flaws in it. The most notable flaw is that it doesn’t have any other data types except the string type. In addition, it doesn’t suit well today’s technologies (object-orientation etc.) because it has its roots in 70’s. To fix these flaws the XML community began to seek alternatives for DTD. Today there are several alternatives for DTD, but we focus our attention to XML Schema (the most popular one) and shortly review other alternatives.
In Example 2-5, there is a simple XML document containing inline DTD declaration. The example shows the way to use inline document definitions. The second line introduces document type definition ‘greetings’ and the third line shows the document type definition (DTD) of the document.

### 2.7.3 XML Schema

The W3C XML Schema Definition Language is an XML language for describing and constraining the content of XML documents. W3C XML Schema is a W3C Recommendation.

In Example 2-6, a similar document as in Example 2-5 is shown. At this time, it is validated with an external XML Schema.

XML Schema has several strengths comparing to DTD. Schema supports a wide variety of basic data types and compound data types (structure, union, enumeration etc.). It supports some object-oriented features like inheritance and abstract type. There is the support for namespaces whose absence in DTD is the most criticized feature. XML Schema is itself XML-based.

### 2.7.4 Other schema descriptions

In Table 2-2 there are several other schema languages that are more or less predecessors of XML Schema and DTD.
### Table 2-2: Other Schema Languages

<table>
<thead>
<tr>
<th>Schema Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Data</td>
<td>This is predecessor of XML Schema. Also known as XDR. The XML Schema definition language (XSD) is the current World Wide Web Consortium (W3C) specification for XML schemas. XML-Data Reduced (XDR) schema is an interim schema language offered to developers while the W3C worked on a draft implementation.</td>
</tr>
<tr>
<td>Schema for Object-Oriented XML (SOX)</td>
<td>This is predecessor of XML Schema. SOX was developed while the XML Schema definition was under construction.</td>
</tr>
<tr>
<td>Document Content Description (DCD)</td>
<td>XML based type declaration language that is based on RDF Model and Syntax Specification. It has several simplifications.</td>
</tr>
<tr>
<td>Document Definition Markup Language (DDML)</td>
<td>This is predecessor of XML Schema. DDML is DTD type definition language except the format was XML.</td>
</tr>
<tr>
<td>Schematron</td>
<td>The Schematron is a simple and powerful Structural Schema Language. It differs in basic concept from other schema languages in that it not based on grammars but on finding tree patterns in the parsed document. This approach allows many kinds of structures to be represented which are inconvenient and difficult in grammar-based schema languages. Schematron can be useful in conjunction with many grammar-based structure-validation languages: DTDs, XML Schemas etc.</td>
</tr>
<tr>
<td>Data types for DTD (DT4DTD)</td>
<td>DTD extension adding about 50 data type to DTD language. Data types are compatible with XML Schema data types.</td>
</tr>
<tr>
<td>Document Structure Description (DSD)</td>
<td>DSD is simpler and more expressive than XML Schema is (1999).</td>
</tr>
<tr>
<td>Regular Language Description for XML Next Generation (RELAX NG)</td>
<td>An alternative for XML Schema recommendation from OASIS.</td>
</tr>
</tbody>
</table>

The list is not complete but it contains most commonly referenced Schema languages. Serious alternatives for XML Schema are Schematron and RELAX NG.
3. XML PROCESSING TECHNOLOGIES

3.1 General

In this report, XML processing is divided in four categories: scanning, parsing, validation and translation.

Scanning is an operation that divides input data in small pieces using lexical rules and delivers pieces to parser. The parser is responsible for checking that the stream of lexical pieces forms something rational. The parser uses syntax rules to determine if the input is logical and if so, the input can be called as well formed.

Well-formed XML data has all its elements in correct order. It doesn’t care if one element should or should not be in the place it is found or is it even a part of the document structure. It just cares if individual element looks right and has its ending tag in a correct place, and there is no interlacing between elements. The parser just tests if the document is well formed.

Validation is a process where the well-formed XML document is validated against document’s type definition. These type definitions can be given either Document Type Definition (DTD) or in various schema formats. The type definition tells to the validating processor what kind of document is considered as valid document. Type definition tells the structure of the document class.

Translation processor converts a document to another form using given translation rules. In principle, the document not necessarily is required to be XML-based. There is a recommendation for translations called XSLT, but there can also be other proprietary solutions.

3.2 Parsing XML

3.2.1 General

There are two common API’s to parse XML documents. The first technique is called SAX and it is based on events that occur after successful determination of XML elements. The second technique is called DOM (Domain Object Model) and it creates a tree like data structure into the memory. These two API’s are de Facto standards in the industry. SAX is developed by David Meggingson [SAX] with the help of xml-dev mailing list and DOM is W3C Recommendation, which is as close to standard as anything can be [DOM]. In addition, there are techniques like ‘Event Handling’, which is partial combination of both SAX and DOM features.

Today there are mainly three ways to process XML data: SAX, DOM or a proprietary grammar based parsing and validation method. In addition, there are grammar based API’s that transforms any type of data to XML data using XML based grammar.

3.2.2 Scanning

Scanning is a vital part of parsing process. It prepares the input data to smaller grammatical entities. The input data may or may not be textual data as it is in XML or it can be binary data as it is in RTF. In this chapter, a character is used to describe the smallest data unit in the data (byte, Unicode character etc.).
Scanner walks through the input data character by character. Scanner has a set of scanning rules, which determines the form of grammatical entities. In principle, the scanner doesn’t care about the order of those entities. That is the responsibility of parser and its grammatical rules.

In this study, we define two types of scanners important to us. The other uses XML scanner rules defined by [XML10] and the other uses customized scanner rules. Difference between these two is that the XML scanner requires XML data whereas the custom scanner can process any data.

The basic assumption in this study is that both types of scanners produce XML information as a result.

3.2.3 Event based SAX (Simple API for XML)

SAX is an event based API. There are three categories of events: content, document type and error. Content event are responses to XML-document parsing process. The content interface (ContentHandler) offers events from starting and ending the document, each element, processing instructions, the element content etc.

SAX is a stream-based parser. There is a XML stream of elements and each element causes beginning and ending events. When SAX is used, the application writer must take care of the state of the XML stream. Even though SAX checks if the document is well formed and even valid the application must be aware of the current point in the document. After an event is handled, there is no possibility to go back to that element that caused the event.

Example 3-1: SAX events while parsing document

<table>
<thead>
<tr>
<th>Original XML document</th>
<th>SAX events</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;?xml version=&quot;1.0&quot;?&gt;</td>
<td>StartDoc</td>
<td>Book</td>
</tr>
<tr>
<td>&lt;Book&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Chapter num=&quot;1&quot;&gt;&lt;/Chapter&gt;</td>
<td>startElement:</td>
<td>Chapter [&lt;num, &quot;1&quot;]/]</td>
</tr>
<tr>
<td>&lt;Para&gt;Title1&lt;/Para&gt;</td>
<td>startElement:</td>
<td>Para</td>
</tr>
<tr>
<td>&lt;Para&gt;Title2&lt;/Para&gt;</td>
<td>characters:</td>
<td>Text 1</td>
</tr>
<tr>
<td>&lt;/Chapter&gt;</td>
<td>endElement:</td>
<td>Para</td>
</tr>
<tr>
<td>&lt;Chapter num=&quot;2&quot;&gt;&lt;/Chapter&gt;</td>
<td>startElement:</td>
<td>Chapter [&lt;num, &quot;2&quot;]/]</td>
</tr>
<tr>
<td>&lt;Para&gt;Title3&lt;/Para&gt;</td>
<td>characters:</td>
<td>Text 2</td>
</tr>
<tr>
<td>&lt;Para&gt;Title4&lt;/Para&gt;</td>
<td>endElement:</td>
<td>Para</td>
</tr>
<tr>
<td>&lt;Para&gt;Title5&lt;/Para&gt;</td>
<td>startElement:</td>
<td>Chapter [&lt;num, &quot;3&quot;]/]</td>
</tr>
<tr>
<td>&lt;/Chapter&gt;</td>
<td>endElement:</td>
<td>Chapter</td>
</tr>
<tr>
<td></td>
<td>startElement:</td>
<td>Para</td>
</tr>
<tr>
<td></td>
<td>characters:</td>
<td>Text 3</td>
</tr>
<tr>
<td></td>
<td>endElement:</td>
<td>Para</td>
</tr>
<tr>
<td></td>
<td>startElement:</td>
<td>Para</td>
</tr>
<tr>
<td></td>
<td>characters:</td>
<td>Text 4</td>
</tr>
<tr>
<td></td>
<td>endElement:</td>
<td>Para</td>
</tr>
<tr>
<td></td>
<td>startElement:</td>
<td>Para</td>
</tr>
<tr>
<td></td>
<td>characters:</td>
<td>Text 5</td>
</tr>
<tr>
<td></td>
<td>endElement:</td>
<td>Para</td>
</tr>
<tr>
<td></td>
<td>endElement:</td>
<td>Chapter</td>
</tr>
<tr>
<td></td>
<td>endElement:</td>
<td>Book</td>
</tr>
</tbody>
</table>

The SAX event flow is depicted in Example 3-1. The beginning of the element event also carries a list of all attributes and namespace information.

3.2.4 Tree based DOM (Document Object Model)

DOM parser creates a tree of nodes based on the given document. The parser validates each node at the end of the element. DOM Parser creates a memory element from each XML element and inserts it into the tree like structure. After reading XML document the whole document is in the memory. When this tree is in the memory, it can be easily traversed from
any node to another node, nodes can be added or removed, the content can be manipulated, and in the end, the document can be written back to the stream.

3.2.5 Comparing SAX and DOM

When comparing these different parsing techniques, it is not sensible to say that one technique is better than the other is. They are all reasonable tools for different tasks.

SAX is reasonably fast for many tasks like searching an element or fetching small portion of data from a big document. It consumes much less memory than DOM while processing the XML document. For processing XML document with SAX, one must always restart from beginning and continue until the end of the document and thus there is no possibility to traverse backward.

SAX is well suited for handling large documents. SAX allows interrupting the parsing process at any time thus making it good for searching and fetching. SAX is good for retrieving small amounts of read-only data that remains unchanged. SAX is well suited for changing document structure (filter) for example one can make a DOM tree from business objects instead of DOM node objects: filter makes business objects instead of making DOM node objects (class Node -> class Book). SAX is good if one cannot afford the DOM overhead. For example, one can have 5 MB of data where only 100 bytes are relevant. Thus, it is wise decision to use SAX instead of DOM.

DOM suits well for relatively small amount of read-write data. DOM is always stored in memory whereas SAX is not. This memory-based information is well suited for random access operations. When the data is in memory, one can implement rather complex searches through the data. Because it is easy to modify this in memory data, there is a need to save and load this data. This property is built in the DOM. DOM works better with XSL than SAX. This is particularly true if one uses XPath language. The XPath is tree based by nature and it requires the possibility to traverse back and forth in the tree.

DOM is very good is the data must be traversed or the structure must be available all the time. That consumes memory. To improve processing speed some processors has so called ‘Deferred processing’ that doesn’t process all child elements of the element but leaves them in textual form. When deferred element is accessed, the processor will parse that node and its child elements. That is, elements are processed only when needed. This deferred mode will grow memory consumption a lot [XML&JAVA; page 120-128].

3.3 Validation

Validation is an additional task that the parser has to do if it wants to assure the validity of the document. Validation confirms that recently parsed element is placed correctly and it has a right amount of correctly formed attributes and child elements.

Validation is a very time-consuming task. That is why one must consider carefully when the validation should take place. For example, if one can rely on a correct transaction between a producer and a consumer, it may be better if the consumer doesn’t validate the document.
### 3.4 Transformations

#### 3.4.1 General

XML itself is a data description format that has no knowledge of its visual appearance. Transformations are a way to give visual presentation for plain data. This is the original meaning of Cascading Style Sheets (CSS). It has means to change colors, fonts, margins and even positions of individual elements. In principle, there is no actual way to select the data that should be presented on the rendering media (browser, printer etc.). One can make an element as invisible, but there is no way to exclude any element from the result.

To tackle some shortcomings with CSS, W3C created a new style sheet language. Extensible Stylesheet Language (XSL) contains three parts: XSLT, XPath and XSL-FO. The XSLT (XSL Transformations) provides a language for transforming documents. The XPath provides tools to access or refer various parts in the document. XSL-FO (XSL formatting objects) specifies the formatting semantics (similar to CSS). As one can assume, XSL is XML application and thus it has XML syntax.

#### 3.4.2 Cascading Style Sheets, level 2 (CSS2)

CSS is a tool to give a visual appearance for a raw data. It contains definitions for colors, fonts, margins etc. to be used in displaying or printing. There are also some definitions for aural translations.

CSS has a format of its own. It can be used with XML, but it requires that the XML document have a reasonably linear structure that can be displayed without extensive manipulation.

**Example 3-2: Translation with CSS**

<table>
<thead>
<tr>
<th>Document source</th>
<th>Cascading Style Sheet</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;?xml version='1.0'?&gt;</code></td>
<td><code>Book {</code></td>
</tr>
<tr>
<td><code>&lt;?xml:sstylesheet</code></td>
<td><code>display: block;</code></td>
</tr>
<tr>
<td><code>type='text/css'</code></td>
<td><code>background: khaki;</code></td>
</tr>
<tr>
<td><code>href='books.css'?&gt;</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>&lt;Book&gt;</code></td>
<td><code>Chapter {</code></td>
</tr>
<tr>
<td></td>
<td><code>display: block</code></td>
</tr>
<tr>
<td><code>&lt;Chapter num='1'&gt;</code></td>
<td><code>font-size: 0.6em;</code></td>
</tr>
<tr>
<td><code>&lt;Para&gt;</code>Text 1&lt;/Para&gt;`</td>
<td><code>background: skyblue;</code></td>
</tr>
<tr>
<td><code>&lt;Para&gt;</code>Text 2&lt;/Para&gt;`</td>
<td><code>height: 6pt;</code></td>
</tr>
<tr>
<td><code>&lt;/Chapter&gt;</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>&lt;Chapter num='2'/&gt;</code></td>
<td><code>Para {</code></td>
</tr>
<tr>
<td><code>&lt;Chapter num='3'&gt;</code></td>
<td><code>Display: block;</code></td>
</tr>
<tr>
<td></td>
<td><code>font-style: italic;</code></td>
</tr>
<tr>
<td></td>
<td><code>color: black;</code></td>
</tr>
<tr>
<td></td>
<td><code>background: gold;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>Book, Chapter, Para {</code></td>
</tr>
<tr>
<td></td>
<td><code>margin: 0.5em;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>
In Example 3-2, there is an XML document translated with CSS to a visual form. There is a style definition for all elements in the document. The ‘Chapter’ element has minimum height to make the empty chapter number 2 visible.

### 3.4.3 XSL / XSLT

XSL formatting objects contains powerful tools to describe the display layout of the document. There are tools to define page layouts (header, body, footer), text formatting etc.

The XSLT and the XPath makes a tight pair. XSLT defines a language, which uses XPath for queries to the document structure. XSLT can be used to transform one document to other form for example from XML to HTML as in Example 3-3.
Example 3-3: Translation with XSLT

<table>
<thead>
<tr>
<th>Document source</th>
<th>XSLT source</th>
</tr>
</thead>
</table>
| ```xml version='1.0'?>
 <?xml:stylesheet
 type='text/xsl'
 href='books.xsl'?>
 <Book>
  <Chapter num='1'>
   <Para>Text 1</Para>
   <Para>Text 2</Para>
  </Chapter>
  <Chapter num='2'/>
  <Chapter num='3'>
   <Para>Text 3</Para>
   <Para>Text 4</Para>
   <Para>Text 5</Para>
  </Chapter>
 </Book>``` |
| ```<?xml version='1.0'?>
 <xsl:stylesheet version='1.0'
 xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
 <xsl:output method='html' indent='yes'
 doctype-public=''-//W#C//DTD HTML 4.0 Final//EN'/>``` |

<table>
<thead>
<tr>
<th>CSS source</th>
</tr>
</thead>
</table>
| div.Book {
   font-size: 1.2em;
   background: khaki;
   margin: 0.5em
} div.Chapter {
   font-size: 1.0em;
   background: skyblue;
   margin: 0.5em
} div.Para {
   font-size: 0.8em;
   font-style: italic;
   color: black;
   background: gold;
   margin: 0.5em
} |

<table>
<thead>
<tr>
<th>Translation result:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Book" /></td>
</tr>
</tbody>
</table>

In Example 3-3, there is the same XML document source as in Example 3-2 but now it’s transformed using XSLT. To give visual appearance, CSS is used instead of XML-FO to make the example a little bit shorter. All texts seen in the translation result are taken out from the XML document and formatting depends on rendered element. That was not possible in pure CSS version.
4. XML SCHEMA (HIERARCHICAL SCHEMAS)

This chapter is mainly based on [XMLSCH].

4.1 Basic elements

The idea behind the schema is to make definition for data entity. This entity has a unique structure constructed from various data types. Data types can be simple basic data types or they can be more complex data types, which are combined from simple data types. There are also mechanisms for schema deployment and reuse, document class or type abstraction and inheritance like in object-oriented world, and data type definitions through regular expressions.

4.2 Simple data types

Simple data types are predefined in XML Schema Part 0 [XSD11]. Simple types are string, numeric, date and time, list type, and anySimpleType. All simple types are derived from anySimpleType using derivation by restriction. More comprehensive description can be found from [XMLSCH] or [XSD11].

Strings can be divided into four groups (figure 4-1); no white space handling (string), white space replacement (normalizedString), white space collapse (token etc.), and special semantics.

![Figure 4-1; string types [XMLSCH p.24]](image)

The string type has no white space handling i.e. the string keeps its form during the processing. Normalized strings have only white space replacement in use, which means that every white space character is replaced with SPACE (#x20) character. Tokens have white space collapsing, which replaces all other white spaces with SPACE character and then removes all consecutive SPACEs with single SPACE character. Binary data can be inserted into the XML document using encoding tags hexBinary and base64Binary.

Numerical data types (Figure 4-2) can be divided in three categories: decimal, floating point, and Boolean. Decimal numbers can be arbitrarily long, but machine depended long, int, short, and byte with their unsigned counterparts are restricted to their natural microprocessor-friendly lengths. The decimal number can have arbitrarily long fractional part, but there are
also real floating-point types: float and double. Floating-point numbers are IEEE simple (32 bits) and double (64 bits) precision types. Boolean is a simple truth-value type having values true and false (or 1 and 0).

![Diagram of Numerical Types](image)

**Figure 4-2; Numerical types [XMLSCH p:31]**

XML schema has several date and time types including absolute date and time, durations, and recurring times and periods. XML Schema data and time data types are a subset of the ISO 8601 standard.

![Diagram of Date and Time Types](image)

**Figure 4-3; Date and time types**

XML Schema has three predefined list types: NMTOKENS, IDREFS, and ENTITIES. They represent a whitespace separated lists of NMTOKEN (a set of characters without spaces), IDREF (reference to ID that must be found from document), and ENTITY (unparsed non-XML content: &LT; is <). There is also a possibility to make user defined lists from any simple data type using for example complexType and sequence element together.

The last simple type is anySimpleType, which is a wildcard type. When a type is a wildcard, it accepts anything and it doesn’t add any lexical constraints to the value. Its usage should be limited to a type derivation and it cannot be used as a basis of the user defined simple data type. It is a void data type.

### 4.3 Object orientation

#### 4.3.1 Inheritance

Although simple types offers wide variety of data types, definitions of simple types may be too weak or too simple to express the required data correctly. That is why XML Schema has tools to define restricted, extended or structured data types. Restriction (<restriction>-element) removes information from data type (integer -> negativeInteger). Extension introduces new
information to existing type. Restriction and extension are similar to inheritance or generalization between two classes. Example 4-1 shows some examples of restriction and extension.

**Example 4-1: Extending or restricting types**

```xml
<?xml version="1.0" ?>
<xs:schema
xmlns:xs="http://www.w3c.org/2001/XMLSchema">
  <xs:complexType name="personName">
    <xs:sequence>
      <xs:element name="title" minOccurs="0"/>
      <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="surname"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="extendedName">
    <xs:complexContent>
      <xs:extension base="personName">
        <xs:sequence>
          <xs:element name="generation" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="addressee" type="extendedName"/>
</xs:schema>

<addressee>
  <forename>Albert</forename>
  <forename>Arnold</forename>
  <surname>Gore</surname>
  <generation>Jr</generation>
</addressee>
```

Structural information for data can be constructed using `complexType` element. More information about object oriented features can be found from [XSD11] and [XMLSCH].

### 4.3.2 Abstract document type

Abstract Schema definition is an answer to the question ‘how can the base type instantiation be prohibited’. A type or an element is too general for public use and there should not be any instance of that item.

XML Schema provides a mechanism to define abstract types or elements using `abstract` attribute in type or element definition. If some item is defined as abstract, it is not possible to instantiate a document a document containing abstract item. One must always bind the abstraction to real element (Example 4-2). This again clearly resembles the notation of abstract classes in object-oriented programming languages.
Example 4-2: Abstract schema

1)  

```xml
<schema xmlns=http://www.w3.org/2001/XMLSchema
targetNamespace="http://example.com/schema"
xmlns:target="http://example.com/schema">
  <complexType name="Vehicle" abstract="true"/>
  <complexType name="Car">
    <complexContent>
      <extension base="target:Vehicle"/>
    </complexContent>
  </complexType>
  <complexType name="Plane">
    <complexContent>
      <extension base="target:Vehicle"/>
    </complexContent>
  </complexType>
  <element name="transport" type="target:Vehicle"/>
</schema>
```

2)  

```xml
<transport xmlns="http://example.com/schema"/>
```

3)  

```xml
<transport xmlns="http://example.com/schema"
xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:type="Car"/>
```

4)  

```xml
<transport xmlns="http://example.com/schema"
xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:type="Plane"/>
```

The example above defines three types and one element (Example 4-2 section 1). ‘Vehicle’ type represents abstract and general vehicle type (class). Two other types, ‘Car’ and ‘Plane’, are specializations of the ‘Vehicle’ type. Finally, the schema defines the element ‘transport’ for a document that can contain element of ‘Vehicle’ type. Section 2 shows illegal use of ‘transport’ element. Because ‘transport’ element is abstract, the document cannot instantiate the element without first specifying the actual type. Sections 3 and 4 shows the correct use to instantiate the ‘transport’ element. In section 3, the ‘transport’ element is of type ‘Car’. In section 4, the ‘transport’ element is of type ‘Plane’ type.

Assume that there are two documents one using ‘transport::Car’ and the other one using ‘transport::Plane’. They both can be roughly validated with a document schema that has general (abstract) definition ‘transport::Vehicle’. These documents are both valid on that abstract level even though the ‘Plane’ and the ‘Car’ can differ a lot.
5. XML STYLES AND STYLE GUIDELINES

This is sort collection of XML guidelines and it is by no means comprehensive.

5.1 XML style guidelines

1. Avoid whitespaces whenever possible, because they introduces unnecessary overhead while processing the document.
2. You should think seriously if you are using attributes to carry data instead of elements. When you use attributes, you may have better performing application in memory wise and in speed wise.
3. Use meaningful element names instead of arbitrary or generated names. XML is meant to be human readable even though machines processed it.

5.2 XSLT

1. Don’t use indent-attribute with value ‘no’ because it makes files larger. Use indention only if it is really needed.
2. Use XSLT when ever possible. It is a lot faster than DOM when you should find something from the document.

5.3 XML Schema

1. Item should be defined to type instead of an element. This makes the schema more reusable.

More information about XML Schema [XSD BP]

5.4 XPath

1. XPath is not good for search engines. It is not meant to be like that.
2. Use “/” rarely, because it traverses the whole tree to a find proper mach
### APPENDIX A REFERENCES

#### A-I Books and standards

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[XFragment]</td>
<td>W3C Candidate Recommendation 20010212</td>
</tr>
<tr>
<td>[XPath]</td>
<td>XPath version 1.0 W3C Recommendation 19991116</td>
</tr>
<tr>
<td></td>
<td>XPath version 2.0 is currently W3C Working Draft, but it has also been used as a reference material, but mainly XPath 1.0 is used as a reference.</td>
</tr>
<tr>
<td>[XLink]</td>
<td>XLink version 1.0 W3C Recommendation 20010627</td>
</tr>
<tr>
<td>[XPointer]</td>
<td>XPointer version 1.0 W3C Candidate Recommendation 20010911</td>
</tr>
<tr>
<td>[DOM]</td>
<td>W3C Recommendations for DOM levels 1, 2, and 3.</td>
</tr>
</tbody>
</table>

#### A-II Web sites

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>
# APPENDIX B  TOOLS

## B-I XML Parsing

### B-I.1 XML Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XERCES</td>
<td>Apache’s XML parser. It is based on IBM’s XML4J XML parser.</td>
</tr>
<tr>
<td>JAXP</td>
<td>JAXP is SUN’s specification for vendor neutral API to process XML documents. It provides more general API than the most of the parser vendors. While being general, the JAXP has its downsides. Because it is general API, it presents general functions for processing XML. The interface it defines is not extremely powerful. Other vendors have their own functions to speed up the processing or manipulation. JAXP has API’s for SAX, DOM and XSLT.</td>
</tr>
<tr>
<td>MSXML</td>
<td>Microsoft’s XML/XSLT parser.</td>
</tr>
<tr>
<td>X-Performer</td>
<td>Republica’s XML parser</td>
</tr>
<tr>
<td>JDOM</td>
<td>There is no compelling reason for a Java API to manipulate XML to be complex, tricky, unintuitive, or a pain in the neck. JDOM™ is both Java-centric and Java-optimized. It behaves like Java, it uses Java collections, it is completely natural API for current Java developers, and it provides a low-cost entry point for using XML. While JDOM interoperates well with existing standards such as the Simple API for XML (SAX) and the Document Object Model (DOM), it is not an abstraction layer or enhancement to those APIs. Rather, it seeks to provide a robust, light-weight means of reading and writing XML data without the complex and memory-consumptive options that current API offerings provide.</td>
</tr>
</tbody>
</table>

### B-I.2 Other

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEX/YACC</td>
<td>Originally AT&amp;T’s lexical analyzer and grammar parser tools.</td>
</tr>
</tbody>
</table>

## B-II Transformation

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XALAN</td>
<td>Apache’s XSLT processor. Code base is the same and IBM’s XSLotus.</td>
</tr>
<tr>
<td>X-Wrapper</td>
<td></td>
</tr>
<tr>
<td>FOP</td>
<td>Apache’s Formatting Object Processor. FOP (Formatting Objects Processor) is the world’s first print formatter driven by XSL formatting objects and the world’s first output independent formatter. It is a Java application that reads a formatting object tree and then renders the resulting pages to a specified output.</td>
</tr>
</tbody>
</table>
### B-III Serialization

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZEUS</td>
<td>Zeus is, briefly, an open source Java-to-XML Data Binding tool. It provides a means of taking an arbitrary XML document and converting that document into a Java object representing the XML. That Java object can then be used and manipulated like any other Java object in the VM (virtual machine). Then, once the object has been modified and operated upon, Zeus can be used to convert the Java object back into an XML representation.</td>
</tr>
</tbody>
</table>