4

W3C Recommendations for the Web

In this chapter we describe the W3C Recommendations for the Web related to our XLink based Xspect prototypes. This includes an introduction to the XML standard and several of the adjunct formats based on XML.

4.1 The World Wide Web Consortium (W3C)

The World Wide Web Consortium was founded by Tim Berners-Lee in October 1994 at the Massachusetts Institute of Technology (Berners-Lee et al., 1994; W3C, 2002). The purpose of W3C is to encourage discussions in an open forum about the further development of the Web. This has led to many W3C Recommendations describing the specification of new technologies e.g., several versions of HTML, XHTML, HTTP, CSS, XML, SMIL, and others.

Before describing the next generation of Web formats, properties of the current Web is briefly outlined.

The architecture of the current Web comprise: the URL (Universal Resource Locator) addressing system, the HTTP (Hypertext Transfer Protocol), and the HTML (HyperText Markup Language). Links expressed in HTML are uni-directional and embedded in the source documents. Link types, describing the relationship between the source document and the destination anchor of a link, can be provided by the link attributes `rel` and `rev` (Raggett et al., 1999, sec 12.1.2). These link types are, however, poorly supported by browsers and only a few types are standardised e.g., ”stylesheet”, “start”,

“next”, and ”prev”.

In the following sections XML and the derived standards SVG, XPath, XPointer, and XSL are described. The relationship of these standards is depicted in figure 4.1 on the facing page. In the figure, XML is placed at the lowest level, because all data used in this architecture are expressed as XML elements. On the next level are the SVG, XLink, and XPath standards. SVG (Scalable Vector Graphics) is a standard for advanced vector graphics. XPath is designed to perform simple calculations over and navigate inside XML data. On the third level are the XPointer and XSL standards. XPointer defines locating schemes to locate single points or whole ranges in XML data. The XPointer schemes defined in the standard use XPath to locate single points or the start and end points for ranges in XML data, but is not limited to XPath. The stylesheet language XSL is used to format and transform XML data. XSL stylesheets describe how an XSL processing engine shall format or transform source XML data. Similar to XPointer, XSL is heavily dependent on XPath. XSL stylesheets often use XPath to express simple calculations and extract information from the source XML data. And finally, XLink, the XML linking standard, is placed at the fourth level.

XLink rely on the URL addressing scheme to specify the location of remote resources. This is depicted at the bottom to the left in figure 4.1 on the next page by the HTTP protocol.

When used together, these standards can produce interactive or static publications. Interactive publications e.g., hypertexts contain links or dynamic formatting, and static publications refer to traditional read only publications.

4.2 The XML standard

The Extensible Markup Language (XML) (Bray et al., 2000) was created to provide a simple mechanism to structure, store, and deliver information. XML is based on principles from two earlier markup languages, namely HTML and SGML.

The Standard Generalised Markup Language (SGML) was ratified by the ISO (8879) in 1986, and has since remained largely unchanged. The SGML specification describes an advanced and complex markup language for data format definition (SGML, 1986). SGML was very complex and applications supporting typical subsets of the standard tended to be incomplete, error-prone, and expensive. The conventions and principles of SGML were (and

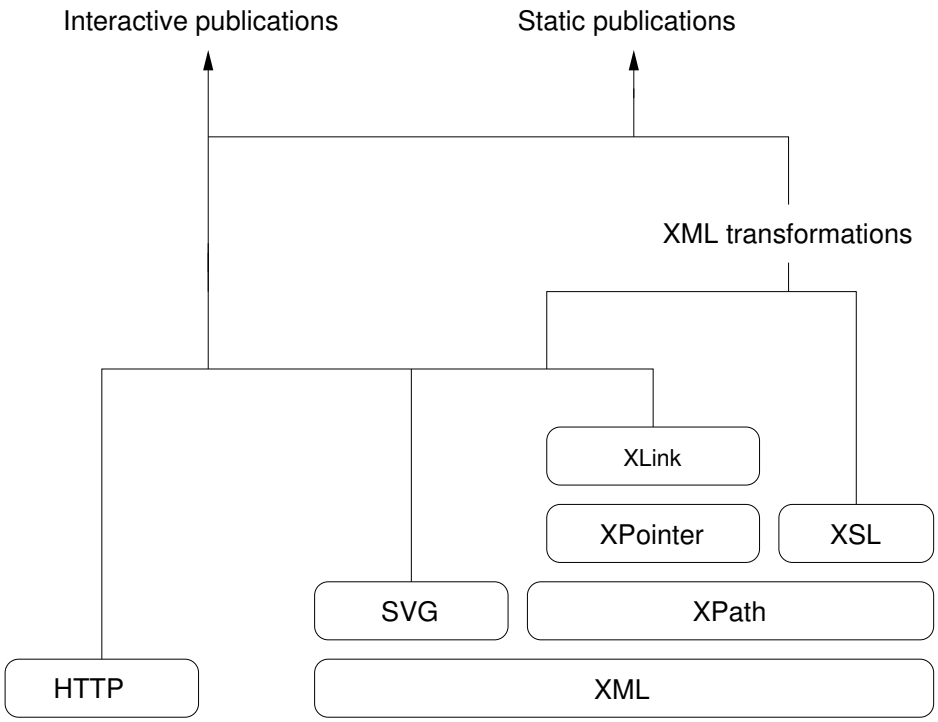


Figure 4.1: The relationship of the XML based standards. The lines show how the XML formats may be combined into different kinds of publications.

still are) overwhelming for implementors to deal with. This led to the idea of a simplified version of SGML to make it more attractive for software developers and end users (Bosak and Bray, 1999).

HTML was developed in 1990 as the hypertext document format for the Web (Berners-Lee, 1997; Berners-Lee et al., 1992, 1994). This new markup language had a very simple design compared to SGML. In contrast to the SGML meta language, HTML was designed as a fixed vocabulary. The simplicity of HTML made the task of creating parsers, viewers and editors for HTML easier than it had been for SGML. The first Web browser "WorldWideWeb" (Berners-Lee, 1997) was developed in 1990 by Tim Berners-Lee and made the Web available for browsing. Since then, the HTML specification has been extended many times with more features, but is still a fairly simple hypertext format.

The ideas of a simplified version of SGML and an enhancement of HTML suggested a new meta markup language to fill the gap. In 1996 the first working draft for XML was published and later in 1998 the recommendation

for XML 1.0 was presented. The development of XML continues and the recommendation for XML 1.1 is currently being prepared (Bray et al., 2000).

4.2.1 XML elements

First of all what is XML data? It can be seen as just a plain text file with markup tags. The markup tags define XML elements and the elements form the XML document structure. The XML markup language is used as the fundamental building block by the XML family of formats as illustrated in figure 4.2.

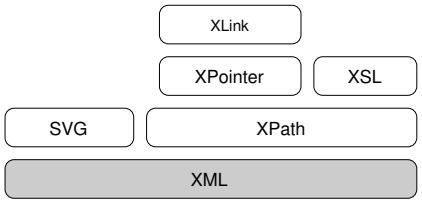


Figure 4.2: The XML standard is the fundamental building block for the adjunct standards.

An XML element is defined by a start-tag and an end-tag together with the data enclosed by them. The start-tag is delimited using the `<` and `>` characters. The end-tag is delimited with `</` and `>`.

Attributes can be added to the start-tag of an element to provide additional information. An attribute consists of an attribute name and its value. All white spaces (space, tab, line break) is equivalent to a single space character within the markup and may be used to separate attributes.

The XML specification additionally defines two kinds of XML elements besides the one described above: Declaration and Processing Instruction elements. Declaration elements are delimited by `<!` and `>`. An example of a declaration element is a comment which is delimited with `<!--` and `-->`. Declaration elements can contain embedded declarations using the `[` and `]` characters. Embedded declarations are often used in DTDs (described in section 4.2.3). Processing Instruction elements are delimited by `<?` and `?>`. These elements provide information about the XML data and are not normal data elements.

4.2.2 XML documents

A data object is an XML document if it is well-formed. To be well-formed the following three rules must be obeyed:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<XLINKNAVDATA>

  <ANNOTATION type="extended">
    <NOTE type="resource" label="daimi.14.1820313093">Testing 1.. 2..</NOTE>
    <LOC type="locator" title="Aabogade" label="daimi.54.70574732372"
     href="http://www.daimi.au.dk/~kgronbak/aabogade/#xpointer(\
     string-range(/,'institutions',1,13)[1])"/>
    <ARC type="arc" from="daimi.54.70574732372" to="daimi.14.1820313093"
     show="new"/>
  </ANNOTATION>

  <ANNOTATION type="extended">
    <NOTE type="resource" label="daimi.51.1594814848">31</NOTE>
    <LOC type="locator" title="Aabogade" label="daimi.12.41841843058"
     href="http://www.daimi.au.dk/~kgronbak/aabogade/#xpointer(\
     string-range(/,'34',1,3)[1])"/>
    <ARC type="arc" from="daimi.12.41841843058" to="daimi.51.1594814848"
     show="replace"/>
  </ANNOTATION>

</XLINKNAVDATA>
```

Figure 4.3: A well-formed XML document from the Xspect system.

- The document starts with an XML declaration.
- There is a root element in which all other elements are contained.
- All elements must be properly nested. Overlapping elements are not allowed.

Furthermore, XML documents can be validated against a Document Type Declaration (DTD). DTDs are grammars used to define restrictions on the logical structure of XML documents. An XML document is valid if it has an associated DTD and the document complies with the constraints expressed in this DTD.

An example of a well-formed XML document (representing a linkbase with two annotations from our Xspect prototypes) is presented in figure 4.3.

4.2.3 Logical structure (DTD and others)

The XML standard defines the construction of a document type declaration (DTD) that provides the grammar for a set of XML documents. A DTD dictates the formal logical structure of XML documents. This is expressed in list of declaration elements that define "data" elements and the structural relations between them.

DTD element declarations define new elements and their allowed contents. The declaration contains the keyword `ELEMENT` followed by the name of the element and a specification of the allowed content:

```
<!ELEMENT name .... >
```

The allowed content of an element declaration may be e.g., parsable character data (`PCDATA`), other elements, or a combination thereof. The content definition is called a model group. When a model group contains more than one content token, sequence control can be used. The sequence of tokens in the model group is controlled by the connector operator "`,`" and the choice operator "`|`". The connector operator corresponds to a logical `and` and the choice operator to a logical `or`. Furthermore, the quantity of each token representing an element may be set by a quantity indicator ("`?`", "`+`", and "`*`") placed right after the token. If a token is followed by the "`?`" indicator, the element is optional. The "`+`" indicator specifies one or more repeated occurrences of the element. The equivalent of the (illegal) combination "`?+`", an optional repeated element, is represented by the "`*`" indicator. If a token has no quantity indicator, a single occurrence of the element is required.

The attributes of an element are declared separate from the element declaration. An attribute declaration element is identified by the keyword `ATTLIST` followed by the name of the "data" element containing the attribute. In the remainder of the attribute declaration, the actual attributes are defined as name and type pairs:

```
<!ATTLIST name_of_element attr_name attr_type.... .... >
```

The most commonly used attribute type is the `CDATA` (character data) which indicates a string of characters. This type provides exactly the same restriction as if no DTD was used. The type `ID` indicates that the value should be a unique string, and values of the type `IDREF` may only hold values predefined

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT XLINKNAVDATA (ANNOTATION*)>
<!ELEMENT ANNOTATION (NOTE, LOC, ARC)>
<!ATTLIST ANNOTATION
        type CDATA #FIXED "extended"
>
<!ELEMENT LOC EMPTY>
<!ATTLIST LOC
        type CDATA #FIXED "locator"
        title CDATA #REQUIRED
        label ID #REQUIRED
        href CDATA #REQUIRED
>
<!ELEMENT NOTE (#PCDATA)>
<!ATTLIST NOTE
        type CDATA #FIXED "resource"
        label ID #REQUIRED
>
<!ELEMENT ARC EMPTY>
<!ATTLIST ARC
        type CDATA #FIXED "arc"
        from IDREF #REQUIRED
        to IDREF #REQUIRED
        show (new | replace | embed) #REQUIRED
>
```

Figure 4.4: A DTD which the XML example in figure 4.3 on page 47 conforms to.

in an `ID` type attribute value. An attribute type can also be a finite set of allowed values delimited by "`(`" and "`)`".

The attribute type definition may be followed by a default value which is used when the XML document does not specify a value for the attribute. Instead of a default value, it may be stated that the attribute is required to be defined by the keyword `#REQUIRED`. If the attribute is optional the keyword `#IMPLIED` is used. Finally the keyword `#FIXED` is used when the attribute holds a constant value.

The XML example in figure 4.3 on page 47 conforms to the DTD shown in figure 4.4. Notice the element declaration for `XLINKNAVDATA`. This element contains zero or more `ANNOTATION` elements. An `ANNOTATION` element have three child elements: `NOTE`, `LOC`, `ARC` and the fixed attribute `type` with value

"extended". The `LOC` element has four attributes: `title`, `href` ,`type` (which is fixed to the "locator" value), and the `label` attribute which contains a unique ID. The `NOTE` element has PCDATA as its content and two attributes: `type` which is fixed to "resource" and `label` which holds a unique ID. Furthermore, the `ARC` element has no content defined, but has four attributes, namely `type` which is fixed to "arc", `from` and `to` which hold values introduced by the `label` attribute of `LOC` and `NOTE`, and the attribute `show`.

The declarations forming a DTD may be placed in a document type declaration at the top of each XML document, which should conform to the DTD. More commonly though, the DTD is stored in a separate file and referred to from the XML document within a document type declaration element. This mechanism is described further in section 4.2.4. A document type declaration is identified by the keyword `DOCTYPE` followed by the root element name of the XML data which should conform to the DTD:

```
<!DOCTYPE root_element_name .... >
```

After the doctype name, a reference to an external file containing the declaration elements can be provided by the keyword `SYSTEM` followed by a quoted URL to the file. A combination with an internal list of embedded declarations is allowed.

The DTD specification was designed with traditional text documents in mind, so the concept of data types in DTDs is rather weak. Thus DTDs provide poor means for specifying criterias concerning the content of a given element, other than text.

The XML Schema standard (Fallside, 2001) is an alternative to DTDs for describing restrictions on the structure and content of an XML document. Schemas support the concept of data types. Compared to DTD which is a formal description primarily for text formats, a Schema is designed to describe database like structures where element content requires validation.

DTDs and Schemas can be used independently and in the same document. Both have certain qualities and are appropriate in different situations, therefore it is not likely that Schemas will replace DTDs entirely. Schemas are not the only alternative to DTDs, but at this time the most widely supported. Other validation technologies exist such as RELAX (Murata, 2000), Schematron (Jelliffe, 2001), and DSD (Klarlund et al., 2000). All these are XML based and use patterns to define the grammars. In order to support the concept of data types, they can be used in combination with Schemas.

4.2.4 Physical structure

The XML specification defines mechanisms to physically isolate and separate any part of an XML document. This could e.g., be each chapter of a book or each template in a stylesheet. Such separate parts are called entities. Each entity is assigned a name so it can be referenced. When the XML data is fed to an XML processor the references are resolved and replaced with the entity content.

Entities may be distributed among a number of separate files, facilitating reuse of entities throughout several documents. Furthermore, entities may contain references to other internally or externally defined entities and even to non-XML data.

Entities are often considered in the following situations:

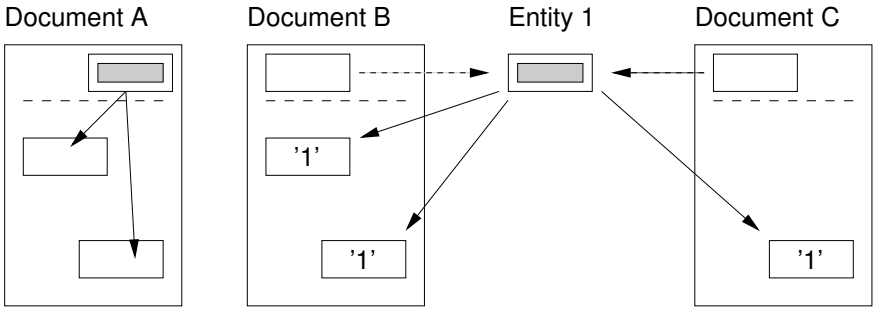
- The same information is used in several places.
- An XML document is so large that it should be split into appropriate units.
- A reference to data which is not XML based is needed.

An entity must be declared before it is referenced. The entity declaration is identified by the keyword `ENTITY` and is placed inside a document type declaration.

```
<!DOCTYPE root_element_name [
  <!ENTITY entity_name .... >
]>
```

The keyword `ENTITY` is followed by the entity name and the entity content. The content can either be a quoted string or the keyword `SYSTEM` followed by a quoted URL which locates a file containing the content. An entity reference is defined by the entity name enclosed with the "`&`" and "`;`" characters.

An entity declaration can be either internal or external. The architecture of the two kinds are depicted in figure 4.5 on the following page. In figure 4.5(a) the entity is declared as an embedded declaration inside a document type declaration element of document `A`. All references to this entity are contained inside the document. In figure 4.5(b) the external entity named `1` is declared in a separate file. The documents `B` and `C` have to import (wrap) entity `1` in



(a) An internal entity.

(b) An external entity.

Figure 4.5: The architecture of the two kinds of entity declaration: internal and external.

the document type declarations, before it can be referenced by local entity references.

Entities are often used to encode data with special characters which otherwise could not be expressed and many XML processors have a set of built-in entity declarations. Some of these entities can be used instead of the actual characters in element and attribute values to avoid confusing the XML processor while parsing the document. Examples of such entities are:

- `<` for '<'
- `>` for '>'
- `&` for '&'
- `&apos;` for '''
- `&quot;` for '"'.

Entities provide a flexible way to physically arrange and assemble XML documents.

4.2.5 XML Namespaces

XML namespaces (Bray et al., 1999) provide a scoping mechanism for XML element names. The motivation for using namespaces is to avoid name clashes

of element names or attribute names.

The attribute name `xmlns` is used to assign a namespace URL to the a namespace prefix. In the following example, the namespace URL "http://www.w3.org-/1999/xlink/" is assigned to the prefix `xlink`:

```
xmlns:xlink="http://www.w3.org/1999/xlink/"
```

A qualified namespace name consists of a namespace prefix and a local part, separated with a single colon. The prefix maps directly to the namespace URL and the local part is an element or attribute name defined in this namespace:

```
<LINK xlink:type="extended"/>
```

Though not required, it is common practice to let the namespace URL refer to some sort of a description of the namespace on a Web page. An example of an XML document using the XLink namespace is presented in figure 4.6 on the next page. Notice that the namespace prefix `xlink` is defined in the start-tag of the `XLINKNAVDATA` element and used in the attribute names of the subelements. Furthermore the example is a valid XML document being well-formed and conforming to the associated DTD.

4.3 Adjunct standards

4.3.1 Navigating XML data (URL and XPath)

Uniform Resource Locator

The URL (Uniform Resource Locator) scheme is used to identify resources on the Internet, but can also be used to locate resources on a local machine or on an intranet. URLs are used in HTML to specify link destinations and in XML to locate entities, provide unique identifiers for XML namespaces, and for location-specifiers in XLink.

The URI (Uniform Resource Indentifier) standard is a more recent addressing standard and includes the URL scheme.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE XLINKNAVDATA SYSTEM "anno.dtd">
<XLINKNAVDATA xmlns:xlink="http://www.w3.org/1999/xlink/">

<ANNOTATION xlink:type="extended">
  <NOTE xlink:type="resource" xlink:label=\
   "daimi.14.1820313093">Testing 1.. 2..</NOTE>
  <LOC xlink:type="locator" xlink:title="Aabogade" xlink:label=\
   "daimi.54.70574732372" xlink:href="http://www.daimi.au.dk/~kgronbak/\
   aabogade/#xpointer(string-range(/,'institutions',1,13)[1])"/>
  <ARC xlink:type="arc" xlink:from="daimi.54.70574732372"
   xlink:to="daimi.14.1820313093" xlink:show="new"/>
</ANNOTATION>

<ANNOTATION xlink:type="extended">
  <NOTE xlink:type="resource" xlink:label="daimi.51.1594814848">31</NOTE>
  <LOC xlink:type="locator" xlink:title="Aabogade"
   xlink:label="daimi.12.41841843058" xlink:href="http://www.daimi.au.dk/\
   ~kgronbak/aabogade/#xpointer(string-range(/,'34',1,3)[1])"/>
  <ARC xlink:type="arc" xlink:from="daimi.12.41841843058"
   xlink:to="daimi.51.1594814848" xlink:show="replace"/>
</ANNOTATION>

</XLINKNAVDATA>
```

Figure 4.6: A valid XML document from the Xspect annotation system using the XLink namespace and the DTD presented in figure 4.4 on page 49 (anno.dtd).

A URL is simply a string used to identify a resource on the Web. The first part of a URL indicates what protocol to use, and the second part specifies the IP address or the domain name where the resource is located:

$$\overbrace{\text{http}}^{protocol} : // \overbrace{\text{fahbentor.daimi.au.dk}}^{domain} / \overbrace{\text{xspect/index.html}}^{path}$$

URLs can be extended with a fragment identifier, using the "#" character, to address into a resource. In HTML the fragment identifier refers to the name of a predefined anchor element. The XPointer standard (described in section 4.3.2) provides a similar fragment identifier mechanism for XML data. However, the locating mechanism of XPointer goes beyond simply referring to predefined anchors and supports locating fragments by context.

XPath

XML Path (XPath) (Clark and DeRose, 1999) defines a common syntax to address parts of an XML document. XPath expressions are always evaluated in the context of an XML document. How the document is located, is not a part of the XPath standard, but is specified by URLs. In the following examples, the XPath expressions are evaluated in the context of the XML document presented in figure 4.6 on the facing page.

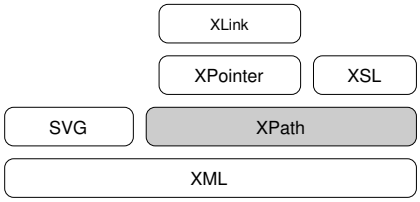


Figure 4.7: The XPath standard is designed to navigate inside XML data and to perform simple calculations over the XML data.

To specify the node set of all `ANNOTATION` elements, the following XPath expression can be used:

```
/XLINKNAVDATA/ANNOTATION
```

To locate a single node in a node set, the set may be extended by an index as illustrated below. The index string can be a number (starting from 1) or a function as defined in the XPath specification e.g., the `last()` function returns the index of the last node in the set. The example path below evaluates to the value of the `xlink:title` attribute, contained by the `LOC` element in the first `ANNOTATION` element:

```
/XLINKNAVDATA/ANNOTATION[1]/LOC/@xlink:title
```

Simple tests on a node set can be used to select certain nodes. The functions defined in the XPath specification provide tests on node sets, strings, and numbers. The example path below illustrates the string function `contains()` which returns `true` if the first string argument contains the second string argument and `false` otherwise:

```
/XLINKNAVDATA/ANNOTATION/LOC/[contains(@xlink:title,'bog')]
```

This example results in a node set with the two `LOC` elements of each `ANNOTATION` element.

Note that the XPath standard does not define how to handle the result of an evaluated expression. In e.g., XSLT stylesheets (described in section 4.3.3) the result can be extracted and further transformed into output, but this is beyond the XPath specification.

4.3.2 XML Linking (XLink and XPointer)

One of the purposes of XML is to be the next generation of HTML. Likewise, XML Linking (XLink) is designed to be more powerful and flexible than HTML linking but easier to understand and implement than the SGML linking standard HyTime (HyTime, 1992).

XML linking consists of two parts: XLink, expressing link structures and XPointer, expressing location of resources.

XLink

The XLink recommendation (DeRose et al., 2001) describes a vocabulary that provides linking mechanisms for XML resources. XLink can be used as traditional HTML-style links, but have more advanced features making it easier to maintain links over time and separated the link structures from the linked resources into separate files.

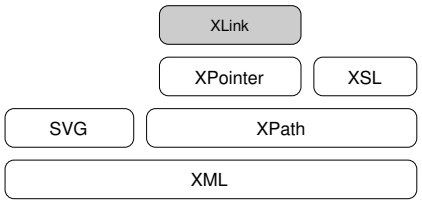


Figure 4.8: XLink is the XML linking standard.

The model of XLink consists of three main items:

Links which contains the other components and which may have a description of the link as a whole attached.

In figure 4.9 on the next page the element `LINK` represents a link with the description "Link 4".

Participants in a link can be whole XML or non-XML resources or fragments of such resources. Any number of resources can participate in a link and meta-data can be associated with each participant. Participants are either contained directly in a link (a local resource) or indirectly by means of a URL reference (a locator to a remote resource).

In the XLink example in figure 4.9 on the facing page, three participants are defined by the `LOC` elements. They are all remote resource-fragments referred to by locators.

Arcs connect participants and designate the starting point and the ending point for a link. Meta-data may be attached to describe each arc. Furthermore, the behaviour attributes `show` and `actuate` can be used

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<XLINKNAVDATA xmlns:xlink="http://www.w3.org/1999/xlink">

<LINK xlink:type="extended" xlink:id="daimi.4.1017075150"
 xlink:title="Link 4">

  <ARC xlink:type="arc" xlink:title="AARHUS"
   xlink:from="daimi.6.1017075150"/>

  <LOC xlink:type="locator" xlink:title="Welcome to Computer Science\
   in Aarhus (DAIMI)" xlink:href="http://www.daimi.au.dk/#xpointer(\
   string-range(/,'UNIVERSITY OF AARHUS',15,6)[1])"
   xlink:label="daimi.6.1017075150"/>

  <ARC xlink:type="arc" xlink:title="Department"
   xlink:from="daimi.23.1017079993"/>

  <LOC xlink:type="locator" xlink:title="About the Department"
   xlink:href="http://www.daimi.au.dk/doc74.html#xpointer(\
   string-range(/,'the Department',5,10)[1])"
   xlink:label="daimi.23.1017079993"/>

  <ARC xlink:type="arc" xlink:title="Information"
   xlink:from="daimi.28.1017080011"/>

  <LOC xlink:type="locator" xlink:title="Contact Information"
   xlink:href="http://www.daimi.au.dk/doc12.html#xpointer(\
   string-range(/,'Information',1,11)[1])"
   xlink:label="daimi.28.1017080011"/>

</LINK>

</XLINKNAVDATA>
```

Figure 4.9: A linkbase from the Xspect hypermedia system.

to describe what happens when the link is activated and how the link is activated.

The example in figure 4.9 illustrates the use of XLink. Three arcs are specified by `ARC` elements. Each arc defines a link starting point with the `xlink:from` attribute, but the ending point is not explicitly defined. When no ending point is defined for an arc, it defaults to the set of all participants in the link. This example is used throughout the

introduction of XLink and will be described in detail as the various items of XLink are introduced.

The meta-data used on these components are divided into human-readable *titles* and machine parsable *roles*. Role attributes have URLs as values, which can optionally specify meta-data describing the meaning of the element.

The XLink namespace defines a set of attributes to be used for specifying the linking properties of XML elements. Because the XLink standard only uses attributes to define link structures, there is no normative DTD defined for the namespace. This can be seen in the XLink example in figure 4.9 on the preceding page where non of the element names have a namespace prefix. Instead, the attribute `xlink:type` is used to define the linking properties of each element.

XLink supports two kinds of links: *extended* links and *simple* links. The relationship of the XLink attributes for the two kinds of links are illustrated in figure 4.10 on the next page. Notice that the element names are of no importance to the linking properties and how it is the `xlink:type` attribute which decides the meaning of the other XLink attributes. The XLink attributes and the relationship among them are now described in more detail.

simple: A `simple` link is a shorthand for an often used link structure. This is when a link is outbound (described below) and have exactly two participating resources. A `simple`-type link element is not dependent on other XLink elements. The information needed to determine all characteristics for the link is provided with use of the following attributes: `xlink:title`, `xlink:role`, `xlink:href`, `xlink:actuate`, `xlink:arcrole`, and `xlink:show`.

extended: An `extended` link can connect any number of local and remote resources. Thus, inbound and third-party links (described below) can be expressed with `extended` links. An `extended`-type link element can have the attributes `xlink:title` and `xlink:role` defined to describe the meaning of the link element. The information of the participating links are supplied by embedding the following XLink type elements in any order and number:

- `locator`-type elements which refer to remote resources participating in the link.
- `arc`-type elements which describe the link's traversals behaviour.

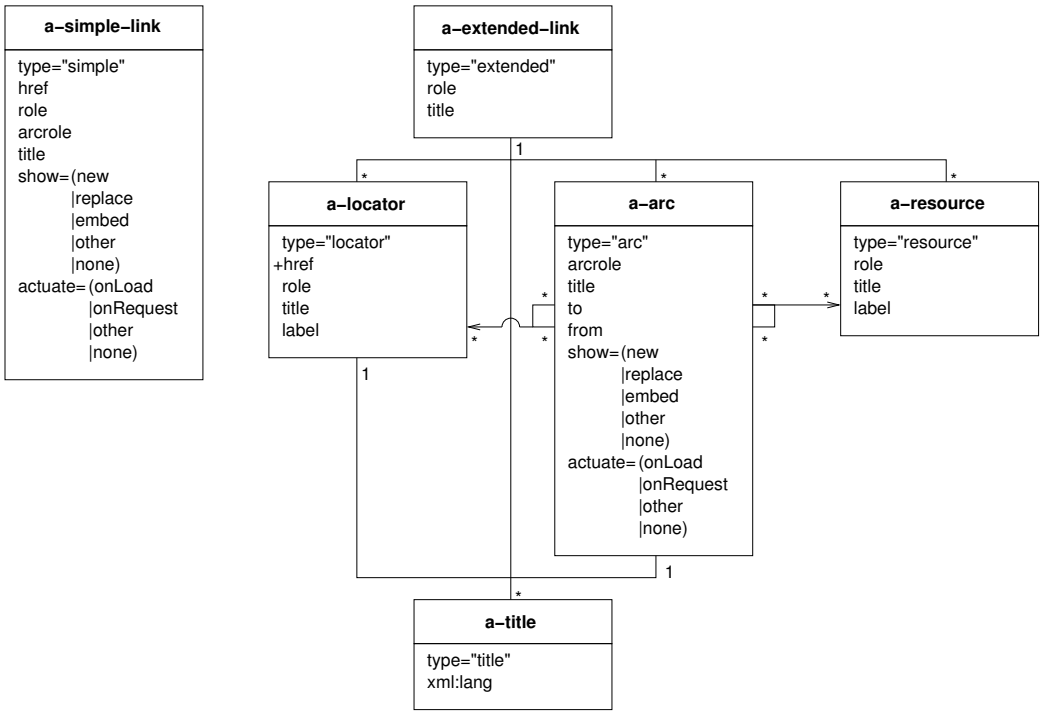


Figure 4.10: A generic XML tree showing the intended use of XLink attributes for both simple and extended links. The precise semantics of the diagram is described in appendix C.

- `title`-type elements which provide human-readable descriptions for the link.
- `resource`-type elements which refer to local resources participating in the link.

If other types of link elements are used inside an `extended` link e.g., a new `extended` link element, it has no XLink specific meaning. The three element types `locator`, `arc`, and `resource` are intended to be used inside an `extended` link. If used otherwise they have no XLink specified meaning.

arc: An `arc`-type element is always used in an `extended` link as a child element. The attributes `xlink:from`, `xlink:to`, `xlink:show`, `xlink:-actuate`, `xlink:arcrole`, and `xlink:title` can be used to describe an `arc` type element. Any content is allowed in an `arc`-type element, but the only XLink specified type which can be used is the `title`-

type element. The `arc`-type elements define associations between resource, how they should be traversed, and presented. With the two attributes `xlink:to` and `xlink:from`, an association between two labeled resources can be described. When several `arc` type elements have the same participant for the `xlink:from` attribute, a multi-headed link is specified starting from this particular participant.

resource: The `resource`-type element is used as a child element in an `extended` link. It can have the attributes `xlink:role`, `xlink:title`, and `xlink:label`. The elements with this type are used to embed local resources inside an `extended` link. A `resource` element may have any content.

locator: A `locator`-type element is used as a child element of an `extended` link. The element can have the attributes `xlink:role`, `xlink:title`, and `xlink:label`, and must have the `xlink:href` attribute defined. A `locator`-type element does not describe an association between itself and the remote resource. It has to be used together with an `arc`-type element to fully describe a link association. The content of a `locator`-type element can be of any kind, but the only XLink defined subelement is the `title`-type elements.

title: The `title`-type element can be used as a subelement in the `extended`, `arc`, and `locator`-type elements. These three element types can also have a `xlink:title` attribute. Both the `xlink:title` attribute and the `title`-type element are used to describe the surrounding element in a human readable language, so why have the `title`-type element at all? In cases where a link element needs to have multiple titles to support e.g., a number of different natural languages or context sensitive titles, the `title`-type element becomes very useful. However, there is no consensus on how to the use the `title`-type element instead of the `title` attribute. When both a `xlink:title` attribute and one or more `title`-type elements are defined to describe the same element, they have no XLink specified relationship.

When an arc has a local start resource and a remote destination resource, the arc is said to be outbound. If an arc has a local destination resource and a remote start resource, then the arc is inbound. An arc is called a third-party arc if neither the start nor the destination resources are local. A special kind of XML documents containing only inbound and third-party links are called linkbases. With the use of linkbases, the linking elements

can be separated entirely from the linked resources. The XML document presented in figure 4.9 on page 57 is such a linkbase, as it only contains third-party links.

XPointer

The XML Pointer Language (XPointer) (Daniel et al., 2001) provides a mechanism for addressing into the internal structures of XML documents. More precisely, XPointers are fragment identifiers that can be added to a URL when linking to an XML document.

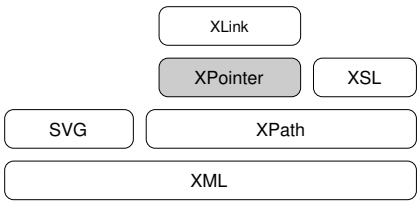


Figure 4.11: XPointer defines locating schemes to locate single points or whole ranges in XML data.

Below is an example of an XPointer referring to the `NOTE` element of the second `ANNOTATION` element in the root element `XLINKNAVDATA` of a file `anno.xlink` (anno.xlink is the file presented in figure 4.6 on page 54).

```
anno.xlink#xpointer(XLINKNAVDATA/ANNOTATION[2]/NOTE)
```

The keyword `xpointer` in the example is the name of the used XPointer scheme. A scheme is a protocol on how to locate parts of an XML resource. The XPointer specification introduces the scheme `xpointer` which uses XPath expressions to specify locations, but goes beyond XPath in that it can locate ranges across element borders. The `xpointer` scheme is primarily used to address XML elements containing text. Location points and ranges are defined with the granularity of a single character.

When a scheme not defined in the XPointer specification is used for an XML based media type, it is recommended that a fallback to the scheme `xpointer` is always provided. An XPointer expression can consist of one or more XPointer parts each having a scheme name and an expression. These parts are separated by whitespaces and will in turn be evaluated from left to right until a part can be evaluated successfully. In this way the fallback actions can be expressed directly in an XPointer expression. This is particularly useful if a location can be expressed in different ways.

4.3.3 Formatting and Transformation (XSL)

One of the design goals for XML documents is to be easily understandable and readable for both humans and machines. However, this is not always the case for human readers. The solution is a stylesheet describing how the elements of an XML document should be presented.

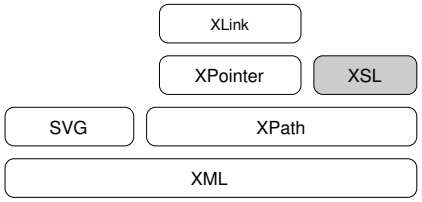


Figure 4.12: The stylesheet language XSL is used to format or transform XML data.

The eXtensible Stylesheet Language (XSL) (Adler et al., 2001) is a language for expressing stylesheets. The standard is split into XSL-FO (XSL Formatting Objects) which is a vocabulary of formatting instructions and XSLT (XSL Transformation), which is a language expressing transformation of XML documents.

XSL-FO is intended to be used as a typesetting and formatting description. XML formatted by XSL-FO can be served directly to an XSL-FO aware client or be processed further to a client-compatible document format e.g., HTML or PDF.

The XSLT standard specifies a language to describe transformations of XML documents. Transformation rules are listed as templates which matches XML data input using XPath expressions. A template contains the description of how an XML element is to be transformed. XSLT also defines a small programming language which can be used to manipulate the matched XML. It provides variables, conditional statements, iterators, and recursions over the input data.

Figure 4.13 on the facing page presents a stylesheet which can be applied to the XML document in figure 4.6 on page 54. In the stylesheet, the variable `newline` is defined and later used in the output. The `xsl:for-each select` statement is an iteration over the node set of all `NOTE` elements, printing the `xlink:label` attribute and `NOTE` content. Notice how all navigation and extraction is specified by XPath expressions. The result of the stylesheet applied to the XML document in figure 4.6 on page 54 is:

```
The content of note daimi.14.1820313093 is: Testing 1.. 2..
The content of note daimi.51.1594814848 is: 31
```

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

  <xsl:output method="text" indent="no"/>

  <xsl:variable name="newline">
    <xsl:text>
    </xsl:text>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:for-each select="/XLINKNAVDATA/ANNOTATION/NOTE">
      <xsl:text>The content of note </xsl:text>
      <xsl:value-of select="@xlink:label"/>
      <xsl:text> is: </xsl:text>
      <xsl:value-of select="."/>
      <xsl:value-of select="$newline"/>
    </xsl:for-each>
  </xsl:template>



</xsl:stylesheet>
```

Figure 4.13: A stylesheet made to extract information from the XML document presented in figure 4.6 on page 54.

4.3.4 XML graphics (SVG)

SVG (Scalable Vector Graphics) is a standard for two dimensional vector graphics. Vector based graphics describe an image using the abstraction level of geometric figures. However, since all displays are raster oriented a SVG image has to be computed and rasterised before it can be presented.

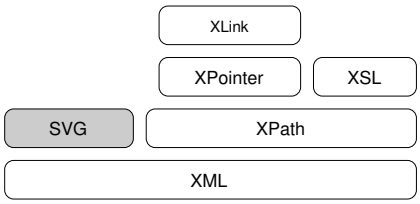


Figure 4.14: SVG (Scalable Vector Graphics) is a standard for advanced vector graphics.

SVG can be used as a stand-alone format, but is also intended to be used together with other XML formats. As SVG images are expressed in XML syntax they can be ad-

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE svg SYSTEM
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="15cm" height="15cm" viewBox="0 0 100 100">

  <line y1="50" x1="20" y2="50" x2="80" stroke="rgb(0%,0%,0%)"
   stroke-width="2"/>

  <text x="15" y="35" font-family="Arial" font-size="10">A</text>

  <ellipse cx="20" cy="50"  rx="8" ry="8" fill="rgb(100%,100%,100%)"
   stroke="rgb(0%,0%,0%)" stroke-width="2"/>

  <text x="75" y="35" font-family="Arial" font-size="10">B</text>

  <ellipse cx="80" cy="50" rx="8" ry="8" fill="rgb(100%,100%,100%)"
   stroke="rgb(0%,0%,0%)" stroke-width="2"/>

</svg>
```

Figure 4.15: A SVG file describing two circles A and B connected by a line.

dressed and transformed like all other XML data.

The XML representation of a simple SVG image containing two circles, a line, and some text is presented in figure 4.15. The elements `svg`, `line`, `text`, and `ellipse` and their attributes are defined in the SVG DTD. Figure 4.16 presents the same XML data rendered into a raster image.

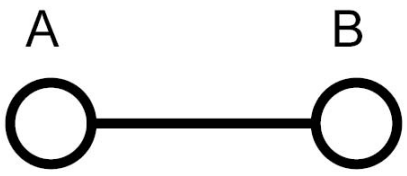


Figure 4.16: A SVG image showing two circles A and B connected by a line