7

XML Linking Language

In this section we describe the XML Linking Language (XLink) [DeRose+ 01b], which defines how hyperlinks can be used in an XML-based environment. In section 1.3, we described how linking is done in today's Web. This description is based on the assumption that resources use HTML as their document markup language. However, increasingly XML documents will become available on the Web. There are two possibilities regarding how hyperlinks can be used with XML documents:

- *Application-specific hyperlink definitions.* It is possible to define hyperlinks specifically for certain application areas. One example is XHTML, where the `<a>` element is defined to have the same semantics as HTML's `<A>` element.[1] However, this approach requires XML processors to have built-in knowledge of these specific applications, otherwise they will not recognize the element's hyperlink semantics. Consequently, this approach is not very general and reduces XML's usefulness as a mechanism to freely define and distribute application-specific hypermedia document types.
- *General hyperlink specification mechanism.* In contrast to an application-specific definition of hyperlinks, a general mechanism for using hyperlinks in XML documents can be defined. This is exactly what XLink does. XLink is an application-independent way to use hyperlinks in XML documents, so that an XLink-aware application will be able to recognize and correctly interpret hyperlinks in any XML document, as long as the document uses XLink for its hyperlinks. This works regardless of the actual schema the document is using.

Since XLink defines a general mechanism for embedding hyperlinks in XML documents, it must define a mechanism that enables XLink-specific

[1] It should be noted that HTML by definition is case-insensitive, while XHTML accepts only lowercase element and attribute names.

information to be recognized in an XML document. This aspect of XLink is described in section 7.1. XLink makes it possible to specify different types of links using a number of different element types, described in section 7.2. For actually embedding the hyperlink information into XML documents, XLink uses a number of attributes to assign different hyperlink semantics to XML elements, and these attributes are described in detail in section 7.3. Section 7.4 contains explanations of the processing model and conformance requirements that must be satisfied by applications claiming to implement XLink. In section 7.5, we look at how XLinks may be utilized. Finally, section 7.6 describes the future of XLink.

7.1 EMBEDDING LINKS INTO XML DOCUMENTS

XML, as described in section 4.1, is a language for the specification of document types and instances (i.e., documents) of these types. As such, element and attribute names in XML are in no way restricted (except for the syntactic naming rules given by the XML specification) and can be freely chosen by document type authors. However, when XML Namespaces (described in section 4.2) are taken into account, the rules for XML names become a bit more restrictive by dividing names into a namespace prefix and a local part specific to a particular namespace. XLink takes advantage of the XML Namespaces mechanism by defining a namespace of its own. (The official XLink namespace URI is `http://www.w3.org/1999/xlink/namespace/`.) Consequently, any applications implementing XLink must also implement XML Namespaces.

XLink's model of embedding links into XML documents is very simple. XLink defines a number of attributes (described in detail in section 7.3), and these attributes are all in the *global attribute partition* of the XLink namespace (as described in section 4.2). This means that once the XLink namespace has been declared, these attributes can be used on elements from any namespace. The purpose of this is to provide XLink with a way to assign hyperlink semantics to arbitrary elements. The following short XML fragment demonstrates this method of embedding XLink into XML documents:

```
<TypeA xmlns:xlink="http://www.w3.org/1999/xlink/namespace/">
  ....
  <TypeB xlink:type="simple" xlink:href="http://transcluding.com/">
    ....
  </TypeB>
</TypeA>
```

In this example, an arbitrary element (`TypeB`) is employed as an XLink simple link by using two attributes from XLink's namespace on that

element. Since, from XML's point of view, the XLink attributes are normal attributes, they have to be declared in the document's DTD. Consequently, the DTD designer must be aware of the fact that XLink is going to be used. However, if DTD designers want to declare elements as being XLink hyperlinks in a more built-in fashion, they can do so by declaring the XLink attributes in the DTD using default values, as shown in the following example:

```
<!ATTLIST TypeA
  xmlns:xlink  CDATA    #FIXED "http://www.w3.org/1999/xlink/namespace/"
>

<!ATTLIST TypeB
  xlink:type   (simple) #FIXED "simple"
  xlink:href   CDATA    #REQUIRED
>

<TypeA>
  ....
    <TypeB xlink:href="http://transcluding.com/">
      ....
    </TypeB>
</TypeA>
```

In this case, the `xmlns:xlink` and `xlink:type` attributes are declared as having fixed values in the DTD, so the attributes do not need to be specified on the element instances. The `xlink:href` attribute is declared as being mandatory and, therefore, must be specified on the element instance. The results of this declaration are that the `TypeB` element type becomes a simple XLink element type by default and that any occurrence of this element has to specify the `xlink:href` attribute for designating the hyperlink's destination (see section 7.5.1 for a more detailed discussion of this technique).

In these examples, we have used the simplest way of specifying hyperlinks with XLink. This method is very similar to HTML's linking mechanisms, but it also has the same restrictions as HTML. However, XLink supports a much more powerful linking model, which is explained in detail in chapter 3 and reviewed in the subsequent section on XLink link types. The basic mechanism of using XLink in the context of XML documents is the same for the simple as well as for the more complex linking situations in XLink.

7.2 LINK TYPES AND ELEMENT TYPES

XLink defines two different types of links, described in section 7.2.1. These conceptual link types utilize certain XML element types (identified by

Table 7.1 Relation Between XLink Link and Element Types

XLink Link Type	Related XLink Element Type(s)
simple	`simple`
extended	`extended`, `locator`, `arc`, `resource`, `title`

XLink attributes) for their implementation. Specifically, the XLink element types, described in section 7.2.2, are different from the conceptual link types supported by XLink. (This can be confusing when starting to learn XLink, because one link often is represented by more than one XML element.) The method of expressing XLink information in XML (i.e., by using XML elements that are interpreted as XLink elements because they contain XLink-conforming attributes) makes it necessary to make some information repeatable by defining specific element types for it. (XML attributes can occur only once on an element, so repeatable information has to be represented by child elements rather than attributes.) Table 7.1 shows how the two concepts relate.

As can be seen, XLink extended links use a number of XLink element types, while XLink simple links use only one. The reason for this is XLink's goal to make the transition from HTML's linking style to XLink easy by providing a mechanism (the simple links) that is very similar to HTML links. In the following two sections, we discuss in detail the link types and the element types.

7.2.1 XLink Link Types

XLink supports two types of links: *simple* and *extended*. While the simple links are modeled after HTML's linking model (making a transition from HTML links to XLink simple links very easy), XLink's extended links are far more powerful and are intended to be used by applications requiring a more elaborate linking model than HTML's. Conceptually, simple links can be viewed as a special case of extended links, but since they use a different syntax, they are regarded as a different type of link. Before we discuss XLink's link types, we will revisit three key concepts introduced in chapter 3.

- *Number of linked resources.* The number of resources included in a link (called the *participating resources*) is one of the key aspects of every link and every link model. While HTML constrains this number to two resources (the local anchor and the remote resource),

theoretically this number can be anything between one[2] (being equivalent to an annotation of a resource) and a very large number.

- *Outbound/inbound/out-of-line.* Outbound and inbound links are links in which at least one of the participating resources is part of the link itself (as a starting resource in the case of outbound links, as an ending resource in the case of inbound links). HTML is restricted to outbound links, because the local anchor (which is the starting resource of the link) is defined to be the content of the `<A>` element. An out-of-line link, however, does not have any of the participating resources as part of the link itself. Therefore, it can be moved around much more easily and, in particular, can be used to create links between resources for which the link creator has no write access.
- *Unidirectional/multidirectional.* Unidirectional links can be traversed only in one direction. For example, HTML links point from the `<A>` element's occurrence to the remote resource. If links are out-of-line, then they can be used from all participating resources to initiate traversal. Every link that can be used for traversal from more than one of its participating resources is said to be multidirectional.[3] This, however, also includes the possibility that the link itself contains information limiting its possibilities for traversal, for example, the specification that one participating resource of a link can never be used as a target for traversal.

Knowing these basic properties of links, we can now take a closer look at XLink's link types and how they are used within XML documents.

Simple Links

XLink simple links are very similar to the links provided by HTML. Simple links associate two resources[4] with a unidirectional link, and they are always

[2] A link having no resources would also be a valid XLink, but it would not make any sense (maybe only as a placeholder).

[3] It is important to recognize that the popular "back" button in today's browsers does not make HTML links multidirectional, since it is a feature of the browser (locally storing a history of previously visited HTML pages) and not of the link itself. Obviously, one cannot initiate traversal in the "back" direction if the current resource has not been the result of a previous traversal (i.e., one cannot use the browser's "back" button if the browser has only just been started).

[4] Note that the XLink recommendation contains a slight contradiction. In one place it states, "A simple link is a link that associates exactly two resources, one local and one remote, with an arc going from the former to the latter," and then in another place it states, "It is not an error for a simple-type element to have no locator (`href`) attribute value. If a value is not provided, the link is simply untraversable. Such a link may still be useful, for example, to associate properties with the resource by means of XLink attributes." In other words, a simple link may be missing the `href` attribute and so have only a single participating resource.

inline. The model of a simple link asserts one local resource (which is part of the link), one remote resource, and a relationship between these two resources that makes it possible to traverse the link from the local resource to the remote resource. Since a simple link is inline, it cannot by definition be multidirectional because traversal can be initiated only from the local resource.

Extended Links

XLink's extended links are more powerful and flexible, and their functionality is a superset of that of XLink's simple links (i.e., all that can be done with a simple link can also be done with an extended link). One generalization of extended links is that they can associate any number of resources, as shown in Figure 7.1. In this figure, the Xlink (represented by the oval) associates five resources, three of which are local resources (indicated by the `resource` attribute value, which will be explained in detail in section 7.3). Consequently, this link is an inline link. The XML representation of this link would include an `extended` type element with three `resource` type and two `locator` type children. Since there are no explicit specifications on how the link may be traversed, traversal is possible from and to all participating resources.

One of the advanced features of XLink is that traversal rules for the participating resources can be specified. These rules define the direction in which an XLink can be traversed. Figure 7.2 shows a scenario where the extended XLink of the previous example has been more specifically defined

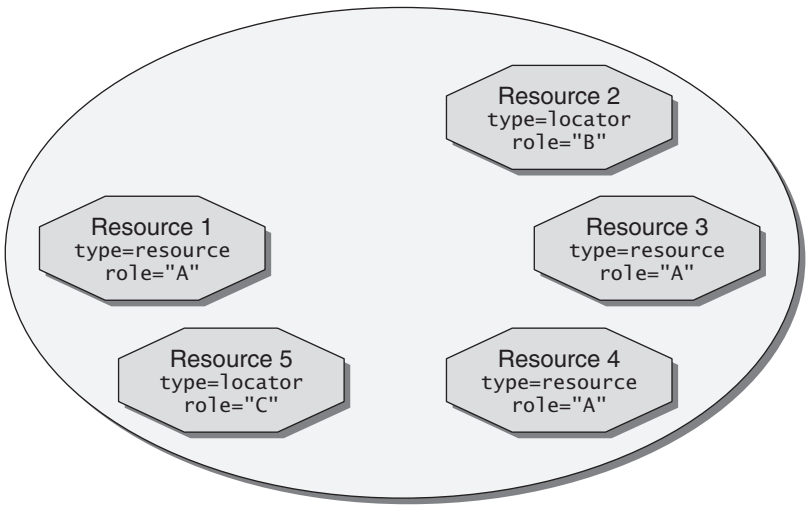

Figure 7.1 Inline extended link

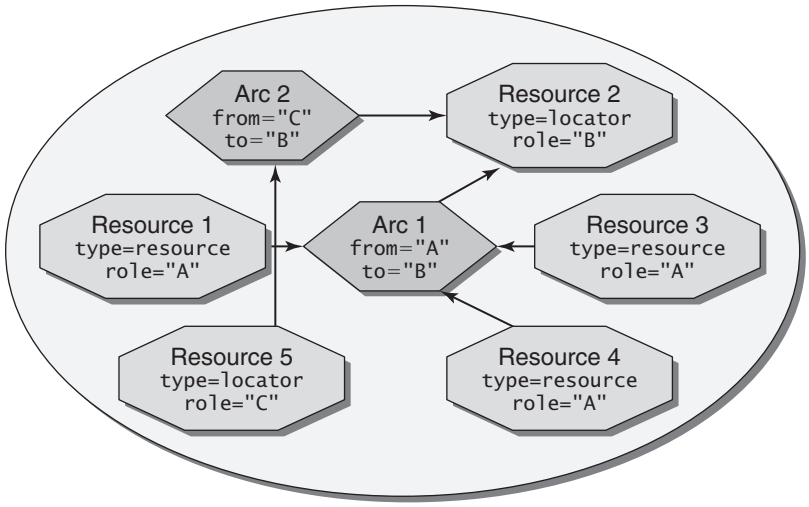

Figure 7.2 Inline extended link with arcs

using two traversal rules.[5] These rules use the `role` attributes of the link's participating resources for specifying the resources between which traversal should be allowed. The XML representation of this link would include an `extended` type element with three `resource` type, two `locator` type, and two `arc` type children. The participating resources of this link are exactly the same as the participating resources of the link in Figure 7.1, but the additional traversal rules define different traversal semantics for this link (for example, it is not possible to traverse this link from Resource 2 to Resource 1 because the traversal rule Arc 1 allows traversal only in the opposite direction).

The previous examples assumed that there are local resources. However, XLink extended links can also be used as out-of-line links, in which case none of the participating resources reside with the link description itself. This scenario is shown in Figure 7.3. In this case, all participating resources are specified as being remote by using the `locator` attribute value. The interesting observation about this link is that it can be created and managed entirely independently from all its participating resources. The XML representation of this link would include an `extended` type element with five `locator` type children. As with our first example of inline links,

[5]In Figures 7.1 through 7.4, we have differentiated types of link information graphically. The link itself is represented by a surrounding oval, resources are shown as octagons, and arcs are depicted by hexagons. These entities are conceptually different, but XLink maps them all to XML element types differentiated by special attributes (described in section 7.3).

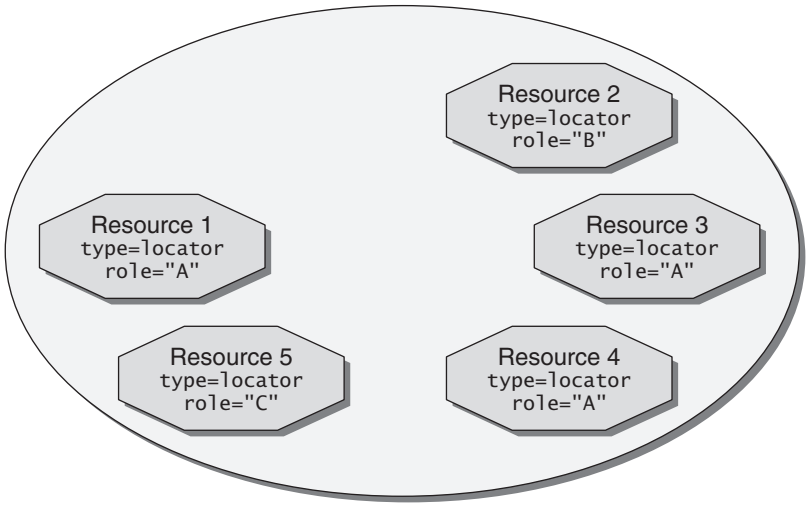

Figure 7.3 Out-of-line extended link

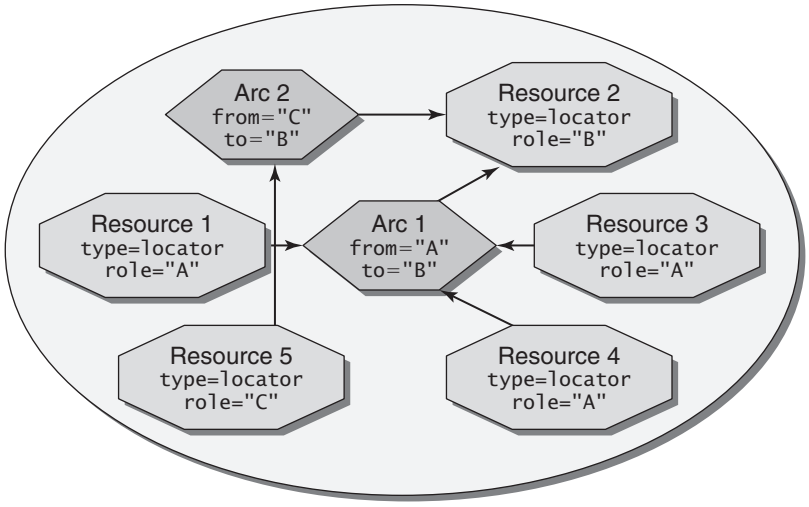

Figure 7.4 Out-of-line extended link with arcs

the first example of out-of-line links does not specify any traversal rules, thereby implicitly allowing the traversal of the link in any direction.

The final example of an extended XLink is an out-of-line link specifying traversal rules, as shown in Figure 7.4. In this example, all participating resources are still specified as being remote; but in addition to the participating resources, the link also specifies rules for the possible traversals.

The XML representation of this link would include an `extended` type element with five `locator` type and two `arc` type children. The rules specified for this example are the same as shown in Figure 7.2, so that these two links are functionally equivalent. However, whereas the link shown in Figure 7.2 has three local participating resources (i.e., resources that are part of the link itself), the link shown in Figure 7.4 does not have any local resources and can therefore be stored and manipulated independently from any of the resources it is linking.

However, this rather conceptual view of XLink's extended links does not explain how extended links are actually expressed in XML. Extended links are described differently from simple links, which are specified using only one XLink element type (the `simple` element type). Specifically, extended links use a number of different element types. In the following section, we take a closer look at XLink's element types.

7.2.2 XLink Element Types

As shown in Table 7.1, XLink represents the two link types using a number of XLink element types. Because these element types make sense only in certain combinations, XLink also defines how XLink element types have to be properly nested. This is shown in Table 7.2, where for each XLink element type it is shown which element types are significant as child elements.

Most importantly, the `extended` element type (shown as an oval throughout the figures in the previous section) represents an extended link but does not directly contain (as an element) all important information. It can therefore be regarded as a container for link information. Much of the information for an extended link is not specified in this element type's attributes but rather in other elements contained in the extended link element. These elements are instances of element types described in the following sections.

Table 7.2 XLink Element Type Relationships

Element Type	Significant Child Element Type(s)	Page
`simple`	none	173
`extended`	`locator`, `arc`, `resource`, `title`	174
`locator`	`title`	178
`arc`	`title`	178
`resource`	none	178
`title`	none	178

Resource

This element type is represented by an octagon in the example figures. In Web terms, a resource is anything that can be addressed via a URI; and typically a link associates several resources. In XLink, resources can be either local or remote. Local resources are specified using the `resource` element type (while remote resources are specified using the `locator` element type). An XLink extended link may contain no local resources (in which case, it is an out-of-line link) but may contain any number of local resources, which all must be specified as `resource` element children of the XLink's `extended` element.

Locators

Remote resources participating in extended links are specified using `locator` elements. In our example figures, these elements are also represented by octagons, because `resource` as well as `locator` elements are used to represents resources. A `locator` element carries almost the same information as a `resource` element with the addition of an attribute for actually locating the resource it represents. This is the most important difference between the `locator` and the `resource` element types: While `locator` elements only represent a resource (essentially by pointing to it), a `resource` element actually contains the resource.

Arcs

A link's most important facets are its resources, but it is also important for applications to be able to specify rules for how link traversal may be used. This is possible using `arc` elements, which specify traversal rules for links. In our figures, these traversal rules are represented by hexagons. Traversal rules specify which resources of a link are starting resources (i.e., from which resources traversal may be initiated) and which resources are ending resources (i.e., resources that may be traversed to). Additionally, it is possible to specify how this traversal of arcs should be done (see section 7.3.4 for details) and how these arcs should be interpreted (see section 7.3.3 for details).

Any resource may be a starting as well as an ending resource, and it is possible to define XLink arcs using arc elements that specify a class of arcs (for an example see Figures 7.2 and 7.4, where the Arc 1 element identifies three starting resources and one ending resource, resulting effectively in three arcs). If no arc is specified for an extended link, it is assumed that traversal may go from any resource of the link to any other resource.

Titles

Titles are specified with the `title` element type. They can be used in different contexts, as shown in Table 7.2. In principle, `title` elements can be used in all contexts where the `title` attribute (as described in section 7.3.3) can

be used (see Table 7.3 for a list of elements),[6] and the purpose of a `title` element is to enable applications to use more complex title content (i.e., XML text with arbitrarily complex structures) than the simple string content possible with the `title` attribute. Whether titles are specified using attributes or elements is entirely up to the link author. The following list shows which XLink element types a `title` element may be a child of:

- *Child of an* `extended` *element.* In this case, the `title` element contains the title of the extended link (as discussed in the "Extended Links" section earlier in this chapter). This title does not describe any individual property of the link, but the link as a whole.
- *Child of a* `locator` *element.* It is possible to describe the title of remote resources (discussed earlier in the "Locators" section) using a `title` element as a child of the `locator` element representing the remote resource.[7]
- *Child of an* `arc` *element.* Arcs between a link's resources (discussed earlier in the "Arcs" section) can also be described with `title` elements containing the arcs' titles. As is the case with all occurrences of `title` elements, it is up to the link creator to decide whether `title` attributes or `title` elements should be used.

Table 7.3 Attribute Use Patterns for XLink Element Types

Attribute Name	simple	extended	locator	arc	resource	title	Page
arcrole	○			○			183
actuate	○			○			184
from				○			188
href	○		•				182
label			○		○		188
role	○	○	○		○		182
show	○			○			185
title	○	○	○		○		183
to				○			188
type	•	•	•	•	•	•	180

[6] The only exception to this is the `simple` element type, which may have a `title` attribute, but not a `title` element type child.

[7] It is not possible to use `title` element type children for the `resource` element (see the "Resource" section on the preceding page). The reason for this asymmetry between remote and local resources is not clear from the XLink specification.

It is important to note that in all cases it is perfectly legal for multiple `title` elements to appear as children of the same parent XLink element, which may be very useful for internationalization and localization purposes. How an XLink application chooses which `title` element to display is outside the scope of the XLink specification, but one obvious approach would be to use XML's `xml:lang` attribute, which identifies languages of element attributes and content by using language tags as defined by Internet RFC 3066 [Alvestrand 01] (see section 7.5.1 for an example).

7.3 ATTRIBUTES

In the previous section, we presented the XLink link types (conceptually, as well as their mapping onto element types); while in section 7.1, we described the basic mechanism of embedding XLinks into XML documents. Now we discuss how to combine these two issues, describing how different aspects of various types of XLinks can be specified in XML documents using particular attributes of the XLink element types.

XLink uses attributes for embedding the link formation in XML documents. XLink defines ten attributes, grouped into five different categories: element type attribute (`type`), locator attribute (`href`), semantic attributes (`role`, `arcrole`, `title`), behavior attributes (`actuate`, `show`), and traversal attributes (`label`, `from`, `to`). These attributes are shown in Table 7.3. In this table, a "•" sign indicates a mandatory attribute,[8] while a "○" sign indicates attributes that are optional. If an attribute occurs on an element but is neither mandatory nor optional for this element type, then it does not convey any XLink semantics.

7.3.1 Element Type Attribute

XLink uses attributes to embed links into XML documents, and one of these, the `type` attribute, assigns XLink semantics to the element on which it appears. It is a very important attribute because it defines which other XLink attributes can be used on a particular element (according to the rules summarized in Table 7.3).

`type`

The `type` attribute determines the XML Linking Language element type of the element that it appears on. Because of the fact that there is a predefined

[8] It is easy to understand why the `type` attribute is mandatory for all XLink element types—because it has to be present to specify the element's type. Without the `type` attribute, it wouldn't be possible to specify an element as being of a certain XLink element type.

set of XLink element types, the `type` attribute may have only the following values:

- `simple` – This value defines an element as being an XLink simple link. Simple links provide the easiest way to use XLink and are very similar in nature to the links provided by HTML. Simple links are described in detail in the "Simple Links" section earlier in this chapter. One important aspect of simple links is that all information relevant for the link is specified in attributes of the simple link element.
- `extended` – XLink extended links are more complex to use than simple links, but are also more powerful. The "Extended Links" section describes their possible uses. It is important to note that most of the information necessary for an extended link is not specified in attributes of the extended link's element but in attributes of other child XLink elements contained in the extended link element.
- `locator`, `arc`, `resource`, `title` – These values are used to assign XLink semantics to elements according to the element types described earlier in the "XLink Element Types" section. These element types may appear only as direct or indirect children of `extended` type elements (according to the relationship shown in Table 7.2).
- `none` – This value can be used to declare explicitly that an element does not have any XLink-specific semantics, so that any attributes appearing on the element or any XLink elements appearing within the element do not have any XLink semantics.

Obviously, the `type` attribute has to be mandatory for all XLink element types because it is the only way an XLink application can identify XLink element types in an XML document.

7.3.2 Locator Attribute

Links connect resources, and therefore one of the most important aspects of a link is the actual identification of these resources. In XLink, resources can be either local (implicitly for simple links, or explicitly by using `resource` type elements as described in the "Resource" section earlier in this chapter) or remote. Remote resources may occur in simple as well as in extended links; and in both cases the same attribute is used, the `href` attribute. In the case of simple links, it is used directly on the `simple` type element (it is optional on `simple` type elements because they may be links with only the local resource); and in the case of extended links, it is used on `locator` type elements (it is mandatory on `locator` type elements because their only purpose is to locate remote resources by reference).

href
This attribute has the same name as the corresponding attribute in HTML's link element, and it serves the same purpose. It is a reference identifying a remote resource; and because XLink is part of the Web architecture, resources must always be identified using a URI reference as described in section 3.2. If the URI points into a resource (i.e., contains a fragment identifier) and this resource is an XML document, then the fragment identifier must be an XPointer.

The `href` attribute is the only way to locate remote resources in XLink, and there is no mechanism for making additional assertions about these resources. If applications are interested in additional functionality—for example, for ensuring link integrity, such as with checksums, cache identifiers, or anything else—it is possible to specify this information in additional attributes specific to the application and not in XLink's namespace.

7.3.3 Semantic Attributes

Not only do links associate resources, but this association also has some meaning. This meaning is, however, application-specific—in other words, there is no predefined set of "link meanings" in XLink. For example, a link for the book you are currently reading may associate resources such as the authors' personal Web pages, the publisher's Web site, a number of online stores selling the book, several Web pages with reviews of the book, and, of course, the book's own Web site. While XLink makes it possible to create such a link, there is no standardized way to describe the actual semantics of the linked resources. XLink follows the path of many Web technologies and defines a way in which semantic information may be specified but does not define a given set of semantics.

Semantics are specified using URI references. Both attributes (`role` and `arcrole`), which carry semantic information that can be interpreted by applications, must be URI references. This reference identifies a resource that describes the intended semantics, but XLink makes no assumption about the actual resource format. In addition to the two machine-readable semantic attributes, there is one that is intended to present semantics in a human-readable way (`title`). The following XLink attributes can be used to specify semantic information:

role
This attribute describes the role that an item plays. It may be used to describe the role of a link (appearing on a `simple` or `extended` element) or the role of a resource (appearing on a `locator` or `resource` element). Even though the `role` attribute has the same name for link (`simple`/`extended`) and resource (`locator`/`resource`) elements, notice that it serves different

purposes. In the first case, it describes the role of the complete link (e.g., that a particular link is describing a book); while in the latter case, it specifies the role of a particular resource within the link (i.e., the "book" link may associate resources describing people, publishers, magazines, and the book itself).

`arcrole`

While the `role` attribute describes the roles of a link and a resource within a link, the `arcrole` attribute describes the role of an arc. In XLink, arcs are represented either by `arc` elements or, implicitly, by a `simple` element; and consequently these are the two element types that may carry an `arcrole` attribute.

In our previous example of the link describing this book, the arcs connecting the various resources of the link may carry `arcrole` attributes that define a "person" resource as being the author of a "book" resource. One resource—for example, a publisher—may have different arcs with different roles attached. For example, the book's publisher may have two arcs going from the "book" resource to the "publisher" resource—one with an `arcrole` attribute for the book's publication; the other with an `arcrole` attribute indicating that the publisher is also running an online store selling the book.

There is one special case of an `arcrole` attribute defined in the XLink specification. This is the special case of a linkbase, where the arc identifies a linkbase for a particular resource. This case is discussed in detail in section 7.5.3.

`title`

While the `role` and the `arcrole` attributes are meant to carry machine-readable role descriptions, the `title` attribute is intended to contain human-readable information about the element on which it appears. It is allowed for all elements that identify links and/or resources (i.e., `simple`, `extended`, `locator`, and `resource` elements).

In XML, attributes can carry only character data—in other words, it is not possible to use structured XML information as an attribute value. If link authors want to create titles that are more than simple character strings (e.g., structured titles or titles in different languages for internationalization purposes), they can instead use the `title` element type (or even use both a `title` attribute and a `title` element), as described in the "Titles" section earlier in this chapter.

XLink's semantic attributes provide a mechanism for link authors to associate semantic information with links or, more specifically, with a link as a whole, its resources, or its arcs. The semantics are entirely application-dependent. XLink's only requires them to be expressed as URI references.

The actual resources behind these references are in no way prescribed or restricted by XLink. Furthermore, if link authors feel that XLink's `role` and `arcrole` attributes are not sufficient for a particular application, they are free to supplement links with their own, non-XLink, semantic information.

7.3.4 Behavior Attributes

A link describes an association between several resources. Generally, applications are expected to do something with links, and as described in section 1.3.2, even today's restricted linking technologies result in varying behaviors, depending on what kind of link is encountered when displaying a Web page (for example, `<A>` links are visually formatted and users may traverse them by clicking on them, while `<IMG>` links are automatically traversed by the browser to embed the image into the formatted Web page). It is not possible to describe all possible behaviors of applications in advance (for example, style sheet processing also uses links, but the actual process of applying a style sheet is much too complex to be described in a generic way), but XLink defines a restricted two-dimensional vocabulary of how applications should behave when traversing a link. Because traversal always implies an arc (links may be traversed only in the ways prescribed by the link's arcs), both attributes may be used only on `arc` or `simple` elements (the latter implicitly defining an arc between the local and the remote resource). The two attributes are described in the following sections.

`actuate`

If an application encounters a link with arcs leading from a starting resource currently being presented to ending resources located elsewhere (for example, a different XML document), it is important to determine the application's behavior. The `actuate` attribute describes the desired timing of link traversal and can have the following values:

- `onLoad` – This value instructs the application to traverse to the ending resource immediately on loading the starting resource (i.e., to perform automatic initial processing). This effect is well-known from HTML's mechanism for using images, as follows:

  ```
  <IMG SRC="http://www.w3.org/Icons/w3c_main" ALT="W3C Logo">
  ```

  In this example, the browser traverses the link when formatting the HTML page. Even if the HTML page contains more than one `<IMG>` element, the browser loads all images, because HTML defines this

behavior as being the default for images. (It can be turned off in most browsers by disabling the automatic image loading.)

- `onRequest` – In many cases, links are not to be followed automatically, but are traversed on request, for example, by user interactions. In case of requested traversal, the application must make sure that a starting resource (i.e., anchor) can be identified (in most browsers, this is achieved by different formatting, for example, underlining and/or color) and that the user can trigger traversal through some kind of interface (in most browsers, a simple mouse click initiates link traversal). HTML's links are a good example for this behavior, as follows:

  ```
  <A HREF="http://transcluding.com">...</A>
  ```

  A link like this will display the `<A>` element's content, formatted in a way that identifies it as a link, and if the user points and clicks on it with the pointer, the browser will traverse the link—in other words, load the target document.
- `other` – The two predefined values for the `actuate` attribute probably cover many application areas, but it is possible that some applications may want to specify alternative (or additional) semantics. XLink therefore supports an `other` value for the `actuate` attribute, which instructs the XLink processor to look for application-specific markup further describing the expected behavior for link traversal.
- `none` – If a link author explicitly wants to specify no specific actuation semantics for the traversal to the ending resource, this can be done using the `none` value, which also implies that there is no markup that is application-specific from which any behavior could be determined.

show

The `show` attribute describes how the result of the link traversal should be presented. The values supported by XLink are as follows:

- `new` – This value indicates that the resource resulting from the traversal should be presented in a new presentation context, for example, a new window. The most popular example for this behavior can be easily expressed in HTML:

  ```
  <A HREF="http://transcluding.com" TARGET="_blank">...</A>
  ```

  A browser traversing this link will open a new window with the target page in it, while the page containing the link will still be visible in its original window.

- `replace` – In this case, the presentation context containing the starting resource (i.e., the resource from which traversal was initiated) should be replaced by the ending resource. This is the default behavior of HTML links as shown in the following example:

  ```
  <A HREF="http://transcluding.com">...</A>
  ```

  A second, more explicitly coded variant is as shown:

  ```
  <A HREF="http://transcluding.com" TARGET="_self">...</A>
  ```

  Clicking on a link like this will instruct the browser to load the target page into the same window as the link, effectively replacing the resource containing the link with the new resource.[9]

  Consequently, a traversal based on `actuate="onLoad"` and `show="replace"` attributes will have an effect similar to automatic forwarding to the presentation context of the ending resource. However, if there is more than one starting resource specifying this behavior, the application behavior is unconstrained by the XLink specification.

- `embed` – In this case, the application is instructed to embed the ending resource into the presentation context of the starting resource, with the exact behavior of embedding being application-dependent. HTML also has an example for this case:

  ```
  <IMG SRC="http://www.w3.org/Icons/w3c_main" ALT="W3C Logo">
  ```

  The browser typically embeds the referenced image into the formatted document (unless image loading is turned off in the browser). XLink's `embed` value implies that the starting resource is replaced by the ending resource.[10] In almost all cases, this is different from the `replace` value, which replaces the entire presentation context containing the starting resource.[11]

  The `embed` value combined with an `actuate="onLoad"` attribute can be used to implement transclusion. XLink does not define how an application should display embedded content, but a reasonable

[9] It is important to note that in this case not only is the starting resource of the link replaced (this would be only the `<A>` element's content) but also the complete presentation context of this resource (i.e., the whole Web page).

[10] Because the `<IMG>` element is always empty, it is not important or is it even apparent that the starting resource (i.e., the `<IMG>` element's empty content) is replaced by the ending resource.

[11] Consequently, if the starting resource is the whole document, then `replace` and `embed` will have the same effect.

implementation should identify the embedded content as coming from another resource and should also provide a way for the user to view the embedded content in its original context.

- `other` – Embedding and replacing in many cases may be sufficient to describe the behavior of links that are used only for creating interlinked presentations. However, links may also imply some kind of processing or other behavior, such as that resulting from HTML's style sheet mechanism:

  ```
  <LINK REL="stylesheet" TYPE="text/css" HREF="general.css">
  ```

  In this case, the browser should not traverse the link to somehow display the ending resource of the link, but instead should retrieve the ending resource (the CSS style sheet) and use it to style the starting resource. This kind of behavior is highly application-specific, and XLink therefore supports an `other` value for the `show` attribute, which instructs the XLink processor to look for markup that is application-specific further describing the expected behavior on link traversal.
- `none` – If a link author explicitly wants to associate no specific behavior with the traversal to the ending resource, this can be done using the `none` value, which also implies that there is no application-specific markup from which any behavior can be determined.

If the starting and/or the ending resource of a link traversal consists of multiple non-contiguous locations, then application behavior is unconstrained. However, application designers either should make sure that resources are always contiguous, or should specify what they expect applications to do in case of non-contiguous locations.

Even though it is fairly simple to explain XLink's behavior attributes using analogies to HTML, the issues get much more complicated when thinking of more complex presentation models such as XML documents formatted by XSL style sheets. In this case, it is easily possible for one contiguous starting resource in the underlying XML document to appear several times in the formatted result (for example, a heading appearing in the table of contents as well as the content itself),[12] and a reasonable application behavior in cases such as this is hard to define. The W3C has noticed this problem and is currently working on a presentation model for XLink [Walsh 01], which we discuss in section 8.1.1.

[12]Note that this case is different from non-contiguous resources, which are explicitly excluded from the `show` attribute's behavior in the XLink specification.

7.3.5 Traversal Attributes

XLink's traversal attributes are used to make the actual connection between a link's arcs (represented by `arc` elements) and a link's resources (represented by `locator` and/or `resource` elements). The mechanism for this is very easy to understand—resources can be labeled with names, and arcs use these names. XLink supports the following attributes:

`label`
This attribute can be used to label resources and may therefore appear on `locator` and/or `resource` elements. Its value must be a valid namespace-compliant XML name. It is not required to be unique within a link (i.e., there can be several resources within one link having the same label).

`from`
The `from` attribute may appear on an `arc` element and specifies the name of the starting resource. There must be at least one resource having a label with that name.

`to`
Complementary to the `from` attribute, the `to` attribute also appears on `arc` elements and specifies the name of the ending resource. There must be at least one resource having a label with that name.

XLink's concept of arcs is described in the "Arcs" section earlier in this chapter. The most important aspect to keep in mind is that one `arc` element can define more than one arc, which is the case if the `from` and/or `to` attribute specify labels that appear on more than one of the link's resources.

7.4 INTERPRETATION OF XLINKS

XLink is a specification that defines a generic way for expressing link information in XML. As such, it is built on top of XML and associated standards, which need to be observed by any application processing XLinks. The next section summarizes the processing requirements that must be met when working with XLink. Furthermore, XLink defines conformance criteria that go beyond the requirements of the underlying XML standards, and section 7.4.2 describes these conformance requirements.

7.4.1 Processing

XLink is built on top of the Extensible Markup Language, and consequently, XML and some of its important companion standards are essential for

correctly processing XLinks. In particular, the following standards must be observed:

- *XML* [Bray+ 00] (see section 4.1) – XML is the foundation of XLink, which specifies information in terms of XML elements and attributes. Consequently, XML defines how elements and attributes are used syntactically within an XML document.
- *XML Namespaces* [Bray+ 99] (see section 4.2) – XLink directly uses XML Namespaces by defining a namespace of its own and expressing all information as attributes within this namespace. For applications to be able to correctly and unambiguously identify XLink information within an XML document, the XML document not only has to be well formed but also has to be compliant with the XML Namespace specification.
- *XML Base* [Marsh 01] (see section 4.3) – Given that XLink is used for linking between resources, it obviously has a strong reliance on references to resources, and these references may be absolute or relative. In the case of relative references, they must be interpreted according to the rules defined by XML Base, which makes it necessary for XLink applications to implement XML Base.
- *URI* [Berners-Lee+ 98; Hinden+ 99] (see section 3.2) – References to resources are always given as URI references, and any application interpreting these references must do so according to the relevant specifications. This does not mean that an XLink application will always be able to retrieve all referenced resources (for example, if a URI reference specifies a scheme not supported by the processing application, then this resource cannot be retrieved), but at least it must be able to correctly process and interpret these references.

Even though XLink specifies all attributes in XML syntax, the standard explicitly states that it is not necessary for XLink applications to work only with XML syntax. If XLink applications instead are set up to work with XML's data model (the XML Infoset [Cowan & Tobin 01], as described in section 4.5), they are allowed to do so. Consequently, it is possible to implement XLink applications that operate on XML Infoset information items and thus never generate or use XML syntax. In this case, XLink's XML syntax would serve only as an exchange syntax and export format to other XLink applications, while internally XLink would be used as a data model.

7.4.2 Conformance

The processing requirements described in the previous section make sure that XLink applications always work with XML syntax and that certain

rules, such as how to interpret relative URI references, are followed. However, XLink also defines constraints not covered by the standards listed in the previous section. These constraints must also be followed by XLink applications. XLink describes conformance on a per-element basis, stating that an XML element conforms to XLink if the following are true:

1. It has a `type` attribute from the XLink namespace whose value is one of `simple`, `extended`, `locator`, `arc`, `resource`, `title`, or `none`.
2. It adheres to the conformance constraints imposed by the chosen XLink element type, as shown in Tables 7.2 and 7.3. While XLink defines that extraneous (i.e., neither mandatory nor optional) elements or attributes are simply ignored (i.e., they do not carry any XLink semantics), it is an error if mandatory elements or attributes are not present.

Applications conform to XLink if they process XML documents or XML Infosets containing conforming XLink elements and observe all rules defined by XLink, for example, the requirement that `from` and `to` attributes of the `arc` elements must use names that appear on at least one of the link's resources.

7.5 USAGE

So far, we have described how XLink can be used to define linking information for XML environments. However, even though this is the foundation on which future XML linking applications will be built, it is also important to have some guidelines for using XLink in an effective way. In this section, we cover some topics that are not essential for XLink from a standards point of view, but that can be very helpful for applying XLink in real-world scenarios.

In particular, in section 7.5.1 we discuss techniques for declaring XLink elements and attributes in schema definitions. Because XLink may be extended for special scenarios where its features are not sufficient, in section 7.5.2 we describe how extensions of XLink can be implemented in schema definitions. Finally, in section 7.5.3 we describe how XLink can be used as the foundation for linkbases and how future applications may use XLink as an export format for huge collections of link information.

7.5.1 XLink Element and Attribute Declaration

XLink defines a number of attributes for embedding link information into XML documents, but it does not make any assumptions about the schema being used for the documents containing XLinks. It is perfectly legal for

XML documents containing XLink information to have no schema at all, in which case they would be well-formed documents (as opposed to valid documents). However, in many cases it is advisable to define schemas when working with XLink, because validating a document may be a useful step in detecting errors as early as possible.

Because XLink makes no assumptions about schemas, it is the author's choice alone as to which schema language to use. The two most popular candidates are DTDs, as defined in the XML specification itself, and XML Schema [Biron & Malhotra 01; Thompson+ 01]. Although we limit our discussion to DTDs (for the sake of brevity), the same principles that we describe for defining DTDs for XLink content also apply to XML Schema (and to any other schema language for XML).

As we have discussed, XLink information is carried only by attributes, even though the `type` attribute also introduces a way to assign XLink semantics to elements. Most applications will probably combine their own data model (which will be entirely application-specific) with XLink's data model to make their data model XLink-enabled. The advantage of this approach is to be able to use existing software for creating, modifying, managing, and presenting XLinks within XML documents. In most cases, application designers will probably want to assign XLink semantics to some of their elements, thus making them links in the XLink sense. This can easily be achieved by defining `#FIXED` attributes for the elements that should have XLink semantics. Consider the following:

```
<!ELEMENT simple ANY >
<!ATTLIST simple
  xmlns:xlink    CDATA       #FIXED "http://www.w3.org/1999/xlink"
  xlink:type     ( simple )  #FIXED "simple"
  xlink:href     CDATA       #REQUIRED
  xlink:role     CDATA       #IMPLIED
  xlink:arcrole  CDATA       #IMPLIED
  xlink:title    CDATA       #IMPLIED
  xlink:show     ( new
                 | replace
                 | embed
                 | other
                 | none )    #IMPLIED
  xlink:actuate  ( onLoad
                 | onRequest
                 | other
                 | none )    #IMPLIED >
```

In this example, the `simple` element type has fixed attributes that declare the namespace[13] and the element's XLink element type (in this case,

[13]If the element is always used in the same document type, then the namespace could simply be declared once on the document element.

it is declared to be a simple link). The other attributes declared for the `simple` element type represent the relevant XLink attributes, as shown in Table 7.3. It should be noted that for some attributes (`title`, `show`, and `actuate`), the allowed values can be specified in the DTD; while for other attributes (`href`, `role`, and `arcrole`), the DTD declaration is too permissive, so that further constraints at the application level are necessary to make sure that the element not only is valid XML but also conforms to XLink.[14] This is a general pattern, showing that some of XLink's constraints can be reflected in a schema while others have to be checked on the application level (i.e., by the application processing the XLinks).

Similar DTD declarations can be made for extended links and the XLink element types associated with them. The XLink specification contains an example of such a declaration.

Two things, which are described next, should always be kept in mind when creating schema definitions for XLink elements and attributes:

1. No schema language today is sufficiently powerful to formally declare all conformance constraints defined by XLink. Different schema languages have different levels of support (for example, URIs have to be declared as simple `CDATA` in DTDs, while XML Schema supports the `anyURI` datatype), though some level of checking on the application level will always be necessary. There are two things to consider:

   - Use a schema language that is as powerful as possible, so that as much checking as possible can be done on the schema level.
   - Use a validation tool that includes XLink support. We currently do not know any XML parser that also validates XLink (or any XLink parser that can be easily put on top of an XML parser); but with the increased support for XLink, software like this will certainly appear.

   Note the following for DTDs as well as for XML Schema: Defining things in a declarative way is always better than writing code, so care should be taken to avoid as much as possible checking that needs to be coded within applications.

2. Even though XLink defines conformance constraints that ensure links can always be interpreted in a meaningful way, applications may

[14] In this case, XML Schema would provide an easy way to already declare the attributes with the appropriate syntactic constraints by using the `anyURI` datatype. Consequently, when XML Schema instead of DTDs is used, applications can rely on stronger validation and thus can be kept smaller and less complex.

choose to be more restrictive and define more constraints. The following are examples:

- For internationalization purposes, applications may choose to generally disallow `title` attributes and instead require as many `title` elements to be present as there are languages to be supported. This requirement could be implemented in a number of different ways, including the following:

```
<!ELEMENT locator  ( title-en, title-de, title* ) >
<!ELEMENT title-en ANY >
<!ATTLIST title-en
  xml:lang         CDATA      #FIXED "en"
  xlink:type       ( title )  #FIXED "title" >
<!ELEMENT title-de ANY >
<!ATTLIST title-de
  xml:lang         CDATA      #FIXED "de"
  xlink:type       ( title )  #FIXED "title" >
<!ELEMENT title    ANY >
<!ATTLIST title
  xml:lang         CDATA      #REQUIRED
  xlink:type       ( title )  #FIXED "title" >
```

  In this example, titles for locators would always have to be present in English and German and would be optional in other languages.

- Applications may restrict local resources to certain types of information and may provide elements to make sure that the local resources always have the expected form, as follows:

```
<!ELEMENT book (title, subtitle, isbn?) >
<!ATTLIST book
  xlink:type   ( resource )  #FIXED "resource" >
  xlink:role   CDATA         #FIXED "http://roles.org/book"
  xlink:title  CDATA         #REQUIRED
  xlink:label  NMTOKEN       #REQUIRED >
```

  In this example, the type as well as the role of the element is fixed, which makes every `book` element an element that is an XLink resource and that plays the role described by the URI reference supplied as the `role` attribute value. The children of the `book` element contain information further describing the book.

This general technique of constraining the XLink application's linking declarations (by designing the schema to specifically use particular attribute values) can make XLink application development much

easier. In particular, the more limitations declaratively specified in the schema, the less work has to be done in the application.

This discussion of what to keep in mind when declaring XLink document types concludes our examination of how to define and use elements and attributes for XLink applications. We have limited our discussion to DTDs, but the general principles are applicable to other schema languages as well. A W3C note [Maler+ 00] is available that specifically discusses how to use XLink in existing document types, but it is limited to XML Schema.

7.5.2 Extending XLink

XLink defines a format for representing links in XML, but it is not the only possible way to define links, and it does not define all possible aspects of link information. For example, an application might want not only to create links to remote resources but also to store these resources in a cache, so that the cached copy is still available if the remote resource itself is unavailable. There are many problems to be solved if caching is to be implemented, and the approach we take here certainly is too simple to be used in practice. It is merely intended to illustrate the issue of how to extend XLink.

A cached resource could be treated as a separate resource and associated with the original resource via a link using a special `arcrole` value indicating that one end of the arc is any resource, while the other end is the cached copy. However, from a modeling point of view, this might be overly cumbersome and create too many links. Also, since a cached copy always is associated with a resource, it could be argued that the caching information should be made part of the remote resource's locator, as follows:

```
<!ELEMENT locator   ( title* ) >
<!ATTLIST locator
   xlink:type       ( locator )  #FIXED "locator"
   xlink:href       CDATA        #REQUIRED
   xlink:role       CDATA        #IMPLIED
   xlink:title      CDATA        #IMPLIED
   xlink:label      NMTOKEN      #REQUIRED
   cache-id         NMTOKEN      #REQUIRED
   cache-timestamp  NMTOKEN      #REQUIRED >
```

In this case, the locator element's declaration is extended by two attributes (which are not from the XLink namespace) describing the caching information for the resource represented by the locator. A locator element like this could be interpreted by the application implementing the caching strategy, which could access the cached copy. However, it could also be used

by any other XLink application, which would ignore the caching information (because the attributes are not from the XLink namespace), but could provide normal access to the remote resource.

This example demonstrates how we can add additional information to suit applications' needs that go beyond XLink's capabilities but that may still benefit from the linking foundations laid by XLink. Another typical example would be to extend the locator element with checksum information so that applications could easily check whether the remote resource had been changed since the checksum was generated. Again, applications supporting the checksum information would benefit from the possibility of detecting changes in remote resources; while basic XLink applications, although failing to notice whether the resource had been changed, could still use the locators.

7.5.3 Using XLink for Linkbases

One of XLink's most interesting features is its ability to create out-of-line links—links that have only remote resources. A collection of out-of-line links is called a *linkbase,* and XLink provides one mechanism for representing this. Linkbases are not a concept introduced by XLink; and even though every linkbase will include XLink concepts, it is not necessarily limited to them. Consequently, linkbases can be built on top of XLink's model but can also extend XLink where appropriate to make them more useful (such as including cache information or checksums, as discussed before in section 7.5.2).

As described in section 7.3.3, the `arcrole` attribute contains semantic information (in the form of an URI reference) about the arc between two resources. XLink defines one special value for this attribute[15] (`http://www.w3.org/1999/xlink/properties/linkbase`), which indicates that the ending resource is a linkbase for the starting resource. Applications processing this kind of arc are expected to retrieve the linkbase (an XML document containing a collection of XLinks) and extract all links from it that are relevant to the starting resource.

The linkbase as a collection of links is an interesting concept that has been widely studied. Linkbases can be used to change the perspective on content and link handling [Wilde & Lowe 00]. In Figure 7.5 we show how information providers can use the concept of linkbases (shown at the storage

[15]This is the only predefined value for semantic attributes in the XLink specification, and the resource behind that URI demonstrates that the specification of semantics often is given only in prose and not in some machine-readable format.

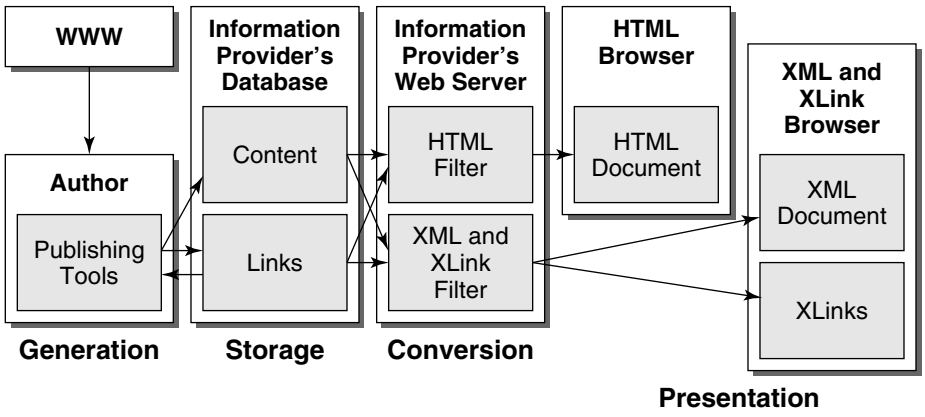

Figure 7.5 XLink and linkbases

level) to create a web of information resources that can be dynamically assembled for presentation. This web is assembled in four stages:

- *Generation.* Authoring tools normally are very content-centric—in other words, they concentrate on the task of creating content. However, when creating content (such as writing articles), authors usually use other interlinked resources (such as Web pages), and authoring tools could be specifically designed to support authors in capturing these interrelationships in the form of links.
- *Storage.* Authoring tools that support capturing link information would require that we not only store the content generated by authors but also store the linking information. On a conceptual level, it is not important exactly how content and links are being stored—whether they use XML-based formats, databases, or other means of storage. The important issue is that we store links separately from content while also ensuring that the content model and the link data model are integrated.
- *Conversion.* While the content is stored in a database or a content management system, the links are kept in a separate linkbase. When we make this information available (in Figure 7.5 this is through a Web server, though many other possibilities also exist), it is necessary to convert the information to a form that can be utilized by appropriate presentation tools.
- *Presentation.* Presentation can be based on very different technologies; but since our focus is highly interlinked information, we assume the use of various Web-based technologies, such as HTML or XML/XLink. The conversion step can be used to adapt

the information to any form necessary, probably using some transformation process such as XSLT.

Even though XLink can be used directly at the presentation level (assuming the browser is XLink-compliant), this is not necessarily required. For example, XLink may simply be used as an exchange format for linking information. Prior to presentation, the XLinks are converted to some form of presentation linking (such as the links in HTML). It would make sense to align the system's internal link data model with XLink if it is foreseeable that XLink will be a popular export format, but that would not be necessary.

Further refining the concept of content management and linkbases, it becomes apparent that where we have a large volume of link information, not only do we need to have an effective form of storage (i.e., a linkbase), but we also need appropriate metadata that describes the links. This is where topic maps [ISO 00; Pepper & Moore 01] come into play. Topic maps are a form of semantic net, and they make it possible not only to express semantic information about resources (such as that one resource describes a particular person and another describes a particular city) but also to associate these information items (for example, this particular person has been born in this particular city). This makes it possible to organize the link information much better than by simply collecting links because more semantic information can be captured.

A simple example can be visited at `http://wildesweb.com/glossary/`, which is a glossary of Web-related terms. The glossary itself is stored as one XML document with a data model similar to topic maps. Starting from this XML document, an XSLT style sheet is applied to generate a set of highly interlinked HTML pages and also a printable version, which is available as a PDF document.[16] Even though this is a small example, it shows the basic steps illustrated in Figure 7.5. At the time of writing, no XML/XLink presentation is available because no browser fully supports these formats, but it would be very easy to extend the conversion process with another XSLT style sheet generating XML/XLink from the original XML document. This glossary example also demonstrates that link information can become very important and that services may exist in the future that offer only linkbase access but no content of their own.

One problem currently unsolved in the linkbase scenario is the access to linkbases. Technically, an XLink processor is required to retrieve the complete linkbase and then use only the links relevant to the resource

[16]The printable version is generated by transforming the XML document into LaTeX(also using an XSLT style sheet) and then producing a PDF document by using the `pdflatex` program.

currently being processed. This obviously is not a reasonable strategy for large linkbases (primarily due to performance considerations), so it is necessary to have a protocol that enables XLink processors to query a linkbase for certain links (e.g., to request all links in which a certain Web page participates as a starting resource). However, this problem is currently unsolved. We hope that in the future standards will emerge that specify how to query XLink linkbases.

7.6 THE FUTURE OF XLINK

XLink is a very new standard, and software products supporting it are only just starting to appear (see section 8.2 for some examples). However, compared to its companion standard, XPointer, XLink is generally accepted; and we hope that future versions of popular software (such as Web browsers) will offer full XLink support. However, even though XLink and XPointer technically are independent, they complement each other; and we hope that the disruptions in XPointer's development will not also stop XLink's success. This remains to be seen; but regardless of their support in popular software, the lessons learned from XPointer and XLink can already be used to implement much better content management systems than are available today.

One political question about XLink's success is also very interesting: Since XLink's new features (in particular, transclusion and out-of-line links) somehow blur the line between different resources, people and companies may have problems with some of XLink's applications and copyright issues. Even HTML with its limited linking abilities has caused a large number of lawsuits against "illegal links" ("deep linking" is one such example, "framing" is another), and XLink opens new doors in this direction, making copyright infringements more likely than today. How companies (and, in particular, software producers) deal with that problem will be one of the main factors governing the fate of XLink.

7.7 CONCLUSIONS

XLink is the language for embedding link information in XML documents. XLink generalizes and extends HTML's linking model and enables users to create complex links. This chapter describes XLink in detail and builds on the foundations laid in chapter 3. XLink will provide the structural fabric of the XML-enabled Web, and the goal of this chapter is to familiarize readers with all the advanced concepts supported by XLink, such as multi-ended links, third-party links, and linkbases.