XML Linking

Steven J. DeRose

Brown University Web: http://www.brown.edu/
Brown University Scholarly Technology Group Web: http://www.stg.brown.edu
Email: Steven_DeRose@Brown.edu

Abstract

The Web Consortium's XML Linking working group is developing specifications to enable more advanced hypertext functionality on the Web: in particular fine-grained anchors, external annotation, and bi-directional links. This paper examines basic goals and approaches; describes HTML linking limitations XML Linking seeks to overcome; and surveys the Working Group's primary specifications: XPath, XPointer, and XLink. As of this writing, the last two, while well advanced, are not final recommendations, and so are subject to change. Consult the W3C Web site for the latest versions.

Background

The HTML tag set and its hypertext element types such as `A` are very useful and popular, yet have difficulties apparent at larger scale and for more diverse data. These include practical matters such as divergence of implementations and mixing of formatting with structure (`I, TT, HR` versus `H1, OL, BODY`); but also more fundamental limitations:

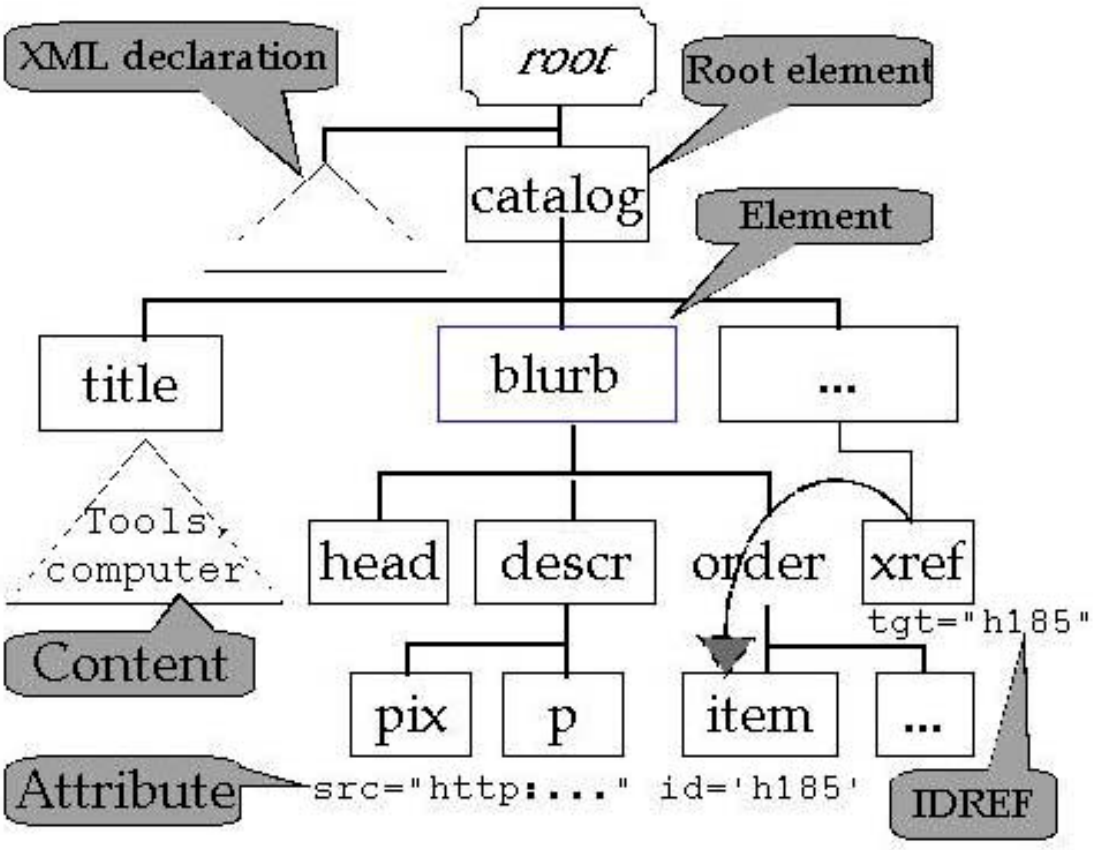
- Element types only of certain kinds, which do not model novel kinds of information (say, `PRICE` for a mail-order catalog).
- Links only of particular kinds (coarse-grained, inline, one-way, and bound up with specific behaviors).

Information modelling

XML permits creating new element types, and trees of them. These can model information structures, moving markup beyond the realm of formatting [Coombs 1987]. This benefits many aspects of document development including hypertext, since documents as structures are processable for more purposes than formatting or printing: retrieval, linguistic and thematic analysis, database interchange, querying, etc. (a list of purposes is less important than the notion of arbitrary processing -- [DeRose 1990]). *Ordered* hierarchies are particularly adept at modelling largely linguistic objects such as documents.

"Generalized", "generic", or "descriptive" markup has been discovered several times, apparently independently. Scribe [Reid 1981] is an early formatter based on structure rather than formatting commands. IBM's GML fed into the SGML standard [ISO 1986]. SGML achieved widespread adoption in high-tech industries and introduced document grammars or schemas (called "Document Type Definitions" or "DTDs") -- HTML, for example, is defined by a DTD. See [Reid 1981] and [Furuta 1992] for more information.

XML [Bray 1998] is a meta-language like SGML, not a specific tag language like HTML. It shares SGML's abstract model while removing abbreviatory features and simplifying parsing. A comparative analysis is [DeRose 1997]. Documents are ordered hierarchies of typed "element" nodes, each type supporting particular named "attributes" (in addition to its hierarchical "content"). Character data and references to non-XML data objects reside at leaves. Intra-document links can use an `ID` attribute type. The W3C "XML Information Set" working draft [Cowan 1999] is formalizing the structure, but an approximate example of an XML document structure is:



This generality is highly valuable as evidenced by XML's rapid adoption; but it also necessitates some way of associating behavior with sets of nodes. Fortunately stylesheets are now widely supported [Bos 1998], [Deach 1999], and so supporting new types such as `PRICE` and `PHONE-NUM` is not problematic. Adding element types with special rendering semantics, such as for equations, may require implementation work (which, however, could in turn be used by other new tags).

XML focuses on elements as classes. Since behavior can be associated with classes rather than just instances, document markup need not be tied to a single behavior. Markup that represents aspects of the nature of information objects (say, "quotation" instead of "single-spaced and indented") facilitates unbounded varieties of other processing. For example, information retrieval can use labels and contexts such as "title", "keyword", or "in an abstract"; voice output can use "quotation" but not "Helvetica"; and most relevant here, structure-aware hypertext tools can benefit greatly from explicit structural information.

Hypertext in HTML and XML Linking

XML linking follows the same philosophy, and aims to bring comparable benefits to the hypertext aspects of the Web. HTML's hypertext facilities have limitations. Some have been worked around via applets, proxy servers, etc. [Gibson 1998a], [Giles 1998], [Hall 1996]. Previous hypertext standards and systems addressed most of these limitations, cf. [Moline 1990]. While not a complete hypermedia specification, model, or language, the XML Linking effort seeks to provide a standard way to overcome several limitations on the Web, particularly in three categories: addressing; the limitation of a closed tag set with fixed semantics; and external/multi-ended links.

Limitations on addressing

Perhaps the most obvious limitation of linking on the Web is the fragility of URLs. Since they identify resources by location, quite minor changes can break them. This problem is being addressed by other efforts (for example URNs [Moats 1997] and DOI [Lynch 1997]), whose work may be combined freely with XML Linking. Davis [Davis 1999] discusses this problem in some detail.

Another problem, however, is that HTML links generally point to entire "resources": most frequently files. This is not a *requirement:* resources are permitted to be "anything that has identity" [Berners-Lee 1998b]. For example some servers transform certain URIs into database queries, and return query results via HTML tables or other structures. Even then, however, there is no standard means of addressing a portion of such a resource.

URIs provide one hook for addressing within resources: a "fragment identifier" appended after "#" to indicate that the user agent (client, browser, etc.) may take some media-type-dependent action after retrieving the full resource. Typically the action is to scroll to the identified fragment and perhaps highlight it. For example,

```
http://www.xyz.com/docs/intro.html#sec27
```

For HTML, the fragment identifier can be any string (XHTML [Pemberton 1999] limits its character set to correspond to XML IDs). It identifies an `A` element by its `NAME` attribute. This is useful but hampered by two non-generalities. First, one can only reference `A` elements, not paragraphs, headings, divisions, or other types, even though many XML elements represent significant rhetorical units that are excellent candidates for anchors. Referring to an `A` placed just within an element is possible, but this misrepresents anchor scope. HTML could be changed to allow `NAME` attributes on all elements, but this would not suffice for non-element selections, such as a point, character, word, or phrase.

The second, more severe problem is that such links can only refer to elements that the author has labelled in advance. This gives authors an unprecedented ability to control what readers can refer to -- a notion absurd in the paper world. It also makes a general annotation system difficult or impossible -- the few Web annotation systems available (such as Microcosm, [Hall 1996] must resort to proxy servers, "transparencies", or similar indirections to overcome this limitation.

XPath and XPointer alleviate these addressing limitations by defining a more general pointing syntax for fragment identifiers that apply to XML resources. XPointer can locate any node, point, or selection in an SGML, HTML, or XML document. Many kinds of data can be and are now being modelled as ordered hierarchies using XML, for which XPointer is expected to be the normative fragment identifier syntax. However, not all data is amenable to XML representations (e.g., bitmaps).

Problems of a closed tag set with fixed semantics

Any closed tag-set will prove insufficient for the variety of document structures that arise in practice. XML itself introduces the possibility of creating new linking elements; it falls to XLink to standardize a way to indicate that a given element *is* a link, to identify certain metadata specific to links as a class, and to provide certain link sub-classes that have proven widely useful. XLink also provides means to identify roles and titles for anchors, and permits authors to associate arbitrary additional metadata with the various parts of links.

The ability to create element types separates link types from link behavior. For example, a certain genre may make use of a "tell me more" kind of link, and define a corresponding element type with particular attributes such as expertise level, degrees of "stretchtext" [Nelson 1987a], etc. One system may express this intent in an expand-in-place interface, while another displays unexpanded text but reads the expansion aloud. Separating structure from particular processing is just as useful for hypertext as for document structures in general [DeRose 1990]. XLink thus provides for identifying which elements represent links.

In HTML, each hypertext element type also has intrinsic behaviors. For example, `A` requires user actuation, and the destination replaces the origin display context (later versions of HTML provide for some exceptions). But `IMG` is automatic (subject, perhaps, to a global user option), and embeds the destination within the origin at the point of reference. `IMG` generally also cannot embed or transclude HTML. The HTML elements conflate several behavioral axes, excluding other combinations. XLink separates two major such axes: whether the link is automatic or user-actuated, and what happens to the display contexts involved (such as open a new context, embed in place, or replace the origin context). These are broad categories of behavior, in contrast to formatting and style issues such as window placements, choice of control styles, and so on. In practice these often represent types of linking more fundamental to the rhetoric of the hypertext, more central to authorial intent, and/or more inherently portable than a detailed specification of processing or formatting.

A significant limitation (not just on the Web) is that no widespread stylesheet mechanism expresses link semantics. CSS [Bos 1998] provides no way to say "elements of type X are links, with a URI found on the Y attribute"; HTML `A` and `IMG` behaviors are thus hard-coded even in CSS-supporting browsers. XSL [Deach 1999] is expected to address some aspects of linking when completed. One of the earliest style languages to support fairly general linking is in the commercial product DynaText [Reynolds 1992]. A general style language maps general categories of link type such as XLink provides into specific behaviors. This mapping might differ for various purposes of platforms, displays, or users. Until such a style mechanism is widespread, XLink's categories can be implemented directly.

External and multi-ended links

Many software systems add some support for linking by embedding one-way link markers at their origin: Links are only "available" from that end, and lead in only one direction. The HTML `A` and `IMG` elements are of precisely this type. Intermedia, a hypermedia system developed at Brown University in the early 1980s, introduced external link databases in which links were separated from the physical location of the anchors they connect [Yankelovich 1985], [Yankelovich 1988]. The Sun Link Service [Pearl 1989] provided a protocol for integrating such systems. Such links are known as "external" or "out of line" links.

Perhaps the largest advantage of external links is that they allow documents to be linked and annotated without modification. Supporting general linking and annotation via embedded links would lead to insurmountable problems:

- Most authors do not wish their documents to be modifiable by everyone.
- Authentication would become vastly more complex.
- Annotators would inevitably corrupt documents accidentally.
- The number of links that would accumulate in important or controversial documents would impose bandwidth, storage, and filtering costs on everyone, even those with no interest in most links.
- With links embedded at one end, the cost of supporting traversal from the other end is very high: to find links attached to one document, the entire document space would need to be searched. Simulating bidirectional links by symmetric pairs of one-way links would require constant and consistent maintenance to remain symmetrical over time.

XLink therefore provides external as well as inline links (following [Sperberg-McQueen 1994], [ISO 1992], and other precedents). This simplifies sharing of link databases. Since there will presumably be too many links and link databases in the world to be of interest to everyone, XLink also provides ways for documents to declare relevant link databases, and envisions other means for users to do so, such as subscriptions or profiles.

The combination of XPath, XPointer, and XLink aims to provide the necessary infrastructure to support fairly general hypermedia functionality on the Web. The following sections provide brief technical overviews of each specification. Note that only XPath is final as of this writing; XPointer and XLink are subject to change.

XPath technical overview

XPath [Clark 1999] provides syntax for addressing fine-grained anchors within XML documents. XPath expressions have multiple uses within XSL stylesheets, but their relevance here is their use in XPointers, which can be attached to the end of a URI to address portions of XML documents.

XPath defines expressions over the structure of a document (not the unparsed XML stream). The W3C XML Information Set Working Group is pinning down details of this structure see [Cowan 1999]. An XPath can locate any element, attribute, or other node in an XML document, though not arbitrary non-node selections for which see XPointer.

Syntactically, an XPath operates over "contexts" to create node sets, using expressions with these constructs:

- **axes** take each node of a given context, and generate new contexts. There are axes that correspond to the basic genetic relationships in ordered trees: child, descendant, parent, ancestor, preceding-sibling, following-sibling, preceding, following, attribute, namespace, and self; plus combined axes for certain common use cases: descendant-or-self and ancestor-or-self).

  For example, if the current context is node X (say, a LIST element), the child axis locates the set of all X's direct subelements (perhaps LISTITEM elements).

- **node-tests** are a special type of predicate, always specified when an axis is used. They can test the general type of node (comment(), text(), processing-instruction(), or just node()), or a specific named type, for example `P` to select paragraphs from an axis that locates element nodes, or `TYPE` to select an attribute from the attribute axis. The node-test `*` matches any node.

  For example, `child::LISTITEM` selects all LISTITEM elements that are children of the current context node, while `child::*` selects all children.

- **predicates** may be appended to location steps, and may use the usual Boolean, comparison, and grouping operators to filter the node list located by the axis and node-test.

  For example, `child::*[self::blockquote or self::quote]` selects all children of the current context node that are either of element type `blockquote` or `quote`.

The combination of an axis, a node-test, and zero or more predicates is called a **location step**. Location steps may be combined via `/`. For example `child::LISTITEM/child::P` would locate all `P` elements that are children of `LISTITEM` elements that are children of the context node(s).

Predicates provide much of the power of XPath. In addition to Boolean, numeric and string constants, functions, and operations, they may include XPath expressions that are evaluated relative to each context node. For example, `child::chapter[child::abstract]` selects chapter children of the context node that have abstracts.

Predicate functions provide information about the state of the filtering operation or about the document in general, or do useful computations.

`Position()` gives the serial position of the node being filtered among all those located by the axis and node-test (always counting outward from the context node); `last()` gives the total number of nodes in the current node list; and `count()` gives the number of nodes located by an embedded XPath expression.

Functions involving the document itself include `root()` to locate the abstract root node of the structure tree (one level above the root *element*), and `id()` to locate element nodes with given ID attribute values. Other functions give access to information about applicable namespaces see [Bray 1999].

XPaths and referential integrity

ID attributes are far more robust than other pointing mechanisms, because they are normally changed only deliberately and specifically. In contrast, geometric measures such as an offset or child number change implicitly. Although one *can* invalidate IDs by explicitly deleting or re-assigning them, with IDs it is also possible (and fairly easy) to manage authoring so such failures do not arise. With geometric measures this is not even possible: one cannot manage editing in such a way that the 300th element within a section always remains the 300th element.

IDs thus remain the most reliable means of locating elements. Nevertheless, it is necessary to be able to locate elements that do not have IDs, as well as non-element locations which cannot have them. XPath can locate nodes even in documents with no IDs at all; but when some IDs are present, XPaths that locate anchors relative to a nearby ID benefit from the stability of the ID.

For example, `id("chap1")/child::ABSTRACT` is likely (though not certain) to continue to point to the abstract within chapter 1 even after editing. This is because the ID immediately constrains the context to a subtree, making the XPath specification safe against edits outside that subtree. Furthermore, `ABSTRACT` makes the expression safe from changes short of insertion of deletion specifically of `ABSTRACT` elements preceding the one originally linked within the same parent element. Other XPath constructs can be used in similar ways: `id("footnote37")/ancestor::DIV`.

XPath cannot ensure referential integrity in the face of arbitrary modification, and mechanisms for authentication remain important. However, well-designed XPaths can greatly reduce the proportion of possible edits that invalidate them, and facilitate usefully approximate attachment even when precise attachment fails. Using IDs as stable starting points for relative locations vastly improves robustness.

XPointer technical overview

XPointer is based largely upon a widely-used technology, the Text Encoding Initiative "extended pointer" [Sperberg-McQueen 1994], [Ide 1995]. Extended pointers provide axes for navigating within trees and a rudimentary predicate language for selecting nodes along axes, and have been implemented in several SGML-based browsing systems.

TEI extended pointers introduced "location terms" including root, here, id, child, descendant, ancestor, previous (sibling), next (sibling), preceding, following, and pattern (content matching by regular expressions) -- very similar to HyTime [ISO 1992], XPointer, and XPath. For example, a TEI extended pointer that locates an element with a certain ID, then walks down the tree structure by choosing successive children by element type and ordinal position, is:

```
id(chap3).child(1,SEC).child(4,LIST)
```

This syntax and semantics formed the primary basis for XPointer. Meanwhile the XSL Working Group developed syntax for choosing what element instances a given style applies to, what elements should be "pulled" or inserted at various points, etc. As the languages converged semantically, the decision was made to develop a common syntax, namely XPath (described above).

XPointer relative to XPath

XPath combines axis and instance functionality of pre-merger drafts of XPointer (see for example [Bray 1997]), with a powerful predicate language from XSL. However, XPath has one primary limitation that makes it insufficient for hypertext linking: it cannot express references to locations that are not entire elements or other nodes; for example, general user selections. A common interface for link creation is "select" / "create anchor"; and users most frequently do not select a whole element or even a sequence of complete elements. Such selections cannot be modelled adequately as nodes.

XPointer's basic node-selection functionality moved into XPath; however, functionality for addressing points and ranges is in XPointer, which is thus needed where modelling user selections (and other non-node phenomena) is required, such as in hypertext systems.

XPointer also adds a small number of constructs such as `here()` and `origin()` functions, for the particular usage scenario XPointer addresses. These constructs refer to the location of the XPointer expression itself (when using it within an XML document), and to the document location from which a user initiated link traversal (in hypertext usage scenarios). Typically they would be used as the starting point for further relative addressing, such as locating the next or previous element regardless of the point of origin.

On modelling selections in XML trees

Selections cannot be acceptably modelled merely as lists of nodes. A point selection is not a node or list of nodes, even assuming that every text character constitutes a node. Stipulating that addressing a single character means addressing the preceding gap would complicate addressing single characters. Selections that cross element boundaries (for example, the last word of one paragraph element, plus some words of the next) pose even more severe problems because some nodes (such as the paragraphs in this example) are neither "in" nor "out" of the selection, but partly "in". And finally, this approach would complicate XPointer semantics by the addition of nested node lists: if an XPointer locates multiple selections (even as an intermediate result), preserving their identities would require lists of node lists, a notion otherwise avoided.

Selections also cannot be acceptably modelled as links between their endpoints. In effect, this would blur the fundamental distinction between addressing and linking, as well as lead to bizarre practical consequences because such a quasi-link would not share most properties of links in general (and could not be embedded as a URI fragment identifier).

A specific mechanism at the XPointer level is required that can address general user selections. The Document Object Model [Wood 1998], [Wood 1999] has found the same need, and developed the "range" and "point". A **location set** where each location may be a node, point, or range, is thus the return type for XPointer; where a node set is the return type for XPath.

XPointer provides two constructs that can locate non-node results: a top-level range expression and `string()`, as well as appropriate operations on them, such as casting, comparison, etc.

The range construct takes two XPointer expressions separated by the keyword `to` (this syntax could change in the final version), and evaluates them relative to the current context node. This results in the selection from the beginning of what the first expression locates, to the end of what the second expression locates. The second expression is interpreted relative to the first. For example, to select the first through third children of the element with ID `a23`,

```
id("a23")/range::child[1] to following-sibling[2]
```

The `string()` function matches a literal string in content, and locates the matched content or a location specified by offset and length from the beginning of the match. Matching is performed without regard to element boundaries, and with each run of white-space treated as a single space. For example, to select the first letter of the third non-overlapping occurrence of the string "Thomas" within the text content of the context node,

```
string::3,"Thomas",1,1
```

XPointer usage

XPointer is intended for use as the "fragment identifier" portion of any URI reference that addresses an XML document see [Berners-Lee 1998b]. It thus provides rules for encapsulating an XPath expression for use as a fragment identifier in a URI reference, and for indicating that the expression is indeed an XPointer (allowing for future extensions or new addressing schemes).

The fragment identifier is appended to a URI after #. XPointer defines three types:

bare names

> are provided for compatibility with HTML, and consist of a single token. While HTML `NAME` attributes are not declared as XML IDs, and may contain whitespace and other non-XML-name characters, XPointer's bare name feature does impose these restrictions, as XHTML is expected to also.

child sequences

> are provided as a very compact and clear way to identify single element nodes. They must begin either with "/1" (to identify the root node) or an ID value (to identify the node with that XML ID), and be followed by a sequence of slash-separated integers, each specifying which child element to choose, working down the tree one level at a time. Of course leading with an ID is far more robust.

full XPointers

> allow the use of the entire XPointer language, as well as for including additional fragment identification syntaxes in a URI reference. A full XPointer consists of any number of `scheme(expr)` groups. The only scheme name defined by XPointer is `xpointer`; other schemes are expected to be introduced for non-XML media types or other applications. The first non-failing scheme group is used.

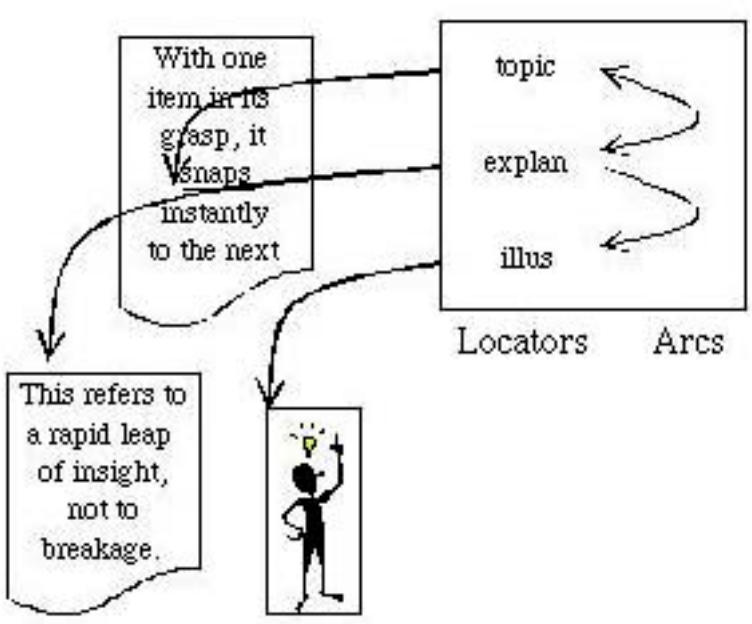
XLink technical overview

XLink is concerned with *linking* itself, as the term is understood in the hypertext community: establishing (non-sequential) connections between data objects. This is distinct from *addressing* mechanisms such as URIs and XPointers: A URI is not properly speaking a link, just as an HREF attribute is not an A element. The XLink specification is intended to enable:

- Identifying linking elements beyond A and IMG;
- Attaching links to documents without modifying them (annotations, "stand-off" tagging, etc.);
- Providing external, symmetrical, and multi-ended links;
- Separating traversal semantics from link type;
- Identifying basic link semantics such as transclusion, expand-in-place, etc.

The biggest single contribution of XLink to the Web may be external links (described earlier), which facilitate link databases and generalized annotation facilities.

XLink's linking model distinguishes anchor, locator, arc, and link. **Anchors** are the actual data being connected by a link: documents, nodes, selections, images, and so on. A **locator** specifies an anchor location, such as via a URI, and may associate information with the location, such as the role it plays in the link relationship. An **arc** connects locator pairs by roles, to indicate the usefulness of traversal (whether by human or program), and may associate information such as traversal semantics. A **link** aggregates the other constructs into a set of a given link-type; it may include multiple locators and arcs to represent multi-ended links with any pattern of traversal between the ends. The following diagram illustrates the structure of an external link.



An XLink link not only connects locations, but describes them and the connections themselves, using machine-interpretable **roles** and human-readable **titles** for link ends, and permitting additional descriptive information as needed.

In certain cases description can be meaningful even without connection. For example a particular link-type or anchor role may assert (somewhat as an attribute may) that something is "wrong" or "an allusion" or "really a PRICE" or "something I want to quote someday". Such an assertion can be made by a link with only one proper anchor. The notion of connecting is not apt because the assertion is abstract rather than a location, and lacks role, title, traversal semantics, and other properties of full-fledged link anchors.

Finally, XLink provides a way to specify a few very broadly useful, very generic link traversal behaviors. Full behavior is expected to be handled via XSL [Deach 1999], but XLink allows specifying:

actuate

: may be `user` or `auto` to indicate whether an arc is to be traversed at user demand or automatically; and

show

: indicates the fate of the originating display context: whether the target is to `replace` it, appear in a `new` window, or `embed` below the source anchor (as for expand-in-place features).

XLink-conforming XML elements are indicated via attributes (for which default values are especially useful). XLink distinguishes **simple** versus **extended** links. The former is much like HTML `A`: its content is one anchor, and link information is expressed with attributes.

```
<mylink xlink:type="simple" href="students.xml"
    role="detail" title="Student List"
    show="replace" actuate="user"
  See also the current List of Students
</mylink
```

An extended link is a container for explicit locator and arc subelements, and is needed for multi-ended, out-of-line, and other more complex links. It may have its own role and title, and provide defaults for show and actuate for its subelements. A locator provides ID, href, role, and title (which may be made a sub-element of its own in order to permit more complex content as needed for internationalization, accessibility, etc.), which together describe one anchor for the link. An arc connects two anchors of the same link by `from` and `to` attributes whose values must be roles of locators in the same link (or "*" as shorthand for all roles). Arcs may specify traversal semantics via `show` and `actuate`.

Finally, XLink provides **extended link groups**, which list documents that may include relevant extended links. In order to discover what parts of a document are anchors of external links, systems need some way of accessing link databases (presumably not *all* link databases in the world will be relevant to all users at all times). Documents may specify relevant link databases; this is valuable because the link database(s) may be modified without touching the document(s) they link. It is also envisioned that users may subscribe to link databases as desired. For example, a user could subscribe to a movie review database, and therefore see its links when viewing pages referenced by its links. System vendors may find it desirable to add more sophisticated subscription mechanisms, such as letting users associate their link database choices with particular XML document types or other metadata: for example, the movie review link database might only be consulted for documents of doctype REVIEW, or documents from certain domains, etc. Subscription mechanisms, however, are not standardized by XLink.

Summary

XML Linking seeks to increase hypertext functionality on the Web by adding standardized support for several capabilities drawn from the broader hypertext field. In particular, the specifications separate addressing from linking more clearly than in HTML, and treat each more generally. XPointer, building on XPath, provides a standard way to locate fine-grained anchors within XML documents; this enables precise annotation and link attachment. XLink then provides a framework for connecting and describing anchors via inline as well as external links, and for assigning significant types of link behaviors (while not attempting to pin down more detailed, application-specific details of behavior).

Bibliography

[Berners-Lee 1998b] Tim J. Berners-Lee, Roy Fielding, and Larry Masinter "Uniform Resource Identifiers (URI): Generic Syntax" in IETF RFC 2396, August, 1998.

[Bos 1998] Bert Bos, Hakon Wium Lie, Chris Lilley, and Ian Jacobs (editors). Cascading Style Sheets, level 2. CSS2 Specification. W3C Recommendation 12-May-1998, Cambridge, Massachusets: World Wide Web Consortium, [Online: http://www.w3.org/TR/REC-CSS2], 1998.

[Bray 1997] Tim Bray and Steven J. DeRose. Extensible Markup Language (XML): Part 2. Linking (Working Draft July-31-97). Cambridge, Massachusets: World Wide Web Consortium. [Online: http://www.w3.org/TR/WD-xml-link-970731], 1997.

[Bray 1998] Tim Bray, Jean Paoli, and C. Michael Sperberg-McQueen (editors). Extensible Markup Language (XML) 1.0, Cambridge, Massachusets: World Wide Web Consortium, [Online: http://www.w3.org/TR/REC-xml], February 1998.

[Bray 1999] Tim Bray, Dave Hollander, and Andrew Layman (editors.). Namespaces in XML. World Wide Web Consortium 14-January-1999, Cambridge, Massachusets: World Wide Web Consortium, [Online: http://www.w3.org/TR/REC-xml-names], 1999.

[Clark 1999] James Clark and Steven J. DeRose. XML Path Language (XPath). Version 1.0. W3C Recommendation 16 November 1999, [Online: http://www.w3.org/TR/xpath], 1999.

[Coombs 1987] James H. Coombs, Allen H. Renear, and Steven J. DeRose. "Markup Systems and the Future of Scholarly Text Processing" in Communications of the Association for Computing Machinery (CACM) 30 (11): 933-947, 1987.

[Cowan 1999] John Cowan and David Megginson (editors). XML Information Set, (Working Draft of May 17, 1999). Cambridge, Massachusets: World Wide Web Consortium, [Online: http://www.w3.org/TR/xml-infoset], 1999.

[Davis 1999] Hugh C. Davis. "Hypertext Link Integrity" in ACM Computing Surveys, Symposium on Hypertext and Hypermedia, 1999.

[Deach 1999] Stephen Deach (editor). Extensible Stylesheet Language (XSL) Specification, (Working Draft 21 Apr 1999). Cambridge, Massachusets: World Wide Web Consortium, [Online: http://www.w3.org/TR/WD-xsl], 1999.

[DeRose 1990] Steven J. DeRose, David G. Durand, Elli Mylonas, and Allen H. Renear. "What is Text, Really?" in Journal of Computing in Higher Education 1 (2): 3-26. 1990. Reprinted with commentary in ACM Journal of Computer Documentation 21(3), August, 1997.

[DeRose 1995] Steven J. DeRose and David G. Durand. "The TEI Hypertext Guidelines." In Text Encoding Initiative: Background and Context. Boston: Kluwer Academic Publishers. ISBN 0-7923-3689-5, 1995.

[DeRose 1997] Steven J. DeRose. The SGML FAQ Book: Understanding the Foundations of SGML and XML. Boston: Kluwer Academic Publishers. ISBN 0-7923-9943-9, 1997

[DeRose 1999] Steven J. DeRose, David Orchard, and Ben Trafford. XML Linking Language (XLink) (Working Draft 26 July 1999). Cambridge, Massachusets: World Wide Web Consortium. [Online: http://www.w3.org/1999/07/WD-xlink-19990726.; http://www.w3.org/TR/WD-xml-link], 1999.

[Furuta 1992] Richard K. Furuta. "Important Papers in the History of Document Preparation Systems: Basic Sources" in Electronic Publishing: Origination, Dissemination & Design 5 (1): 19-44, March 1992.

[Gibson 1998a] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. "Inferring Web Communities from Link Topology" in Proceedings of ACM Hypertext '98, Pittsburgh, PA, 225-234, June 1998.

[Giles 1998] C. Lee Giles, Kurt D. Bollacker and Steve Lawrence "CiteSeer: An Automatic Citation Indexing System" in Proceedings of ACM Digital Libraries '98, Pittsburgh, PA, 89-98, June 1998.

[Halasz 1990] Frank G. Halasz and Mayer D. Schwartz. "The Dexter Hypertext Reference Model" in Proceedings of the Hypertext Standardization Workshop by National Institute of Science and Technology (NIST), January 1990. reprinted in Communications of ACM (CACM), 37(2), 30-39, [Online: http://www.acm.org/pubs/citations/journals/cacm/1994-37-2/p30-halasz/], February 1994.

[Hall 1996] Wendy Hall, Hugh C. Davis, and Gerard Hutchings. Rethinking Hypermedia, The Microcosm Approach. Kluwer Academic, Dordrecht, The Netherlands, 1996.

[Ide 1995] Nancy Ide and Jean Veronis (editors). Text Encoding Initiative: Background and Context, Boston, Kluwer Academic Publishers, ISBN 0792336895, 1995.

[ISO 1986] International Organization for Standardization. 1986. Information Processing-Text and Office Information Systems-Standard Generalized Markup Language. ISO 8879: 1986(E).

[ISO 1992] International Organization for Standardization. 1992. Hypermedia/Time-based Structuring Language: HyTime. ISO/IEC 10744.

[Lynch 1997] Clifford Lynch. "Identifiers and Their Roles In Networked Information Applications" in ARL: A Bimonthly Newsletter of Research Library Issues and Actions, 194,Washington, DC: Association of Research Libraries, [Online: http://www.arl.org/newsltr/194/identifier.html], October 1997.

[Moats 1997] Ryan Moats. URN Syntax, IETF RFC 2141, [Online: ftp://ftp.isi.edu/in-notes/rfc2141.txt and http://www.ietf.org/html.charters/urn-charter.html], May 1997.

[Moline 1990] Judi Moline, Dan Denigni, and Jean Baronas (editors). Proceedings of the Hypertext Standardization Workshop, National Institute of Standards and Technology (NIST). Washington: U.S. Government Printing Office. National Technical Information Service Publication PB90215864, January 1990.

[Nelson 1987a] Theodor Helm Nelson. Computer Lib/Dream Machines (second edition). Redmond, Washington: Tempus Books of Microsoft Press, 1987.

[Pearl 1989] Amy Pearl. "Sun's Link Service: A Protocol for Open Linking" in Proceedings of ACM Hypertext '89, Pittsburgh, PA, 137-146, November 1989.

[Pemberton 1999] Steven Pemberton (editor). XHTML 1.0: The Extensible HyperText Markup Language. A Reformulation of HTML 4.0 in XML 1.0. W3C Working Draft 4th March 1999, Cambridge, Massachusets: World Wide Web Consortium. [Online: http://www.w3.org/TR/WD-html-in-xml], 1999.

[Reid 1981] Brian Reid. Scribe: A Document Specification Language and its Compiler. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA. Also available as Technical Report CMU-CS-81-100, 1981.

[Reynolds 1992] Louis R. Reynolds and Steven J. DeRose. "Electronic Books" in Byte, June 1992: 263-268, 1992.

[Sperberg-McQueen 1994] C. Michael Sperberg-McQueen and Lou Burnard (editors). Guidelines for Electronic Text Encoding and Interchange. Chicago, Oxford: Text Encoding Initiative, [Online: ftp://ftp-tei.uic.edu/pub/tei and http://etext.virginia.edu/TEI.html], 1994.

[Wood 1998] Wood, Lauren (chair). 1998. "Document Object Model (DOM) Level 1 Specification. Version 1.0. W3C Recommendation 1 October, 1998." Cambridge, Massachusets: World Wide Web Consortium. http://www.w3.org/TR/REC-DOM-Level-1

[Wood 1999] Wood, Lauren (chair). 1999. "Document Object Model (DOM) Level 2 Specification. Version 1.0" (Working Draft 23 September, 1999). Cambridge, Massachusets: World Wide Web Consortium. http://www.w3.org/TR/WD-DOM-Level-2/

[Yankelovich 1985] Nicole Yankelovich, Norman K. Meyrowitz, and Andries van Dam. "Reading and Writing the Electronic Book" in IEEE Computer 18, 16-30, October, 1985.

[Yankelovich 1988] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. "Intermedia: The Concept and the Construction of a Seamless Information Environment" in IEEE Computer 21, 81-96, January 1988.