

# XQuery as an Annotation Query Language: a Use Case Analysis

Steve Cassidy

Department of Computing,  
Macquarie University,  
Sydney, Australia  
Steve.Cassidy@mq.edu.au  
<http://www.ics.mq.edu.au/~cassidy>

## Abstract

Recent work has shown that single data model can represent many different kinds of Linguistic annotation. This data model can be expressed equivalently as a directed graph of temporal nodes (Bird and Liberman, Speech Communication, 2000) as a set of intersecting hierarchies (Cassidy and Harrington, Speech Communication, 2000). While some tools are being built to support this data model, there is as yet no query language that can be used to search annotations stored in this way. Since the hierarchical view of annotations has much in common with the XML data model, this paper examines a recent proposal for an XML query language as a candidate annotation query language. The methodology used is a use case analysis. The result of the analysis shows that XQuery provides many useful features particularly when queries include hierarchical constraints but that it is weak in expressing sequential constraints.

## 1. Introduction

Recent work has shown that a single data model can represent many different kinds of Linguistic annotation. This data model can be expressed equivalently as a directed graph of temporal nodes (Bird and Liberman, 2000) or as a group of intersecting hierarchies (Cassidy and Harrington, 2000).

While the data model is now well defined and progress is being made on implementing annotation tools based on the model there is as yet no query language that can be used to search corpora stored in this format. Bird et al. (2000) propose an annotation graph query language which closely follows the underlying graph structure in the data model. While this language appears to have adequate expressive power, some quite simple queries can become difficult to express (Cassidy et al., 2000). Other query languages for annotations exist in systems such as Emu (Cassidy and Harrington, 2000) and MATE (McKelvie et al., 2001); while these may prove useful starting points for an eventual query language design, they are either known to be incomplete or have not yet been shown to provide adequate coverage of all Linguistic annotation requirements.

In a bid to better understand the requirements for an annotation query language, we are collecting a set of representative use cases that might be used in an objective analysis of a query language proposal. This paper illustrates this approach by presenting an evaluation of the XQuery language (Boag et al., 1999) which has been developed to search XML documents. As we will show, XML shares many properties with the annotation data model and so our work can be usefully informed by the efforts of the XML query community.

### 1.1. Data Models

A *data model* is a way of structuring data for access by a computer program. In this case it is a way of storing the annotations on a piece of Linguistic data. A data model should capture the important features of the real data and make them available to programs in a natural way.

The annotation graph data model (Bird and Liberman, 2000) has been proposed as a general purpose data structure for representing Linguistic annotations in computer systems. An annotation graph (AG) is a directed graph whose nodes represent temporal locations (which may or may not be associated with time values) and whose arcs represent annotated regions with associated features. The AG model foregrounds the sequential nature of Linguistic data and annotations and represents hierarchical structures implicitly by having parent arcs span their children (Figure 1).

Another view of this data model is given by the Emu system (Cassidy and Harrington, 2000) which foregrounds the hierarchical structure of an annotation. In the Emu model, an annotation consists a directed graph with annotations as nodes and arcs representing three kinds of relationship: sequence, domination and association (Figure 2). While this appears very different to the AG model we believe them to be isomorphic (Cassidy and Harrington, 2000) and of equal expressive power. The main advantage of the Emu view is that it makes the relationships in the annotation data explicit and corresponds more closely to the conceptual model held by corpus constructors. For this reason, this paper will consider annotations according to the Emu data model. However it is important to emphasise the equivalence with the AG model and so we will refer to Emu/AG annotations throughout the paper.

### 1.2. Relationship to XML

XML (Bray et al., 2000) is a standard markup language developed by the World Wide Web Consortium which is now used in many application areas that require a standardised data annotation format. The XML standard defines a

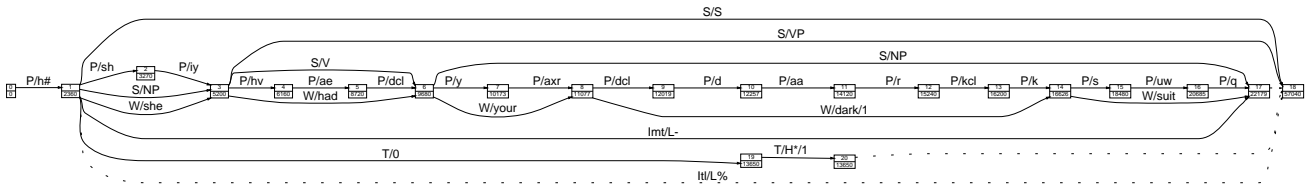


Figure 1: An example annotation graph showing an utterance from the TIMIT database which has been augmented with both a syntactic annotation and a ToBI style intonational annotation. Each anchor (small rectangles) represents a time point and each arc represents a span of time. The labels on each arc consist of a type and a label, so P/sh denotes an /sh/ phoneme. Hierarchical relations are represented here implicitly by parent arcs spanning their children.

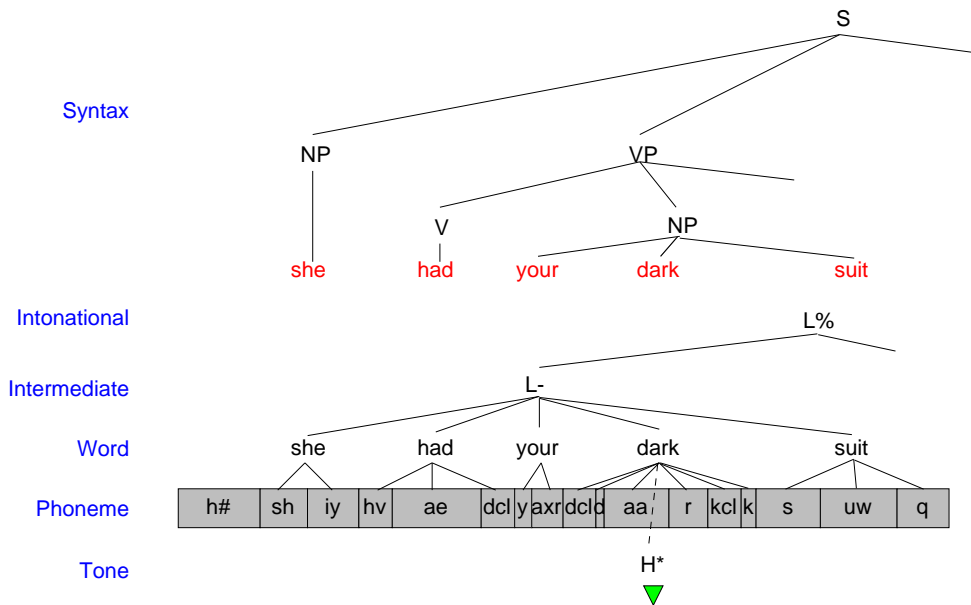


Figure 2: An example Emu annotation showing the same example as Fig 1. The names of the levels are shown on the left, the Word level has been duplicated to show the links to both the syntactic and intonational hierarchies. The single Tone event H\* is associated with the word 'dark'. Time information at the phoneme level is used to derive times for all higher levels.

serialisation syntax (the well known `<tag>...</tag>` syntax) and a data model, known as the XML Information Set (Cowan and Tobin, 2001). In essence, the XML data model consists of a single hierarchy of nodes which can be either the root (document) node, elements or text nodes. Element nodes can have associated attributes with arbitrary values and these can be used to link disparate parts of the hierarchy using a pointer mechanism based on unique attribute names (IDREF).

There is a close correspondence between the XML and Emu/AG data models which is summarised in Table 1. In both cases, the data model consists of hierarchies of nodes which can have associated attributes (called features in AG/Emu). Emu/AG ensures that each node has a defined start and end anchor (which may be associated with a time); these can be modeled in XML with required attributes for each element node. The main difference between the data models is

XML Data Model	Emu Data Model
Nodes: documents, elements with attributes	Nodes: segments with start/end time and attributes
Single hierarchy of elements with unique root node	Multiple hierarchies, many root nodes
Node must have a single parent	Node can have many parents
Nodes are fully ordered (each node has a unique successor)	Nodes are partially ordered (they may overlap and a node may have more than one successor)
Elements have attributes	Nodes have one or more features
Arbitrary elements can be related using IDREFS	Nodes can be related by association relations

Table 1: The relationship between the XML and Emu data models

the requirement for a unique root node and hence a single hierarchy in XML documents. A consequence of this is that multi-hierarchy annotations can't be expressed directly in a single XML document.

This restriction has been overcome by using techniques such as standoff markup (Thompson and McKelvie, 1997) where multiple hierarchies are stored in different XML documents and linked to a source document using external references. Using this technique allows storage of a data model that is largely equivalent to that of Emu/AG.

The obvious motivation for using XML as an annotation format is that there is now a massive effort to develop tools for manipulation of XML in the software industry. These include authoring environments, transformation and presentation tools, query languages and efficient storage and transport mechanisms.

### 1.3. XML Query Languages

One of the stated advantages of XML is that it allows a document designer to make the semantic structure of a document explicit (Berners-Lee et al., 2001). Following this, large collections of XML marked up data have become available and the requirement for a query language to extract information from these documents is clear.

The problem of searching XML documents can be seen as a special case of querying *semi-structured data*: data which has some structure but which is not stored in a fixed format like a relational database (Buneman, 1997). The work to define an XML query language has been driven largely from work in this area. Recently a proposal has been made under the banner of the World Wide Web Consortium for an XML query language called XQuery (Boag et al., 1999) which draws on earlier proposals (Robie et al., 2000; Deutsch et al., 1998) and existing W3C standards.

Queries in XQuery work on XML documents and return a set of document fragments. This provides the important property of compositionality which allows queries to work on the results of earlier queries. A central part of the XQuery standard is the XPath path language which is a W3C standard in its own right (Clark and DeRose, 1999). XPath expressions select nodes in the XML hierarchy based on paths from the root node or some other known point in the document. XQuery expressions can provide additional constraints on the nodes selected by XPath expressions and can then describe the structure of the returned XML fragment which embeds the nodes satisfying all constraints.

Since XQuery is only a proposed standard the language is something of a moving target. The discussion here concentrates on the major features of the language which are likely to be preserved in the final standard. In some cases informed guesses have been made about how some features of the language might work.

### 1.4. Use Case Analysis

In order to evaluate XQuery as an annotation query language we attempt to express some real queries, or use cases, in the language. Note that we don't want to have to first transform the annotation to multiple XML documents in order to do this, rather we'll suggest how XQuery might be extended to be able to cope with intersecting hierarchies in the Emu/AG model.

Use cases are selected to illustrate real Linguistic queries but also to exercise different requirements of a query language. In the larger use case gathering exercise we seek to collect a representative set of queries. The cases described here have been selected to show the strengths and weaknesses of XQuery in particular. All of the queries cited here refer to the style of annotation illustrated in Fig 2.

## 2. Use Cases

### 2.1. Simple XPath Query

The first example query illustrates the basic structure of XQuery and the use of path expressions to locate elements in an annotation.

**Find all vowel phonemes labels A, E, I, O or U spoken by male talkers.**

This is a common kind of query in phonetics research and is well supported in the current Emu query language (Cassidy and Harrington, 2000). The result of the query would be a list of vowel tokens which could then be used to access associated speech data (via their start and end times).

One way of expressing this query in XQuery is shown here:

```
//talker[@sex="male"]//word/  
  phoneme[@label = (A, E, I, O, U)]
```

This query<sup>1</sup> uses an XPath (Clark and DeRose, 1999) path expression to restrict the result to phonemes which are direct children of words (/ denotes a direct link) which are descendants (//) of talkers. The talkers are restricted to those which have a sex attribute of *male* and the phonemes are restricted based on their label attribute. These restrictions are made in XPath by following the element name by additional conditions in square brackets. The result of the query is a collection of phoneme elements which, in XML, would be returned as XML document fragments.

If this were an XML document, the returned values would be the serialisation of the matching nodes and all their attributes and child nodes. Importantly, each match will be a valid XML document in itself which could form the target

---

<sup>1</sup>XPath expressions would normally be written on one line like `//talker//word/phoneme`, they have been split onto multiple lines here to fit into the two-column paper format

for a further query. This is an important property to retain in the annotation domain; hence the result of a query should be a valid annotation. In this case, the result would be a collection of annotations (and associated features and anchors) corresponding to the nodes matching the path expression and this would be a valid annotation.

An alternative return value from a query would be a list of references into the original annotation. This might be useful if later queries or reports wanted to refer to other associations with the matched elements. For example, if after finding the vowels matched by this query we wanted to locate the words in which they appear. This could be supported by retaining unique identifiers for each matching result such that the resulting annotation is a proper subset of the original annotation.

## 2.2. Embedded Query

In some cases it is not possible to express a query in a single clause. XQuery allows embedded queries where the result of one query is passed to another (recall that the result of a query is a well formed XML document). This query illustrates the use of this facility.

### **Find all unique words and return their phonemic transcriptions.**

This query can be expressed in XQuery as follows:

```
FOR $w1 IN
  distinct-values(//word/@label)
RETURN
  <word label={$w1}>
    $w1/parent::word/phoneme
  </word>
```

This query is structured in two parts: the FOR clause defines a variable \$w1 which will take on successive values as returned by the path expression. The RETURN clause specifies the form of the XML data to be returned for each match.

The outer query here finds distinct word labels which are attribute nodes in the XML data model but would be node feature values in the Emu/AG model. The RETURN clause constructs a word element with the same label as the one matched in the FOR clause. The children of this word will be the result of another XPath expression which finds the phoneme children of the word node matched in the outer query. Note that we take care to return words with distinct labels (word types) rather than distinct word elements (word tokens) which would be returned by the path expression `distinct(//word)`.

To locate this word, the path expression `$w1/parent::word` is used. This is a path expression which selects word along the parent axis (rather than the default child axis). In this case the word element is the parent of the attribute node matched in the outer query. XPath includes a number of axes that can be used in path expressions, for example `following-sibling` or `ancestor`. The notation `/word/phoneme` is shorthand for `/child::word/child::phoneme`.

The result of the inner path expression is a set of phoneme nodes. Hence the result of the the whole query is a set of word nodes dominating phoneme children.

## 2.3. Multiple Hierarchies

The primary difference between the XML and Emu/AG data models is the presence of multiple intersecting hierarchies in annotations. This query illustrates the need to be able to traverse these intersecting hierarchies.

### **Find L- Intermediate phrases containing words which form an NP Syntax phrase**

There are two approaches to answering this query. The first method looks for NP phrases phrases which dominate words which all have the same L- Intermediate phrase parent. The second considers these two kinds of phrases as potentially overlapping temporal regions and finds NP syntax regions that are wholly inside L- Intermediate phrases.

The first approach can be stated as:

```
FOR $np in //syntax[@label="NP"]
LET $iparent ::=
  $np//word[1]/parent::inter
WHERE
  EVERY $int IN
    $np//word/parent::inter
  SATISFIES ($int == $iparent) AND
    ($int/@label = "L-")
RETURN
  $iparent
```

This query works by finding NP syntax elements and constraining every intermediate (note: shortened in the example to `inter`) parent of the word children of that NP to be the same as that of the first word – ie. that all the words are siblings relative to the intermediate node. The LET clause is used to establish a variable relative to a node matched in the FOR clause. In this example the variable \$iparent is bound to the intermediate phrase parent of the first word in \$np. The

WHERE clause provides additional constraints on the matched nodes, in this case that every intermediate parent of a word child of `$np` is the same as `$iparent`, the parent of the first word, and has a label "L-". Note that it is not sufficient to constrain the label on the intermediate node to have an L- label since this would allow NPs spanning two adjacent L-phrases to match.

This query could be made more compact with the definition of a new function `sibling` which was true if a set of nodes were siblings relative to another node:

```
FOR $np in //syntax[@label="NP"]
WHERE
    siblings( $np//word,
              $np//word[1]/parent::inter)
RETURN
    $iparent
```

It is interesting to note that the only generalisation of the XQuery model required here is to allow expressions involving the parent axis to return node sets rather than single nodes. Other than this these queries are valid examples of XQuery.

The alternate approach looks at the boundary times of the two kinds of phrase segments. It could be stated as:

```
FOR $np in //syntax[@label="NP"]
    $int in //inter[@label="L-"]
WHERE
    $np/@start >= $int/@start AND
    $np/@end <= $int/@end
RETURN
    $int
```

This solution is similar to that used for the example queries in (Teich et al., 2001) to query annotations stored in XML documents. While this seems to provide a simpler solution, note that depending on the corpus structure, this approach could yield false positives since it does not constrain the two phrases to share any words. If the corpus contains annotations of overlapping speech, this query could return an intermediate phrase by one speaker that overlaps an NP by another.

## 2.4. Sequential Constraints

The previous cases illustrate the suitability of the XPath component of XQuery to specify hierarchical constraints within an annotation, even when multiple hierarchies are involved. A common requirement in Linguistic corpora is to specify sequential patterns which can often involve complex constraints. A good example is the use of a query to find examples of groupings defined by a particular syntactic or phonological construct. This construct might be defined in terms of a regular or context free grammar. Hence the query language needs to be able to express regular or context free constraints. The example cited here is taken from a phonetic study which was interested in syllabic structures defined as sequences of consonants and vowels following a simple regular grammar:

**Find sequences of any number of consonant phonemes followed by a single vowel followed by any number of consonants: C+VC+.**

Such patterns are easily specified in regular expressions on strings but in XQuery we have to work to satisfy this query. While XPath can be used to specify sequences of nodes (using the `following-sibling` axis:) it is not able to state the 'one or more' constraint in the regular expression 'C+'. (This limitation is not isolated to sequential paths, XPath can't apply a constraint like this along any axis.)

The solution in this case is to write functions specific to this query which check for 'one or more consonants' preceding or following a vowel. User defined functions are an integral part of XQuery and this recursive function returns sequences of elements along the `following-sibling` axis which have the label "C":

```
define function fsequence( element $s )
    returns element * {
    let $c := $s/following-sibling::
                phoneme[@label="C"]
    if (empty($c)) then return ()
    else return ($c, fsequence( $c ) )
}
```

A similar function (`psequence`) can be defined to work on the `preceding-sibling` axis and these two functions can then be used to state the query:

```
FOR $v in //phoneme[@label="V"]
LET $pc ::= psequence( $v )
```

```

    $fc ::= fsequence( $v )
WHEN
    $pc != ( ) AND $fc != ( )
RETURN
    <result>
        ( $pc, $v, $fc )
    </result>

```

This query first finds vowel phonemes and then sets up two variables in the LET clause corresponding to C+ sequences before and after the vowel. The WHEN clause tests whether the sequences of consonants are empty and if they are not the RETURN clause constructs a <result> node containing the matched sequence.

While the functions psequence and fsequence might be generalised a little it seems clear that quite a bit of work needs to be done in order to specify patterns of this nature. More complicated conditions would be harder to implement and the fit between the statement of the query and its realisation is very poor for this class of query. In addition, the use of user defined functions in this way prevents any optimisation of these kinds of queries in a database system.

## 2.5. Embedded Clauses

While the previous case tried to express a regular expression like query, this case is concerned with finding instances of a context-free grammar rule. It illustrates a similar class of weakness in XPath to the previous case but along a different axis.

**The context free grammar rule 'NP → Adj NP' can produce embedded NP clauses to arbitrary depth. Find examples of NP clauses where all embedded NPs are of this type except for the final NP which should be a singleton noun.**

This query might match clauses like 'big angry hairy goat' but should reject 'the goat' (which follows the structure 'NP → Det NP'). To answer this query we must check that each NP node in the parent/child sequence has Adj and NP children except for the final one. An XPath expression for a one-level deep clause would be:

```

//syntax[@label='NP' and
    child:syntax[label='Adj']]
/syntax[@label='NP']
/syntax[@label='N']

```

(In fact, this expression would give false positive matches since we don't say that the top NP has *only two* children). This path expression uses a conjunction inside the qualifier for the first path step stating that the syntax node must have an NP label and have a syntax child with an Adj label. The second step states that the NP must have a syntax child labelled NP which has a syntax child labelled N. To extend this to the arbitrary level of embedding required by the use case requires a function similar to that in the previous case to enforce a constraint on a sequence of steps along the child axis.

## 3. Discussion

The examples above illustrate the main features of XQuery and show that it can be applied to the Emu/AG data model without the need for syntactic changes. In particular, the XPath component of XQuery is very well suited to specifying hierarchical constraints in annotation queries and extends easily to intersecting hierarchies if the multiple-parent assumption is made.

The provision of different axes (parent, child, sibling, etc) in XPath is a powerful mechanism which could be usefully extended for annotation queries: one could imagine defining, for example, an 'overlap' axis to allow selection of temporally overlapping nodes.

An important weakness of XQuery is illustrated by the final two use cases which require constraints on a sequence of nodes along an axis. The problem is that an XPath is made up of a sequence of steps along an axis but that here we need to specify conditions on multiple steps. This is most apparent for the sibling axes but might also occur for example along an associative axis or on the parent/child axis. The AGQL query language proposed by Bird et al. (2000) includes this facility for sequential path expressions. The C+VC+ query can be expressed in that language as:

```

select
    F(W).[label:CVC].F(Z)
where
    W.[cons]+.X.[vow].Y.[cons]+.Z

```

(where the details of the conditions in square brackets have been omitted for brevity). This query asks for one or more `cons` arcs between anchors `W` and `X`, a `vow` arc between `X` and `Y` and one or more `cons` arcs between `Y` and `Z`. The combination of this kind of facility with the flexibility to use different axes in XPath would provide a powerful component of an annotation query language.

It is clear that the most natural queries to express in XQuery are those dealing with purely hierarchical constraints. An earlier review of the AG query language proposed by Bird et al. (2000) found that it could express sequential constraints well but was awkward hierarchical queries (Cassidy et al., 2000). These strengths come directly from the forms of the underlying data models: XML is primarily hierarchical, AGs are primarily sequential. Our observations are that the requirements of Linguistic annotation are for a free mixture of sequential and hierarchical structures. A query language combining the strengths of XQuery and the AGQL would go a long way towards satisfying these requirements.

#### 4. References

- Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The Semantic Web. *Scientific American*, May. <http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html>.
- S. Bird and M. Liberman. 2000. A Formal Framework for Linguistics Annotation. *Speech Communication*.
- S. Bird, P. Buneman, and W. C. Tan. 2000. Towards a Query Language for Annotation Graphs. In *Proceedings of LREC 2000*, Athens, Greece.
- Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, Jerome Simeon, and Mugur Stefanescu. 1999. XQuery 1.0: An XML Query Language. W3C Working Draft, November. <http://www.w3.org/TR/xpath>.
- Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. 2000. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, October. <http://www.w3.org/TR/REC-xml>.
- Peter Buneman. 1997. Semistructured data. In *PODS'97*. Invited Tutorial.
- S. Cassidy and J. Harrington. 2000. Multi-level Annotation in the Emu Speech Database Management System. *Speech Communication*, 33:61–77.
- S. Cassidy, P. Welby, J. McGory, and M. Beckman. 2000. Testing the adequacy of query languages against annotated spoken dialog. In *Proceedings of the Speech Science and Technology Conference*, pages 428–433, Canberra, Australia. Australian Speech Science and Technology Association.
- James Clark and Steve DeRose. 1999. XML Path Language (XPath) Version 1.0. W3C Recommendation, November. <http://www.w3.org/TR/xpath>.
- John Cowan and Richard Tobin. 2001. XML Information Set. W3C Recommendation, October. <http://www.w3.org/TR/xml-infoset/>.
- Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. 1998. XML-QL: A Query Language for XML. available as <http://www.w3.org/TR/NOTE-xml-ql>, August. W3C Note.
- D. McKelvie, A. Isard, A. Mengel, M. Grosse, and M. Klien. 2001. The MATE Workbench - an annotation tool for XML coded speech corpora. *Speech Communication*.
- Jonathan Robie, Don Chamberlin, and Daniela Florescu. 2000. Quilt: An XML Query Language. Presented at XMLEurope 2000, July. <http://www.gca.org/papers/xmleurope2000/papers/s08-01.html>.
- E. Teich, S. Hansen, and P Fankhauser. 2001. Representing and querying multi-layer annotated corpora. In S. Bird, P. Buneman, and M. Liberman, editors, *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 228–237. <http://www ldc.upenn.edu/annotation/databases>.
- Henry S. Thompson and David McKelvie. 1997. Hyperlink semantics for standoff markup of read-only documents. In *SGML Europe '97*. <http://www.ltg.ed.ac.uk/~ht/sgmlEU97.html>.