Chapter Three

Representations of Sources and Data: Working with Exceptions to Hierarchy in Historical Documents

Donald Spaeth

In 1990, Steven DeRose et al. proposed that a text is best represented as an 'ordered hierarchy of content objects' (OHCO).[1] This definition has been influential, with one commentator describing this article as the *Principia* of markup studies.[2] The hierarchical model persists in XML, now recommended by the Text Encoding Initiative and the World Wide Web Consortium (W3C). Yet markup specialists have long known of exceptions to the OHCO thesis. Problems of multiple and overlapping hierarchies have attracted the most attention. These may occur because encoders wish to maintain multiple views of a document. But they are also inherent in the text, as in plays where the structures represented by acts, scenes and speeches and by the presentation of the text in verse lines unfold side by side.[3] Examples of multiple and overlapping hierarchies have been found in studies of poetic scansion, semantic

---

1. S. J. DeRose, D. G. Durand, E. Mylonas, and A. Renear, 'What is Text, Really?', *Journal of Computing in Higher Education* 1 (1990), pp. 3-26.

2. P. Caton, 'Markup's Current Imbalance', *Markup Languages: Theory & Practice*, 3 (2001), pp. 1-13.

3. D. T. Barnard, et al., 'SGML-Based Markup for Literary Texts: Two Problems and Some Solutions', *Computers and the Humanities* 22 (1988), pp. 265-276.

structures, scribal variants, and the notebooks of Ludwig Wittgenstein.[4] Fragmented texts, implied and ambiguous data, and cross-references present further problems which violate the hierarchical assumptions of OHCO. This chapter considers instances of such exceptions to hierarchy occurring in historical documents, and explores their implications for the encoding and analysis of data embedded within texts.

Historians who use computers have most often taken a data-oriented view of their sources, rather than a text-oriented view. They have studied well-structured 'roll call' documents, such as census enumerators' books and pollbooks, which take the physical form of tables, or sources such as parish registers which are sufficiently regular to be easily represented as matrices. Less regular sources have been shoehorned into the same table structure. Advocates of source-oriented data processing have long criticised the data-oriented approach, arguing for the need for

---

4. A. Renear, E. Mylonas, and D. Durand, 'Refining Our Notion of What Text Really Is: The Problem of Overlapping Hierarchies', in *Research in Humanities Computing*, vol. 4, eds. S. Hockey and N. Ide (Oxford: Clarendon Press, 1996), pp. 263-77; C. M. Sperberg-McQueen and C. Huitfeldt, 'GODDAG: A Data Structure for Overlapping Hierarchies', in *DDEP-PODDP 2000*, ed. P. King and E. V. Munson, (Lecture Notes in Computer Science 2023; Berlin: Springer, 2004), pp. 139-160; D. T. Barnard, et al., 'Hierarchical Encoding of Text: Technical Problems and SGML solutions,' *Computers and the Humanities*, 29 (1995), pp. 211-231; Text Encoding Initiative, 'SIG:Overlap' <http://www.tei-c.org.uk/wiki/index.php/SIG:Overlap>, accessed 21 May 2006.

specialist historical software, such as Kleio.[5] XML appears to hold considerable promise for historians who wish to integrate data-oriented and source-oriented views of historical documents, permitting data structures to be represented, while remaining true to the form of these documents as texts.[6] Many historical documents, such as depositions and charters, take the form of texts, but nonetheless contain clearly identifiable data elements and are reasonably regular in structure. This article explores one such category of documents, probate records, and particularly the probate inventory, using a small XML database of records from seventeenth-century Thame, in Oxfordshire. This database contains around 300 inventories, listing over 30,000 domestic, agricultural and trade goods.[7]

Unlike bibliographies, invoicing systems, and other domains commonly used to demonstrate how data may be represented in XML, historical documents contain data that are both mixed and semi-structured. Data are mixed when textual

---

5. M. Thaller, 'The Historical Workstation Project', *Computers and the Humanities* 25 (1991), pp. 149-62; *Kleio version 5.1.1* (Queen Mary and Westfield College, 1993); C. Harvey and J. Press, *Databases in Historical Research* (Houndmills: Macmillan, 1996), ch. 7.

6. D. Greenstein and L. Burnard, 'Speaking with One Voice: Encoding Standards and the Prospects for an Integrated Approach to Computing in History', *Computers and the Humanities* 29 (1995), pp. 137-48; D. Greenstein, ed., *Modelling Historical Data* (St. Katharinen: Scripta Mercaturae Verlag, 1991).

7. The database is described in D. A. Spaeth, 'Representing Text as Data: The Analysis of Historical Sources in XML', *Historical Methods* 37 (2004), pp. 73-85. I am grateful to the Thame Research Group for permitting me to use their transcripts of probate records.

information and data elements are intermingled. A document is a text which has data elements embedded within it. Data values may appear anywhere in the text, at any level of the hierarchy, and not just at the 'leaves' which represent the tree's furthest reaches. Besides being inefficient to search, mixed data can also cause difficulties at the analysis stage. Semi-structured data are less predictable than the table structure with which historians using databases and statistical packages are most familiar. Although there are discernible data elements, particular elements may occur at different points of the hierarchy or be omitted entirely. The boundaries of data elements themselves may be ambiguous and uncertain. The depth of the data hierarchy is unknown, since elements may be embedded within other elements. Indeed, the data may not be fully hierarchical.[8] Historical data have all of these characteristics of semi-structured data.

The remainder of this chapter is divided into three parts. First, we will briefly examine the probate inventory as a source, giving examples of its potential uses. Then we will give examples of exceptions to hierarchy, particularly those resulting from implied and fragmented data, and discuss alternative approaches to encoding them. Finally, we will consider the use of XQuery and XPath to interrogate data with hierarchical anomalies. XQuery proves to be a highly powerful tool for dealing with hierarchical and semi-hierarchical data, but its power and flexibility mean that extra care must be taken at the encoding and processing stages to minimise risk of incorrect

---

8. S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From Relations to Semistructured Data and XML* (San Francisco: Morgan Kaufmann, 2004); P. Buneman, 'Semistructured Data', *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (New York: ACM, 1997), pp. 117-21.

results. The goal is to identify markup approaches which permit researchers to interrogate native XML databases without advanced programming skills, just as they would interrogate the structured databases with which they are more familiar.

The Historical Domain

Probate inventories are lists, with values, of the moveable property that individuals owned when they died.[9] Historians have made extensive use of them to study wealth, agriculture, consumption and material culture, among other topics. A number of studies have relied heavily upon quantitative analysis, in which project-oriented methods have dominated over source-oriented data processing. Most studies have extracted data elements and represented them, often in a coded form, in structured databases suitable for quantitative analysis.[10] While this has led to the loss of considerable contextual information, it has meant that these researchers have not had to address directly the complex structure of the documents.

---

[9]. T. Arkell, N. Evans, and N. Goose, eds. *When Death Do Us Part* (Oxford: Leopard's Head, 2000).

[10]. C. Shammas, *The Pre-Industrial Consumer in England and America* (Oxford: Clarendon Press, 1990); L. Weatherill, *Consumer Behaviour and Material Culture, 1660-1760* (Routledge, 1988); A. H. Jones, *Wealth of a Nation To Be* (New York: Columbia University Press, 1980). For an exception, see M. Overton, 'Computer Analysis of an Inconsistent Data Source: The Case of Probate Inventories', *Journal of Historical Geography* 3 (1977), pp. 317-26; M. Overton, et al., *Production and Consumption in English Households, 1600-1750* (Routledge, 2004).

An inventory normally takes the form of a hierarchy. After an introductory paragraph, which records the date and the names of the deceased and the appraisers, a typical example lists the contents of the house, room by room, as well as belongings found in outhouses, the shop, the yard, and surrounding fields. Goods are grouped into 'items' (from the Latin for 'also'), each of which is assigned a value. The document's structure, in which inventories contain rooms, rooms contain items, and items contain objects and values, can be represented as a data tree or graph (see Figure. 3.1). Tables 3.1 and 3.2 demonstrate the kinds of information which can be extracted from inventories. Details of room contents allow the organisation of houses and the functions of particular rooms to be studied. A full-text database of inventories also can be used to study how appraisers represented household contents. The grouping of objects into items, and even the presence of room details, reflected choices made by appraisers.[11] An accurate picture of the reality and representation of houses depends upon the encoding system adopted and upon the degree of correspondence between the representation found in the document and the physical layout of the house.

---

[11]. I am currently prepared an article on appraising its intellectual implications, with the provisional title, 'The Middling Sort Make Sense of the World of Goods: Identification and Differentiation in Seventeenth-Century England'.

Figure 3.1: A data tree for a probate inventory

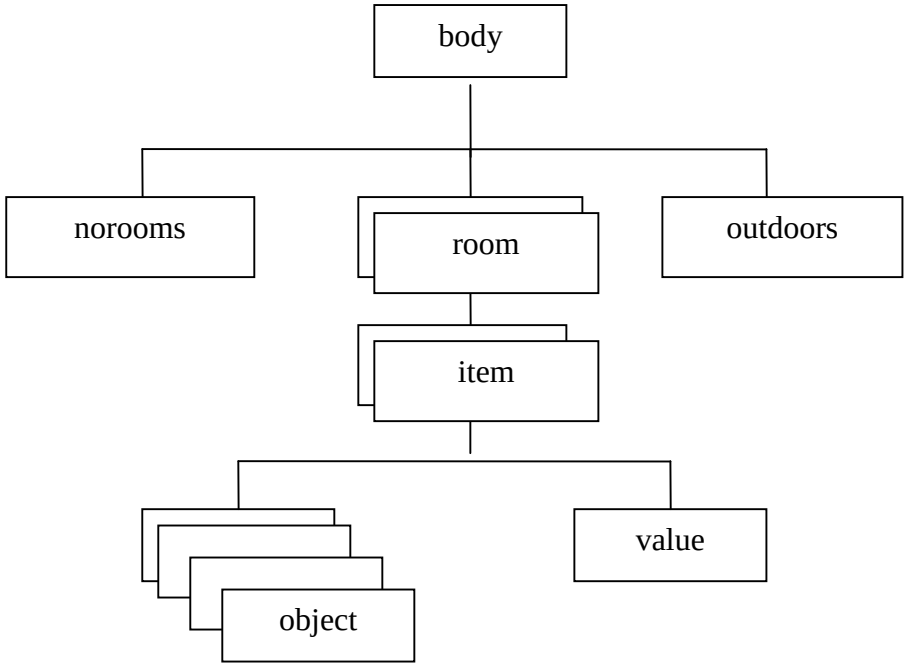

Table 3.1: Percentage of halls with hearth

	1600-24	1625-49	1650-74	1675-99
% of houses with at least one hall	95	92	90	68
% of halls with hearth	68	59	64	56
Total number of inventories listing rooms	58	62	48	38

Source: Spaeth 2004.

Table 3.2: Choices made by Thame appraisers

	Rooms? (by invent)	>1 items (by room)	Counting (by object)	Descriptors (by object)	Lumber (by item)
Thame	68%	43%	34%	21%	10%
Panel	69%	42%	38%	21%	7%
Trinder	85%	51%	43%	13%	3%
Stone	86%	57%	37%	21%	9%
Louch	100%	48%	42%	16%	8%
Jemett	73%	57%	36%	19%	7%
Parslow	38%	33%	32%	35%	9%
Peck	89%	15%	40%	22%	23%

Source: Thame XML Database.

Markup

XML represents a text as an ordered hierarchy of content objects, known as elements. An element may contain data values (PCDATA) and child elements, so long as each child element is fully contained within its parent and siblings do not overlap. The end tag which represents the completion of a child element must occur before the end tag which marks the end of its parent. Child elements may, in turn, contain their own children, which are grandchildren of the original parent. This simple system provides an elegant means of representing data. A relational table may be represented by viewing an element as the container for a table, the element's children as records, its grandchildren as fields, and the data values within the grandchildren as the field values. This structure is at least partially self-documenting because the names of the elements in the hierarchy represent the table, record and fieldnames; exceptions require further documentation, however. Restrictions can be placed upon the number and order of values using a schema (such as a DTD or an XML Schema), although it is advisable to keep restrictions upon semi-structured data to a minimum.

There are two alternative ways to represent data in XML. A table-based approach imposes a structure upon the data by preparing a template of elements in advance, into which the data are transcribed. The order and structure of elements are fixed. This approach is often used in published examples of XML databases, which have been written for database specialists who wish to represent legacy or new data in XML.[12] It is little different than entering data into a conventional relational database.

---

[12]. E.g., W3C, 'XML Query Use Cases: W3C Working Draft, 15 Sep. 2005' <http://www.w3.org/TR/xquery-use-cases/>. See also the useful discussion in R.

The logical view of the data is allowed to take precedence, and aspects of the physical representation of the data which do not align with this logical view are ignored. This approach raises no hierarchical difficulties and will therefore receive no further attention here. Alternatively, a text-based approach exploits the hierarchical nature of XML to represent directly the hierarchies found in the documents. In a purely source-oriented model, the transcribed text is preserved unaltered, without modification, addition or omission. XML elements are inserted into the file to make the data structures within the text explicit. This approach creates a mixed data file, with text present at all levels, including words (such as 'Imprimis' and 'Item') which are structural signposts rather than data. In addition to the data found in the text, data such as identification numbers, normalised forms and classifications, if needed, may be introduced as attributes. The text-based approach has the advantage that, if the markup is later stripped from the file, the original text remains intact. It is nonetheless recognised that the database is only a representation of the source and that some questions, especially those that concern the physical appearance of the document, may require the researcher to view the original source. However, the text-based approach relies upon the alignment of logical and physical views of the data. Hierarchical exceptions, in which data elements do not appear in the source where expected, will cause problems for analysis.

Bourlet, 'XML and Databases' (updated September 2005) <http://www.rpbourret.com/xml/XMLAndDatabases.htm>.

Figure 3.2. An encoded inventory

```
<inventory>
The inventory of the goods of <deceased><name>John Small</name></deceased>
prepared on <date>14 April 1640</date>
<body>Li s d Imprimis <room name="hall">In the hall
<item>
two <object quantity="2">joined tables</object> and
six <object quantity="6">joined stools</object>
appraised at <value>12s.</value></item></room>
<room name="chamber">In the chamber  <!-- etc.  --></room>
</body>
</inventory>
```

A simplified encoded inventory is shown in Figure. 3.2. This hierarchy may be represented schematically as follows, in which elements which may repeat are starred:

inventory*
    body
        room*
            item*
                object*
                value

Although most inventories follow this structure, there are exceptions. Levels of the hierarchy may be omitted entirely, or it may be unclear where a level begins and ends. In Thame, appraisers chose not to organise items by room location in one-third of inventories. Even if they did list by room, most inventories contain some objects whose exact location in the house is unclear. These are instances of semi-structured or ambiguous data rather than serious exceptions to hierarchy, and are easily resolved by introducing virtual rooms, such as <norooms> and <other> which take the place of <room> in the hierarchy. A more serious structural problem occurs when data values are not explicitly stated, although they are implied by the syntax of the text. Instances of implied data are common. For example, an inventory may record that goods were appraised by 'Steven Cooke and John Springall yeomen'. Similarly, a group of domestic goods may be covered by an overall description at the start, such as 'All of the pewter, at 7d per pound, two dishes, three bowls, a candlestick'. The first example leaves no doubt that both men enjoyed yeoman status (a middling farmer, below gentleman), but a markup system which relies solely on the text to provide the data will be missing a status for Cooke. Attributes provide an easy solution, but a more elegant alternative might be to code the linguistic structure explicitly, by introducing an additional level of the hierarchy, which serves as a wrapper (as in Figure. 3.3).

Figure 3.3. Dealing with implied data

a. Cooke lacks status

<app><name>Steven Cooke</name></app>

<app><name>John Springall</name> <status>yeomen</status></app>

b. Adding a group wrapper

<group type=”app” status=”yeoman”>

<app><name>Steven Cooke</name></app>

<app><name>John Springall</name> <status>yeomen</status></app>

</group>

Non-adjacent text presents a more serious exception, since it means that the hierarchy is fragmented. This may occur when one sibling is interrupted by another, before it has itself finished. The inventory of Robert Maund provides a good example.[13] (The inventory has been simplified and line numbers have been added for readability.) (See Figure. 3.4)

[13]. Oxfordshire Record Office, PEC 46/2/26.

Figure 3.4. An inventory with fragmented hierarchy

Inventory of Robert Maund, taken 16 April 1660

1. In the hall [etc.]
2. In the parlour [etc.]
3. In the **kitchen**, a knife, tubs, barrels, spinning wheels [etc.], 1Li 15s
4. In the brewhouse [etc.]
5. In the chamber over the parlour [etc.]
6. In the chamber over the hall [etc.]
7. In the chamber over the kitchen, 2 bedsteads, 2 trunks [etc.], 1Li 1s
8. Upon the bedsteads in these rooms, 2 featherbeds, 5 feather pillows [etc.], 5Li 15s.
9. More in the **kitchen**, 12 pewter platters, 1 pewter bason [etc.], 2Li 10s
10. In the chamber over the kitchen, 9 pairs of sheets, 2 pillowberes [etc.], 5Li 2s.

The data model assumes that all of the goods in a room will be listed together. Yet in this inventory the contents of two rooms -- the kitchen and chamber over the kitchen -- are fragmented. To complicate matters further, the bedding from three chambers (lines 5-7) is listed together in a virtual room (line 8), separate from the other objects in these rooms. A strict source-based markup system fails because the logical and physical structures of the document are not aligned with one another. To put the problem differently, the representation of the house constructed by the appraisers differs both from the physical layout of the house and from the logical structure of the

data. Implementation of a simple markup system based upon the expected hierarchy would lead each part-room to be encoded as if it were a complete room, with the consequence that an extra kitchen and chamber would be added to the house. A count of rooms would find three more than were physically present in the house, while understating the number of objects in each room.

It must be stressed that, although not unique, this inventory is a relatively uncommon exception to the normal hierarchy. Unlike better-known examples of multiple hierarchies in literary texts, only one view of the data is valid in most of the database. Only in a few cases do the logical and physical structures represent different views of the same document, and these apply only to a section of this document. An encoder may therefore be reluctant to impose a more complex system of markup on the data, when this will be needed only in exceptional cases. Set against this must be the likelihood that the presence of exceptions will complicate analysis of the data.

There are several solutions to the problem of fragmented hierarchy, each of which has implications for the analysis of the marked up data.[14] We have already set aside encoding the rooms precisely as they occur in the source, since treating fragmented rooms as if they were complete will produce misleading results. Yet, we would be equally reluctant to adjust the text so that it conforms to the expected logical structure. This would conflict with the source-oriented ethos and obscure the representation chosen by appraisers. Another option is to mark up the *hierarchy* exactly as it appears in the text, so that rooms are literally located within other rooms,

---

[14]. Semantic web technologies may provide an alternative solution, by allowing the definition of multiple ontologies. These technologies, and their application to historical documents, are discussed in ch. 4, below.

even though this violates the expectation that rooms are siblings. The chambers are then encoded as children of the kitchen. Although unorthodox, this has the advantage of recording the structure of the original text, and perhaps also the physical construction of the house and its representation. Chambers were sometimes added above existing rooms, so there is a sense in which they can be regarded as contained within them. Reading the inventory, we can see that appraisers went from the kitchen to the brewhouse, then upstairs to the chambers, finishing with the chamber over the kitchen, before returning downstairs to the kitchen, where it appears that they discovered goods that they had missed the first time. It is likely that there was a door between the kitchen and the brewhouse, and that there was at least one staircase, perhaps leading from the brewhouse. It is not entirely clear why the appraisers returned to the chamber over the kitchen; perhaps a second pass through the chambers to list the bedding led them to discover more objects. Using markup to embed some rooms in others may therefore reveal historical features of the inventory, even though it fails to conform to the logical model. As tempting as this option may be, nesting elements which logically are siblings can produce unintended consequences for analysis, so that objects located within the inner rooms may be counted twice. It would not work with this inventory, in any case, because the chamber over the kitchen is also fragmented.

This leaves three different approaches for more detailed consideration; these are summarised in Figure. 3.5. The first two approaches give one view of the data priority, but add tags which enable the other view to be re-constructed, if necessary. It is assumed here that the source view is given priority, but this is not essential. The

first approach follows TEI's recommendations for encoding multiple hierarchies.[15] An ID attribute is added to each of the affected room elements, and the <join> element is then used to put the parts together, using the target attribute to locate the IDs. The room fragments are encoded as <partroom> elements so that an XPath search for <room> fragments does not mistake the fragments for complete rooms. A <join> element is then used to stitch together the part-rooms. At present, a schema is required to define the ID attribute, but work is underway to remove this requirement.[16] Approach 2 is similar but is intended to be more intuitive. A cross-reference is inserted where the data should appear logically, enabling it to be fetched from its position in the document. The detached fragment is again 'hidden', in a <detachedroom> element, so that it is not mistakenly treated as a room in its own right. Since this approach also relies upon an ID attribute, a schema is needed.

---

[15]. C. M. Sperberg-McQueen and L. Burnard, eds., *TEI P4: Guidelines for Electronic Text Encoding and Interchan*ge *[XML Version]* (Oxford: Text Encoding Initiative Consortium, 2002) <http://www.tei-c.org/Guidelines2/index.html>, ch. 31.

[16] W3C, 'xml:id Version 1.0: W3C Recommendation 9 September 2005' <http://www.w3.org/TR/xml-id/>.

Figure 3.5: Three approaches to encoding fragmented hierarchy

---

3.5(a) ***Approach 1***

<partroom name=”kitchen” id=”r1”> ... *data* ... </partroom>

... *other rooms* ...

<partroom id=”r2”> ... *data* ... </partroom>

<join result=”room” target=”r1 r2”/>

3.5(b) ***Approach 2***

<room name=”kitchen”>

... *items* ...

  <item target=”i1”/>

</room>

... *other rooms* ...

<detachedroom id=”i1”> ... *data* ... </detachedroom>

3.5(c) ***Approach 3***

<room name=”kitchen” status=”fragment” part=”1”>

  <item> ... *data* ... </item>

  <item view=”data”> ... *data* ... </item></room>

<room name=”chamber”> ... *data* ... </room>

<room name=”kitchen” status=”fragment” part=”2”>

  <item view=”source”> ... *data* ... </item></room>

In the first two approach, the text fragments occur only once in the encoded document. The third approach is simply to encode both views, so that the problematic

text appears twice, once in its original position and again in its logical position. Attributes are then used to label the two views as either 'data' or 'source', depending upon which view is given priority. This approach is the only one to change the original text; if the markup is removed, two instances of the detached fragment will appear. It goes without saying that, whichever approach is adopted, proper documentation is essential. The markup approach which is chosen will clearly have implications for data analysis, so that simple queries based on an incomplete understanding of the data structures will produce unexpected results.

Processing

XML data can be interrogated using the XPath, XQuery, and XSLT languages developed by the W3C. XPath and XSLT have been W3C recommendations since 1999; at this writing XQuery is still only at the candidate recommendation stage, but several implementations are available.[17] XQuery may be thought of as SQL for XML data. It offers many of the features one would expect of a relational database, although its lack of an update facility leaves XSLT as the preferred tool for

---

[17]. W3C, 'XQuery 1.0: An XML Query Language: W3C Candidate Recommendation 3 November 2005'; 'XSL Transformations (XSLT) Version 1.0: W3C Recommendation 16 November 1999'; 'XML Path Language (XPath) Version 1.0: W3C Recommendation 16 November 1999'; all at <http://www.w3.org/>. The queries in this article were tested using the open-source version of 'The Saxon XSLT and XQuery Processor', Saxon SB-8.7.1, developed by Michael Kay, downloaded from http://sourceforge.net/project/showfiles.php?group_id=29872 [accessed 21 May 2006].

transforming data. XSLT is particularly good at modifying particular parts of the data structure, even when the data are irregular, because it is event-driven and recursive rather then procedural. Templates are used to retrieve and change only those elements which match specified patterns, rather like a mechanic removing and replacing parts. Both XQuery and XSLT rely upon XPath to select particular 'nodes' from the data hierarchy and to navigate between nodes.[18]

Although there are some similarities between XQuery and SQL, those familiar with traditional databases will find the experience of querying XML data is very different. The relational model is relatively simple. In XML, as we have seen, the same data can be represented in several ways, and this flexibility complicates analysis, even with regular data. Complex markup makes matters worse. It may be necessary to look in several places for the same data; if the data are fragmented, then they need to be re-assembled before analysis can continue. This implies a two-stage process, in which the data are first assembled, probably using XSLT, producing a working file which has the expected logical structure. Alternatively, one can interrogate the data directly, using XQuery to build up complex queries, which perform multiple passes through the same data. A single query may include several XPath expressions, each of which passes through the entire data or a defined subset. As in SQL, the results of one query may immediately be used as input to another. There is, however, a price to pay for such flexibility; queries can be both difficult to construct and slow to process.

---

[18]. M. Brundage, *XQuery: The XML Query Language* (Boston: Addison-Wesley, 2004); M. Kay, *XSLT Programmer's Reference*, 2d ed. (Birmingham: Wrox, 2001), 75-81; W3C, 'XQuery Update Facility, W3C Working Draft, 8 May 2006' <http://www.w3.org/TR/xqupdate/>.

XPath handles semi-structured data, in which elements may appear in unexpected places, well, using axes. The descendant axis will retrieve specified elements anywhere they occur in the hierarchy. The ancestor axis can then be used to climb back up the tree as far as needed to retrieve contextual information. Thus, the expression

doc(“example.xml”)//room[@name=”kitchen”]//object

will retrieve all objects found in kitchens. (The double-slash is a shortcut for the descendant axis.) However, the power of the descendant axis carries its own risks, for one can easily retrieve elements that one does not want, because they share the same name. This drawback can be addressed by using attribute values or parent element names to contextualise, and thus refine, a search. Nonetheless, care must be taken when marking up the data, to ensure that one will be able to retrieve all of the data one wants, but no more. The element name is itself data in a way in which a field name in a relational database is not.

Data are retrieved from an XML database by using either an XPath expression on its own or a FLOWR (or ‘flower’) expression, seeded by XPath, to iterate through the data. XPath on its own is quite limited, so FLOWR expressions are needed to re-order, join, and present results. However, there are important differences between XPath and FLOWR in the way they handle data. A crucial feature of XPath 1.0 is that it removes duplicate nodes, namely those which not only have the same name and value, but also have the same identity, essentially the same position in the data file. FLOWR expressions do not remove duplicates, however. It is for this reason that embedding one room in another, as suggested above, produces unpredictable results.

The objects in the embedded room will be counted once if retrieved using XPath, but in some contexts will be counted twice if a FLOWR expression is used.

What implications do the three markup approaches that I outlined earlier have for retrieval using XPath/XQuery? In each case, it is possible to devise a single XQuery which brings together the room fragments and treats the kitchen as a single room. However, it is easier to query the third approach than the other two, as can be seen from the examples in Figure. 3.6. In 3.6(a), a fragmented room encoded with a join (approach 1) is stuck back together again. Queries 3.6(b) and 3.6(c) count the number of objects per room, depending upon whether a source-oriented or data-oriented view is adopted.

Figure 3.6. XQuery solutions

3.6(a). Joining room fragments

```
let $root := doc(“test.xml”)
for $join in $root//join
let $result := $root/id(data($join[@result=”room”]/@target))/*
let $merge :=
<room>{ attribute name {data($result/../@name)[1]}}
{$result}
</room>
return $merge|$root//room
```

3.6(b). Counting objects in source-oriented view

```
for $a in doc("newtest.xml")//room
```

```
return count($a/item[@view="source" or not(@view)]/object)
```

3.6(c). Counting objects in data-oriented view

```
for $a in doc("newtest.xml")//room[@part="1" or not(@part)]
return count($a//object)
```

Markup approaches 1 (Figure. 3.5(a) above) and 2 (Figure. 3.5(b) above) are more difficult to query because it is necessary to retrieve data whether or not they conform to the logical model, while ensuring that the data fragments are recovered in the correct position. The query shown in Figure 3.6a constructs a new room (the kitchen) using the information contained in the <join> element (approach 1); the @target attribute values serve as pointers to the <partroom> elements assigned those ID numbers. This new room (constructed in the $merge variable) is then added to other (unfragmented) rooms in the last line of the query. Although I am sure that this query could be written more elegantly, it demonstrates the difficulties of querying complex data in XML. The queries in Figures 3.6b and 3.6c count the number of objects in each room, including a fragmented room, encoded according to approach 3 (Figure 3.5(c) above). These solutions are relatively simple, because only one pass is required through the data and no elements need to be constructed. Attributes are used to switch on one view of the data and switch off the other. A simple Boolean OR in the XPath predicate ensures that both normal data (which lack the @view attribute) and fragmented data are retrieved. The final query is particularly simple, since the detached room fragment (coded as part=”2”) can be ignored entirely.

It is often said that a system of markup reflects the purpose of the researcher. XML's flexibility means that even researchers who share the same objectives can represent data very differently. As we have seen, there are approaches to marking up even complex and irregular sources which permit easy and consistent analysis of native XML data, with the potential to permit historians to use XQuery without needing specialist programmers.

Concluding Reflections

Although XML is relatively young, markup has a much longer history. Appraisers marked up inventories, using words, symbols and layout to organise and present information about a house and the goods within it. The inventory can be seen as a representation of a house and of the spaces within it. A statistical file, relational database, and XML database may each, in turn, represent the inventory and, at one step removed, this house. Each of these representations is different. None provides a complete description of the house, any more than a transcript of an inventory, however accurate, fully captures the original hand-written document.

The debate between source-oriented and data-oriented approaches to historical data processing reflects the conflict between the complexities of historical sources and the need to simplify data for the purposes of analysis. The approaches to markup explored in this article have all addressed this tension, even when markup is used to delay the choice until the point of analysis. The appraisers of the goods of Robert Maund, and indeed all appraisers, faced similar challenges in reconciling the competing demands of listing and valuing goods. But they did not have to fit the information they recorded into a database.