32 Algorithm for Recognizing Canonical References

This chapter makes extensive use of the TEI Extended Pointer Notation, and may therefore be revised to discuss use of XPath syntax as a preferable alternative at the next release.

When a canonical reference is to be automatically processed according to this method, the following occurs:

1. The reference is analysed into a series of component targets, using the delim and length attributes on the <step> elements within the relevant <refsDecl>, as described below. The target from the first <step> element is made available for use in the subsequent pointer processing as %1, that from the second as %2, and so on. The number of targets N must be noted, as it will be used in later steps of the algorithm. (Targets with numbers greater than N may be referenced, but they will be null strings.)
2. Starting at the root of the tree (i.e. the <text> element, for a TEI-conformant document), a search is made following the specifications of the pointer in the first <step> element: its from and to attributes are processed exactly as would be done with an <xptr> element.
3. If there are more targets to be located, the search continues by following the pointer specifications in the next <step> element, using as location source the span located by the previous <step>. Except for this special location source, the processing is still identical to that done for an <xptr> element. (Note that this does not prevent an expansion of the location source at any step, either by using keywords such as previous which search outside the location source, or by an explicit return to root.)
4. When N <step> elements have been processed, the search is complete. The final result of the reference is a point or span of text, as with extended pointers.

Note that there is no backtracking or other attempt at recovery if a pointer fails. It is instead possible to design the pointers so that backtracking is not necessary: see the final example below.

When analyzing a reference into component targets, the following procedure is adopted for each <step> element that is used:

1. If only the length attribute is specified, exactly that number of characters is taken from the reference. Entity references are resolved before characters are counted. (This implies that references containing entity references may behave differently on different systems.)
2. If only the delim attribute was specified, every character up to the next occurrence of the specified delimiter is taken from the reference, and the delimiter itself is removed.
3. If both length and delim attributes are specified, a test is made for the presence of the delimiter in the reference string immediately following the specified number of characters; if this test fails, the reference fails.
4. If neither length nor delim attribute is specified, the remainder of the reference string is taken. This should happen only for the last <step> in a <refsDecl>.
5. The number of components resulting from this procedure must not exceed the number of steps in the associated declaration, but may be less than it. (For example, a long poem might be divided into cantos and lines, but a reference can point to either a line in a canto or to a whole canto; a reference to a whole canto would not require the <step> for the line number.)

Here is an example of how a reference system for an encoding of the Bible could be specified:

```
<refsDecl>
   <step refunit="book"    delim=" " from="CHILD (1 DIV N %1)" to="DITTO"/>
   <step refunit="chapter" delim=":" from="CHILD (1 DIV N %2)" to="DITTO"/>
   <step refunit="verse"   delim=""  from="CHILD (1 DIV N %3)" to="DITTO"/>
</refsDecl>
```

With this reference declaration, a canonical reference of the form “Matt 5:7” is processed by first searching for the subelement of the <text> element with an n attribute having value Matt; then searching within that element for a subelement with an n attribute having value 5; and finally for a further nested element numbered 7. This example assumes that the unnumbered

elements nevertheless follow a predictable hierarchy: the first level is always for books, the second for chapters, and the third for verses. The following reference declaration would allow intermediate elements of any sort, because it would search at each step not only for the right n attribute but also for a type attribute identifying the structural type of the division:

```
<refsDecl>
   <step refunit="book"    delim=" "
      from="DESCENDANT (1 DIV N %1 TYPE BOOK)" to="DITTO"/>
   <step refunit="chapter" delim=":"
      from="DESCENDANT (1 DIV N %2 TYPE CHAPTER)" to="DITTO"/>
   <step refunit="verse"   delim=""
      from="DESCENDANT (1 DIV N %3 TYPE VERSE)" to="DITTO"/>
</refsDecl>
```

Other reference systems depend on markers such as page and line numbers which do not correspond to structural divisions of the text. These will typically be marked in the text by milestone elements which identify single points in the text, rather than by structural elements which contain the portion of the text to be located. It is then necessary to construct extended pointers in the reference declaration that can locate both the start and the end of any segment. Here is a reference declaration for a work whose reference system consists of page and line numbers: for example, “93.3”.

```
<refsDecl>
   <step refunit="page" delim="." from="FOLLOWING (1 PB N %1)" to="FOLLOWING (1 PB)"/>
   <step refunit="line" delim="" from="FOLLOWING (1 LB N %2)" to="FOLLOWING (1 LB)"/>
</refsDecl>
```

To locate the specified page, the application must first search for the first <pb> element with n equal to 93. It must then find the end of that page, which it does by searching for the next <pb> element after that for the start of page 93; this should mark the start of page 94. A similar procedure is used within the page to find the <lb> elements that mark the start and end of the desired line.

A reference system may combine elements of the last two approaches: a reference system based on line numbers is normally used for early English plays, but because such plays often combine prose and verse the line numbers sometimes refer to structural elements (for verse passages) and sometimes to arbitrary typographical boundaries (for prose). One could simply fill the entire text with milestones, even in the verse passages which do not require them; but a reference declaration can also be constructed which requires no superfluous elements. Here is such a declaration for a collection of plays, and for canonical references of the form “Changeling 1.2.44”.

```
<refsDecl>
   <step refunit="work"  delim=" "
      from="DESCENDANT (1 TEXT N %1)" to="DITTO"/>
   <step refunit="act"   delim="."
      from="CHILD (1 DIV1 N %2)" to="DITTO"/>
   <step refunit="scene" delim="."
      from="CHILD (1 DIV2 N %3)" to="DITTO"/>
   <step refunit="line"  delim=""
      from="FOLLOWING (1 (L|LB) N %4)" to="FOLLOWING (1 (L|LB))"/>
</refsDecl>
```

Instead of using the CHILD or DESCENDANT keywords to locate the <l> elements in verse passages, we use FOLLOWING, which works to locate <lb> elements as well.

The algorithm also allows ambiguity in the reference, as a substitute for requiring backtracking in the processing. Consider the following reference declaration for a text containing several works:

```
<refsDecl>
   <step refunit="work" delim=" " from="DESCENDANT (ALL TEXT N %1)" to="DITTO"/>
   <step refunit="book" delim="." from="CHILD (1 DIV1 N %2)" to="DITTO"/>
   <step refunit="poem" delim="." from="CHILD (1 DIV2 N %3)" to="DITTO"/>
   <step refunit="line" delim="" from="CHILD (1 L N %4)" to="DITTO"/>
</refsDecl>
```

Given the canonical reference Amores I.2, the application will first search for all <text> elements whose n attribute has the value Amores. This creates a location source for the second step consisting

of one or more <text> elements, possibly discontiguous; at the second step, all the <div1> elements numbered I within those <text> elements are selected. At the third, the contained <div2> elements numbered 2 are selected, and the search ends because the reference string is exhausted: it points to a whole poem, not to a single line. This reference declaration is designed for use with a text that contains several works, some of which might have the same name, so that at the first step it is not adequate to search merely for the first <text> called Amores. It may be that several works called Amores exist but only one has a book numbered I, so that at the second step the location source narrows down to the one work desired. In other approaches to this problem, the works called Amores would be treated one at a time, and if one turned out not to be the one desired backtracking to previous steps of the search would be required; this reference-processing method uses composite location sources instead to eliminate the need for backtracking.

VII: Alphabetical Reference Lists of Classes, Entities, and Elements