

1. Introducing Pure ODD

This document is a tutorial guide to some new features of the ODD language introduced at release 3.0 of TEI P5. It assumes the reader already knows something about how ODD is designed and used, and presents only those aspects which have changed with the introduction of 'pure ODD' or discussion and background information about the motivation for these changes see *Resolving the Durand Conundrum*, published in the TEI Journal (issue 6, 2013).

Two major changes are described. Firstly, the *content model* of an element specification, the content of the `<content>` element in an `<elementSpec>`, is now expressed using some new TEI elements, rather than (as previously) expressions in the RELAXNG syntax. Secondly, TEI-defined data specifications are now expressed using a `<dataSpec>` element rather than by a macro specification (`<macroSpec>` of type="dt") and a new `<dataRef>` element is used to select one as content for the `<datatype>` element which defines the datatype of an attribute

1.1. Defining a content model

In Pure ODD, we use the `<content>` element to describe the intended content of an element.

If the element concerned is empty, that is, it has no content at all, but only attributes, then the `<content>` element itself is empty:

```
<elementSpec id="emptyElement">
  <desc>an element which may not contain anything</desc>
  <content/>
</elementSpec>
```

If the element concerned contains text, we use the special element `<textNode>`

```
<elementSpec id="textElement">
  <desc>an element which may contain only textual data</desc>
  <content>
    <textNode/>
  </content>
</elementSpec>
```

A text node may be of any length, including zero.

More usually, an element has what is known as *element content*. In this case, the `<content>` element will contain references to one or more other elements, each represented by an `<elementRef>` element. If there is only one such child element it can be given directly:

```
<elementSpec id="oneElement">
  <desc>an element which may contain only a <gi>one</gi> element</desc>
  <content>
    <elementRef key="one"/>
  </content>
</elementSpec>
```

The attributes `@minOccurs` and `@maxOccurs` are used to indicate repetition. In the following example, we define an element which may contain any number of occurrences of the `<one>` element greater than two:

```
<elementSpec id="manyOneElements">
  <desc>an element which may contain two or more <gi>one</gi> elements</desc>
  <content>
    <elementRef minOccurs="2"
      maxOccurs="unbounded" key="one"/>
  </content>
</elementSpec>
```

1.1.1. Grouping elements

An element may contain references to more than one different element. These elements may be grouped in one of three ways: as a sequence, as an alternation, or interleaved.

In a sequence, all of the child elements must follow each other in the same order as they appear within the `<content>` element:

```
<elementSpec id="OneTwo">
  <desc>an element which may contain a <gi>one</gi> element followed by a <gi>two</gi>
  element</desc>
  <content>
    <sequence>
      <elementRef key="one"/>
      <elementRef key="two"/>
    </sequence>
  </content>
</elementSpec>
```

In an alternation, any of the child elements may appear:

```
<elementSpec id="OneOrTwo">
  <desc>an element which contains either a <gi>one</gi> element or a
  <gi>two</gi> element</desc>
  <content>
    <alternate>
      <elementRef key="one"/>
      <elementRef key="two"/>
    </alternate>
  </content>
</elementSpec>
```

In an interleaved model, the child elements may appear in any order :

```
<elementSpec id="OneAndTwo">
```

```

<desc>an element which contains either a <gi>one</gi> element followed by a
<gi>two</gi> element followed by a <gi>two</gi> element followed by a <gi>one</gi>
</desc>
<content>
  <interleave>
    <elementRef key="one"/>
    <elementRef key="two"/>
  </interleave>
</content>
</elementSpec>

```

Not all target schema languages support the concept of interleaving. An ODD processor may map specifications using the `<interleave>` element to a less precise construct in the target language, or to a combination of constructs in different constraint languages.

The attributes `@minOccurs` and `@maxOccurs` are also used on these grouping elements to indicate repetition of the group. For example

```

<elementSpec ident="onesTwos">
  <desc>an element which contains up to three repetitions of pairs of elements, each
  containing a <gi>one</gi> followed by a <gi>two</gi>
  </desc>
  <content>
    <sequence minOccurs="1" maxOccurs="3">
      <elementRef key="one"/>
      <elementRef key="two"/>
    </sequence>
  </content>
</elementSpec>

```

When alternations are repeated, any one of the child elements may appear any number of times

```

<elementSpec ident="onesOrTwos">
  <desc>an element which contains one or more <gi>one</gi> or <gi>two</gi>
  elements</desc>
  <content>
    <alternate minOccurs="1"
      maxOccurs="unbounded">
      <elementRef key="one"/>
      <elementRef key="two"/>
    </alternate>
  </content>
</elementSpec>

```

Occurrence indicators may be given at both levels. For example

```

<elementSpec ident="onesTwos">
  <desc>an element which contains up to three repetitions of two or more<gi>one</gi>
  elements followed by a <gi>two</gi> element</desc>
  <content>
    <sequence minOccurs="1" maxOccurs="3">
      <elementRef minOccurs="2"
        maxOccurs="unbounded" key="one"/>
      <elementRef key="two"/>
    </sequence>
  </content>
</elementSpec>

```

Sequences, alternations, and interleaved sequences may all be nested and combined as necessary, permitting quite complex structures to be expressed.

1.1.2. Mixed content

An element may have what is known as *mixed content*, meaning that it may contain a mixture of text fragments and some specified elements. This can be represented in pure ODD using `<alternate>`

```

<elementSpec ident="mixedElement">
  <desc>an element which contains any combination of text nodes, <gi>one</gi>
  elements, and <gi>two</gi> elements</desc>
  <alternate minOccurs="0"
    maxOccurs="unlimited">
    <textNode/>
    <elementRef key="one"/>
    <elementRef key="two"/>
  </alternate>
</elementSpec>

```

It may also, more economically, be represented using `<interleave>`:

```

<elementSpec ident="mixedElement">
  <desc>an element which contains any combination of text nodes, <gi>one</gi>
  elements, and <gi>two</gi> elements</desc>
  <interleave>
    <textNode/>
    <elementRef key="one"/>
    <elementRef key="two"/>
  </interleave>
</elementSpec>

```

A text node may appear anywhere within an alternation or a sequence. For example

```

<elementSpec ident="textOne">
  <desc>an element which may contain either two or more<gi>one</gi> elements, or a
  text node</desc>
  <content>
    <alternate>
      <elementRef minOccurs="2"
        maxOccurs="unbounded" key="one"/>
      <textNode/>
    </alternate>
  </content>
</elementSpec>

```

However, not all current schema languages support such content models.

1.1.3. Class references

A `<classRef>` element can be used within a content model in the same way as an `<elementRef>`. It is a shorthand way of saying that any member, or all members, of a named *model class* of elements is permitted. For example

```
<elementSpec id="threeDigitals">
  <desc>an element which contains a sequence of up to three elements which are members
    of the <gi>model.digital</gi> class </desc>
  <content>
    <sequence minOccurs="1" maxOccurs="3">
      <classRef key="model.digital"/>
    </sequence>
  </content>
</elementSpec>
```

A model class contains a predefined set of element types which we wish to manipulate or reference together, usually because they can all appear in the same context, or share other properties. Pure ODD allows us to define such classes by means of a `<classSpec>` element. Once defined, members of that class can be referenced by means of a `<classRef>` element, as we have just seen. An element specification (not discussed here) includes a specification of the classes of which it is a member in its `<classes>` element.

1.2. Datatypes

A further part of the specification for an element is the list of attributes it may bear, provided by an `<attList>` element. The specification for an attribute (provided by an `<attDef>` element) also includes information about the kinds of value it is permitted to take, for example, whether it is a date or an integer. We call this its *datatype*.

In Pure ODD, a datatype is specified using the `<dataRef>` element. In the following example we define an attribute called `@count` on the element `<one>`, and specify that its value must be a positive integer:

```
<elementSpec id="one">
  <desc>an empty element with an attribute</desc>
  <content/>
  <attList>
    <attDef id="count">
      <desc>indicates how often the one element is used</desc>
      <datatype>
        <dataRef name="nonNegativeInteger"/>
      </datatype>
    </attDef>
  </attList>
</elementSpec>
```

The name `nonNegativeInteger` identifies one of the datatypes defined by the W3C as part of its schema language, and is not further defined by our ODD.

It is however possible to provide a more detailed specification for a datatype using the `<dataSpec>` element to document its values, intended uses, etc. The TEI defines many such datatypes in the Guidelines, and these can also be re-used directly within your ODD. The attribute `@key` is used (rather than `@name`) to indicate that a TEI-defined or locally-defined datatype is intended, as in the following example:

```
<attList>
  <attDef id="count">
    <desc>indicates how often the one element is used</desc>
    <dataRef key="teidata.count"/>
  </attDef>
</attList>
```

In this example, the name `teidata.count` identifies a datatype specification. That specification is provided by a `<dataSpec>` element with the identifier `teidata.count`, which is provided as part of the TEI system. A TEI `<dataSpec>` is similar to an `<elementSpec>`. Its `<content>` element may contain a `<dataRef>` element or a `<valList>`, or a number of such elements combined using the elements `<alternate>` or `<sequence>` in the same way as `<elementRef>`s are combined.

For example, if we wish to say that the value of the attribute `@count` can be either a non-negative integer or the string `unknown` we would first define a `<dataSpec>` with appropriate `<content>`:

```
<dataSpec id="dataCountPlus">
  <desc>permits non-negative integers or the string <q>unknown</q> or equivalent</desc>
  <content>
    <alternate>
      <dataRef name="nonNegativeInteger"/>
      <valList>
        <valItem id="unknown"/>
        <valItem id="inconnu"/>
        <valItem id="unbekannt"/>
      </valList>
    </alternate>
  </content>
</dataSpec>
```

As noted above, the `@key` attribute on `<dataRef>` can then be used to refer to this data specification:


```
<attList>
  <attDef id="count">
    <desc>may indicate how often the one element is used if we know</desc>
    <dataRef key="dataCountPlus"/>
  </attDef>
</attList>
```

A `<dataRef>` element can also be used to define the content of an element. For example, there is a TEI data specification called `teidata.xpath` which can be used to indicate that the value of an attribute must be a conformant XPath specification.

```

<attList>
  <attDef ident="path">
    <desc>indicates an XPath to the nodes required</desc>
    <dataRef key="teidata.xpath"/>
  </attDef>
</attList>

```



The same `<dataRef>` might also be used to indicate that the content of an element must be a conformant XPath specification:

```

<elementSpec ident="path">
  <desc>indicates an XPath to the nodes required</desc>
  <content>
    <dataRef key="teidata.xpath"/>
  </content>
</elementSpec>

```

Note however that not all schema languages support the ability to constrain element content in this way.