

A unified model for text markup: TEI, Docbook, and beyond

Sebastian Rahtz Norman Walsh Lou Burnard March 2004

This paper describes the practical steps taken by Docbook and TEI developers to allow for experimental interleaving of documents, and describes how the TEI literate programming model for describing schemas also provides a framework to link with other conceptual models, such as ISO 12620 and CIDOC CRM.

1 Introduction

Surprisingly few schemas or DTDs are very widely used in creating textual material. Leaving aside the proprietary schemas used by some publishers, the three open markup schemes available are HTML, Docbook and the Text Encoding Initiative Guidelines (TEI); beyond this authors seem to simply invent their own. Between them the three schemes cover the areas of web pages, technical documentation, and literary and linguistic material.

There are differences of emphasis between HTML, TEI and Docbook: TEI, for example, was originally conceived of as a means of representing previously existent documents, rather than creating them from scratch; HTML was originally intended for use solely as a means of combining and creating digital resources; Docbook was originally thought of as a publishing DTD, for converting between digital and print versions of newly authored documents. However, there is also a great deal of overlap in that they each provide metadata markup, structural markup, and paragraph-level feature markup. Usage of each has strayed far beyond its original remit, partly because of the inevitable blurring of distinctions between print and digital forms of distribution which XML has facilitated. Manuscripts for publication may be authored in HTML or TEI; textual analysis of Docbook documents is feasible. Where once we had programming language wars, are we now doomed to have document schema wars?

It has been recognised for many years that some areas of markup require specialized vocabularies: two obvious examples being SVG and MathML. These come cocooned in their own namespaces, and are (largely) designed to describe terminal nodes (ie one seldom uses other vocabularies *within* them); it is thus quite easy to embed them in other languages, using XML namespaces:

and so we can conclude that $\text{xmlns}="http://www.w3.org/1998/Math/MathML" \text{overflow}="scroll" <msub> <mi>E</mi> <mrow> <mtext>max</mtext> </mrow> </msub> </math> measures the contribution of the collisions with energy transfer close to . . .$

HTML, TEI, and, Docbook, however, are not really specialised vocabularies: each has aspirations to be a general vocabulary. Consequently, when we try to merge any one of HTML, Docbook or TEI with any of the others, it is much harder to a) know at what level markup can be incorporated, and b) at what points one can return to the outer language. Simple examples would be using TEI elements to mark up a play inside a Docbook document, or using Docbook elements to describe a GUI inside a TEI document.

The principle customization feature of XML DTDs is the parameter entity. With careful planning, parameter entities provide a mechanism for sophisticated, structured redefinition of parts of a schema. At the end of the day, however, parameter entities offer little more than the ability to perform “string substitutions” on the DTD sources. This presents serious limitations:

1. Parameter entities are just strings and once defined can never be extended, adjusted, or changed in any way. (DTDs can declare parameter entities more than once, but only the first declaration encountered has any effect.)
2. Parameter entities must be defined before they are used. All of the modules of a DTD must be arranged so that a single linear pass through the sources encounters the definition of each parameter entity before its first use.
3. Parameter entities operate on the DTD at a lexical level. Their definitions must be constructed so that their replacement text directly satisfies the lexical constraints of the construct where they occur. There is no underlying model upon which they operate.

The limitations of parameter entities are compounded by several limitations inherent in DTDs:

1. Content models must be unambiguous. From any given token in a content model, it must be possible to decide if the next token is valid or not without lookahead. One particular consequence of this rule is that a choice group cannot contain the same element more than once. This means that you can't, for example, say that `<emphasis>` is in both the `technical-inlines` and the `publishing-inlines` if there's a content model somewhere that includes `technical-inlines | publishing-inlines`.

2. Mixed content must be described by a content model that consists of a single, top-level choice of repeatable, optional elements. This means that you can't mix several choices of elements into a content model that has mixed content.
3. There's no indirection. You can't, for example, indicate that a particular element simply isn't allowed. Instead, you have to contrive to explicitly remove it from every element declaration where it occurs.

In the face of these challenges, providing a customization structure that allowed one to mix the TEI and DocBook is practically impossible. It might actually *be* impossible.

In the past couple of years, however, Relax NG expressions of both the TEI and DocBook have been constructed. Neither of these versions are as yet officially adopted standards, but both are developing in that direction.

Relax NG removes all of the limitations imposed by DTDs: it provides a tree-model in which customizations can be constructed, it has a pattern mechanism that operates in that model, rather than at the lexical level, it does not require unambiguous content models, and it imposes no additional constraints on how mixed content is expressed.

Broadly speaking, the key to Relax NG's flexibility is the use of patterns. A Relax NG validator matches the input document against a set of available patterns. If the document matches, it is valid, if it doesn't, it isn't. A pattern can match an element, or a complex structure of elements, or text, or attributes.

Let's examine a few features of Relax NG patterns that enable a design that will allow us to mix TEI and DocBook in sensible ways.

- Patterns can be extended. If you've declared that a particular pattern matches a choice of `emph`, `term`, or `gloss` patterns: `class.hqphrase = emph | tag | gloss` You can still come along later and add a new pattern, perhaps `acronym`, to that pattern: `class.hqphrase |= acronym` The validator will update the pattern and use this augmented value for validation. This means that different modules can independently extend and adapt patterns.
- There's a "null" pattern: the `notAllowed` pattern matches nothing. This means you can define extension points that are explicitly empty. Other modules can extend or adapt these patterns. In DocBook, for example, the `table` pattern is initially defined as "notAllowed". It is used throughout the schema in places where tables are logically allowed. This produces valid patterns, but ones in which the "`| table`" branch will never match anything. If the CALS Table Module were included in a schema, it might redefine the `table` pattern to include CALS tables. The HTML Table Module includes HTML tables. In this way, for example, it is easy to create a customization layer for DocBook which allows either CALS or HTML tables, or both. DocBook, by default, allows both.
- Patterns are not required to be deterministic. Relaxing this constraint allows us to add elements to patterns without concern for whether those patterns appear in choice groups elsewhere. By the same token, we don't have to worry about how mixed content is constructed.

In the remainder of this paper we explore how the TEI and Docbook projects can help make XML vocabulary merging easier.

2 Docbook and TEI development

Both the TEI Consortium (<http://www.tei-c.org>) and OASIS Docbook Technical Committee (<http://www.docbook.org/>) are planning to release major new versions in 2004, both based on Relax NG (<http://www.relaxng.org>) as the normative model for expressing constraints on the vocabularies. Both schemes make heavy use of element and attribute classes, which is being increasingly formalized.

As part of this internal reorganisation of both projects, Norman Walsh was asked to take part in the TEI working party looking at the core markup, and to establish links between the two vocabularies. This paper is one of the results of this collaboration.

In all the examples in this paper, we use the unpublished and unofficial Relax NG versions of the TEI (planned release P5) and Docbook (Docbook NG, being developed). *These versions are not stable!*. It is likely, for instance, that class names in Docbook will change. There is no guarantee that the examples given in this paper will still work at the final release, but the basic principles will not change.

The Docbook NG project consists of Relax NG schema modules directly written in the compact syntax. These make extensive use of the full power of the Relax NG language, and full backward compatibility with DTDs is not a requirement. The system of classes of elements and attributes is directly expressed using Relax NG. In the following example, we see how `<guibutton>` is in the class `gui.inlines`, which is in `domain.inlines`, which is in `inlines`.

```
inlines = text | ubiq.inlines | general.inlines | domain.inlines | extension.inlines
domain.inlines = technical.inlines | error.inlines | os.inlines | programming.inlines | markup.inlines | math.inlines |
gui.inlines | keyboard.inlines
gui.inlines = db.guiicon | db.guibutton | db.guimenuitem | db.guimenu
| db.guisubmenu | db.guilabel | db.menuchoice | db.mousebutton
keyboard.inlines = db.keycombo | db.keycap | db.keycode | db.keysym | db.shortcut | db.accel
db.guibutton = element guibutton {
  guibutton.attlist, (docbook.text | db.accel)* }
```

The TEI, on the other hand, is authored using a Literate Programming scheme, in which a specialized vocabulary (itself a module of the TEI) describes a schema, from which runnable schemas or DTDs can be generated. At the level of the element and attribute content model, it uses Relax NG directly, but the classes of elements are managed more abstractly. In the following example, `<date>` is defined as being a member of the `tei.data` and `tei.date` classes, and it can contain elements from the `macro.phraseSeq` collection. This expands into a structure comparable to the Docbook example above.

```
<tagDoc xmlns="http://www.tei-c.org/ns/1.0" id="DATE" usage="opt"> <ident>date</ident>
<equiv/> <gloss/> <classes> <memberOf key="tei.data"/> <memberOf key="tei.date"/>< /></classes>
<schemaContent> <rng xmlns="http://relaxng.org/ns/structure/1.0" name="macro.phraseSeq"/>
</schemaContent> .... </tagDoc>
```

3 Docbook/TEI Intersections

TEI and Docbook vocabularies are each in their own namespace. Using Relax NG wrapper schemas, it is easy to load patterns from both Docbook and TEI, and to judiciously redefine element classes and allow controlled interleaving of markup. Although RelaxNG patterns are generally in a global namespace, and thus open to conflict, in practice we have not encountered the issue.

If this is to work, the user needs clearly-defined interchange points. The TEI and Docbook developers can provide a list of the places where it makes semantic sense to move between languages. It is obvious that we do not want to allow, for instance, TEI paragraph-level elements to appear in the content models of Docbook inline elements. But what are ‘paragraphs’? At present it is up to the user to decide which of the public classes of the TEI (all named with the prefix `tei.`) and Docbook (not yet so formally named) can be interleaved, but we can provide some help.

Let us first consider the Docbook inline elements, which make up the great majority of the vocabulary. The following table shows the high level inline classes:

<code>inlines</code>	<code>ubiq.inlines</code> <code>general.inlines</code> <code>domain.inlines</code> <code>extension.inlines</code>
<code>general.inlines</code>	<code>publishing.inlines</code> <code>product.inlines</code> <code>bibliography.inlines</code> <code>graphic.inlines</code> <code>indexing.inlines</code> <code>link.inlines</code> <code>glossary.inlines</code>
<code>domain.inlines</code>	<code>technical.inlines</code> <code>error.inlines</code> <code>os.inlines</code> <code>programming.inlines</code> <code>markup.inlines</code> <code>math.inlines</code> <code>gui.inlines</code> <code>keyboard.inlines</code>

The next table shows the elements that these classes are composed of:

3 DOCBOOK/TEI INTERSECTIONS

technical.inlines	db.replaceable db.systemitem db.option db.optional db.nonterminal
programming.inlines	db.function db.parameter db.varname db.returnvalue db.type db.classname db.exceptionname db.interfacename db.methodname db.modifier db.initializer oo.inlines
product.inlines	db.productnumber db.productname db.database db.application db.hardware db.trademark
os.inlines	db.prompt db.envar db.filename db.command db.computeroutput db.userinput
markup.inlines	db.tag db.markup db.token db.symbol db.literal db.code db.constant db.email
bibliography.inlines	db.citation db.citerefentry db.citetitle db.citebiblioid db.author db.personname db.orgname db.editor
publishing.inlines	db.abbrev db.acronym db.emphasis db.footnote db.footnoteref db.foreignphrase db.phrase db.quote db.subscript db.superscript db.wordasword db.coref
math.inlines	db.inlineequation
graphic.inlines	db.inlinemediaobject
indexing.inlines	notAllowed db.indexterm
gui.inlines	db.guiicon db.guibutton db.guimenuitem db.guimenu db.guisubmenu db.guilabel db.menuchoice db.mousebutton
keyboard.inlines	db.keycombo db.keycap db.keycode db.keysym db.shortcut db.accel
link.inlines	db.xref db.uri db.anchor

If we now turn to the TEI, the following table lists the public TEI classes which the user should be able to override or extend (some obscure or problematic ones have been omitted) in the broad area of inline markup:

tei.Incl	groups empty elements which may appear at any point within a TEI text.
tei.addrPart	groups elements which may constitute a postal or other form of address.
tei.agent	groups elements which contain names of individuals or corporate bodies.
tei.bibl	groups elements containing a bibliographic description.
tei.biblPart	groups elements which can appear only within bibliographic citation elements.
tei.complexVal	groups elements which express complex feature values in feature structures.
tei.data	groups phrase-level elements containing names, dates, numbers, measures, and similar data.
tei.date	groups elements containing a date specifications.
tei.demographic	groups elements describing demographic characteristics of the participants in a linguistic interaction.
tei.edit	groups phrase-level elements for simple editorial correction and transcription.
tei.editIncl	groups empty elements which perform a specifically editorial function, for example by indicating the start of a span of text added, deleted, or missing in a source.
tei.fragmentary	groups elements which mark the beginning or ending of a fragmentary manuscript or other witness.
tei.hqinter	groups elements related to highlighting which can appear either within or between chunk-level elements.
tei.hqphrase	groups phrase-level elements related to highlighting.
tei.lists	groups all list-like elements.
tei.loc	groups elements used for purposes of location and reference

tei.metadata	groups empty elements which describe the status of other elements, for example by holding groups of links or of abstract interpretations, or by providing indications of certainty etc., and which may appear at any point in a document.
tei.notes	groups all note-like elements.
tei.personPart	groups those elements which form part of a personal name.
tei.phrase	groups those elements which can occur at the level of individual words or phrases.
tei.placePart	groups those elements which form part of a place name.
tei.refsys	groups milestone-style elements used to represent reference systems
tei.seg	groups elements used for arbitrary segmentation.
tei.segment	groups segmenting elements.
tei.singleVal	group elements which express single feature values in feature structures.
tei.stageDirection	groups elements containing specialized stage directions defined in the additional tag set for performance texts.
tei.temporalExpr	groups component elements of temporal expressions involving dates and time, and defines an additional set of attributes common to them.
tei.tpParts	groups those elements which can occur as direct constituents of a title page (<docTitle>, <docAuth>, <docImprint>, <epigraph>, etc.)

The second broad classification is of block-level structural elements. In Docbook this is covered by a clean set of public classes:

```

blocks
blocks.nopara

admonition.blocks
extension.blocks
formal.blocks
graphic.blocks
informal.blocks
list.blocks
para.blocks
publishing.blocks
synopsis.blocks
technical.blocks
verbatim.blocks

blocks.nopara para.blocks
list.blocks admonition.blocks formal.blocks
informal.blocks publishing.blocks
graphic.blocks technical.blocks
verbatim.blocks synopsis.blocks

```

The TEI's class system is rather different from this, in that it classifies elements as much by their structural properties as by their semantics. In general, however, the TEI classes described earlier in the inline section are more flexible than the comparable Docbook ones, as they can often occur in both inline and block-level contexts.

tei.common	This class defines the set of chunk- and inter-level elements available in all bases.
tei.divbot	groups elements which can occur at the end of a text division; for example, trailer, byline, etc.
tei.divtop	groups elements which can occur at the start of any division class element.

4 PRACTICAL EXAMPLES

tei.dramafront	groups elements which appear at the level of divisions within front or back matter of performance texts only.
tei.fmchunk	groups elements which can occur as direct constituents of front matter, when a full title page is not given.
tei.front	groups elements which appear at the level of divisions within front or back matter.
tei.paragraph	The paragraph element, made into a class for the purpose of interchange.
tei.teiText	Main element covering the body of a TEI document

The last major set of classes is that which describes metadata. In Docbook, this is managed by classes which describe the metadata attached to a large number of elements:

docbook.info, docbook.info.titlereq, docbook.info.titleonly, docbook.info.titleonlyreq, docbook.info.titleforbidden

In the TEI, all metadata is (usually) grouped together in a single `<teiHeader>` element, and referenced by text elements. Some of the TEI classes which are used in the header are listed next.

tei.categorize	groups elements which may be used inside <code><catDesc></code> and appear multiple times
tei.encoding	groups elements which may be used inside <code>encodingDesc</code> and appear multiple times
tei.header	groups elements which may be used inside <code>teiHeader</code> and appear multiple times
tei.teiHeader	Metadata elements at the top of a TEI document
tei.profile	groups elements which may be used inside <code><profileDesc></code> and appear multiple times

It will be evident from the foregoing discussion that adjoining TEI and Docbook vocabularies is not entirely straightforward, as there is not a great deal of semantic overlap. The picture is also confused by the fact that the TEI operates as a series of modules which overload a small number of classes with extra elements, whereas Docbook tends more to defining new classes in its modular sections.

In our work, we have identified three ways of proceeding. First, the Docbook inline elements can be used in any of the TEI ‘phrase-like’ situations, and the TEI technical classes can be used alongside Docbook inlines. Second, the main block-level classes are comparable (eg TEI `tei.lists` and `list.blocks`). Lastly, the highest level Docbook info classes should be interchangeable with TEI’s `<teiHeader>`. In the following section we give examples of each of these strategies.

4 Practical examples

We choose five scenarios to model using a mixture of Docbook and TEI. The aim was not to demonstrate real-life situations, but to cover four basic operations:

1. adding an individual element from one schema to a class in another scheme
2. replacing a class content model from one schema with that from another
3. extending a class content model with alternatives from another scheme
4. interleaving classes of elements from both schemes

In these examples, we show the XML document we want to validate, and the schema in Relax NG compact form. It is assumed that the reader is moderately familiar with Docbook, TEI and Relax NG vocabulary. Three basic technique used is to extend or replace an existing class; in RelaxNG, a class is

implemented as a pattern containing a choice. Thus we can group `<para>`, `<quote>` and `<list>` by defining a pattern `blocks` as follows:

```
blocks = para | quote | lists
```

We can then add another choice to the class by writing

```
blocks |= speech
```

So any reference to `blocks` now means `para | quote | lists | speech`.

4.1 Docbook adding to TEI

A TEI document uses the Docbook `<qandaset>` element inside sections to ask a basic question:

```
<div> <head>In normal TEI mode </head> <p>Marley was dead: to begin with. There is no
doubt whatever about that. The register of his burial was signed by the clergyman, the clerk,
the undertaker, and the chief mourner. Scrooge signed it. And Scrooge's name was good upon
'Change, for anything he chose to put his hand to. Old Marley was as dead as a door-nail.</p> </div>
<div> <head>Docbook QandA example</head> <qandaset xmlns="http://docbook.org/docbook-ng" >
<qandaentry> <question> <para> To be, or not to be? </para> </question> <answer> <para> That is the
question. </para> </answer> </qandaentry> </qandaset>
```

The need here is to explicitly add the Docbook `<qandaset>` element to the TEI `lists` class. We load the default TEI core tagsets, and all of Docbook. It is also necessary to override the Docbook default starting elements, as the master scheme here is TEI.

```
default namespace = "http://www.tei-c.org/ns/1.0" include "teilib.rnc" { tei.lists |= db.qandaset }
include "docbook.rnc" { start |= TEI }
```

4.2 TEI adding to Docbook

A Docbook document uses the TEI play markup to embed portions of Hamlet:

```
<article xmlns:tei="http://www.tei-c.org/ns/1.0" xmlns="http://docbook.org/docbook-ng"
version="bourbon"> ... <section><info><title>And now for something completely different</title></info>
<tei:stage type="setting"> Elsinore. A platform before the castle. FRANCISCO at his
post.</tei:stage> <tei:stage type="entrance"> Enter to him BERNARDO. </tei:stage> <tei:sp
who="Barnardo"><tei:speaker>Bernardo</tei:speaker> <tei:l part="Y">Who's there? </tei:l> </tei:sp>
<tei:sp who="Francisco"><tei:speaker>Francisco</tei:speaker> <tei:l >Nay, answer me: stand, and
unfold yourself. </tei:l> </tei:sp> <tei:sp who="Barnardo"><tei:speaker>Bernardo</tei:speaker> <tei:l
part="Y">Long live the king! </tei:l> </tei:sp> </section> </article>
```

Here the master scheme is Docbook; there is no need to override the default start pattern for TEI, as we load a view of the TEI (`teilib.rnc`) which is intended for interchange, and has no `<start>` pattern at all. The TEI class `common` is extended to allow the Docbook `technical.blocks` class.

```
include "docbook.rnc" include "teilib.rnc" technical.blocks |= tei.common
```

4.3 TEI class replacing Docbook class

A Docbook document in which the normal metadata header is replaced by a TEI `<teiHeader>`:

```
<article xmlns="http://docbook.org/docbook-ng" version="bourbon"> <teiHeader
xmlns="http://www.tei-c.org/ns/1.0"> <fileDesc> <titleStmt> <title>Testing TEI headers inside
Docbook</title> <author>Sebastian Rahtz and Norman Walsh</author> </titleStmt> <publicationStmt>
<p>published by OUCS</p> </publicationStmt> <sourceDesc> <p>written from scratch</p>
</sourceDesc> </fileDesc> </teiHeader> <section> <info><title>Introduction</title></info> ...
</article>
```

Again, we simply load Docbook and TEI interchange core; but this time we do not extend an existing class, but replace it.

```
include "docbook.rnc" { article.info = tei.teiHeader } include "teilib.rnc"
```

4.4 Docbook class replacing TEI class

A TEI document in which the normal metadata header is replaced by a Docbook `<info>`, and MathML is used in the body of the TEI document:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0"> <info xmlns="http://docbook.org/docbook-ng"> <title>Simulation of Energy Loss Stragglng</title> <author><personname> <surname>Physicist</surname> <firstname>Maria</firstname></personname></author> <pubdate>2004-03-11</pubdate> <copyright> <year>1999</year> <holder>CERN</holder> </copyright> </info> <text> <body> <p> ...can be significantly affected by such fluctuations in their active layers. The description of ionisation fluctuations is characterised by the significance parameter <formula notation="MathML"> <math xmlns="http://www.w3.org/1998/Math/MathML" overflow="scroll"> <mi></mi> </math> </formula>, .... </body> </text> </TEI>
```

This is a much more complex example. First, we declare some namespaces, and load MathML 2 and Xlink modules (the latter is used internally by MathML)

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0" namespace xlink = "http://www.w3.org/1999/xlink" namespace s = "http://www.ascc.net/xml/schematron" default namespace = "http://www.tei-c.org/ns/1.0" include "Schema/xlink.rnc" {} include "Schema/mathml2-main.rnc"
```

Next we load Docbook, and override its start pattern:

```
include "docbook.rnc" { start |= TEI }
```

Loading the TEI is more complex, as it has several optional modules which we need for this example. In the core, we overload the existing class for headers with that from Docbook, and in the module which covers figures and formula, we overload the datatype pattern for formula content with the root MathML pattern.

```
include "teilib.rnc" { tei.teiHeader = article.info } include "Schema/linking.rnc" {} include "Schema/figures.rnc" { datatype.Formula = mathml.math }
```

Finally, we create a new element `<code>` and add it to an existing TEI class.

```
code = element code { code.content } code.content = code.attributes, text code.attributes = tei.global.attributes, attribute type { datatype.Text }?, [ a:defaultValue = "code" ] attribute TEIform { text }? tei.oddPhr |= code
```

4.5 TEI and Docbook interleaved

A TEI document which allows for Docbook elements for describing GUIs in inline contexts, and for TEI structured inline elements to then appear inside the Docbook GUI elements

```
<TEI xmlns="http://www.tei-c.org/ns/1.0" xmlns:dbk="http://docbook.org/docbook-ng"> .... <text> <body> <p>The button on our web page has the current date on it: <dbk:guibutton> <date calendar="Julian" value="1732-02-22">Feb. 11, 1731/32, O.S.</date> </dbk:guibutton> or at least the date on which we last updated it.</p> </body> </text> </TEI>
```

This final example is deceptively simple, but very powerful. We tell the TEI that it can use the Docbook `gui.inlines` class in with its normal phrase-level elements, and tell Docbook that it can use the TEI class `data` in *its* phrase-level class.

```
include "teilib.rnc" { tei.hqphrase |= gui.inlines } include "docbook.rnc" { docbook.text |= tei.data start = TEI }
```

5 Validation of testing

The Relax NG world is relatively rich in good-quality validators and schema-aware editing tools (<http://www.relaxng.org/#validators>), and we have tested our five examples with a variety of the available software, as follows.

MSV <http://www.sun.com/software/xml/developers/multischema/> - Sun's multi-schema validator

Jing <http://www.thaiopensource.com/relaxng/jing.html> - A Relax NG Validator in Java.

RNV <http://davidashen.net/rnv.html> - a Relax NG validator in C (Compact Syntax only)

nxml mode <http://www.thaiopensource.com/download/> - an Emacs mode for editing XML files using Relax NG

xmllint <http://xmlsoft.org/> - part of the Libxml2 XML C parser and toolkit developed for the Gnome project

<oxygen/> <http://www.oxygenxml.com> - commercial XML editing tool

No unexplained errors were encountered during this informal testing. We may reasonably conclude that the methodology is sound, and that it can serve as a basis for more sophisticated testing in the future.

On a usability level, it may be helpful to look at two screen shots for test 5, in Emacs () and <oxygen/> (). These are traditional XML editors, offering tag prompting and completion, and may be regarded as typical of their kind.

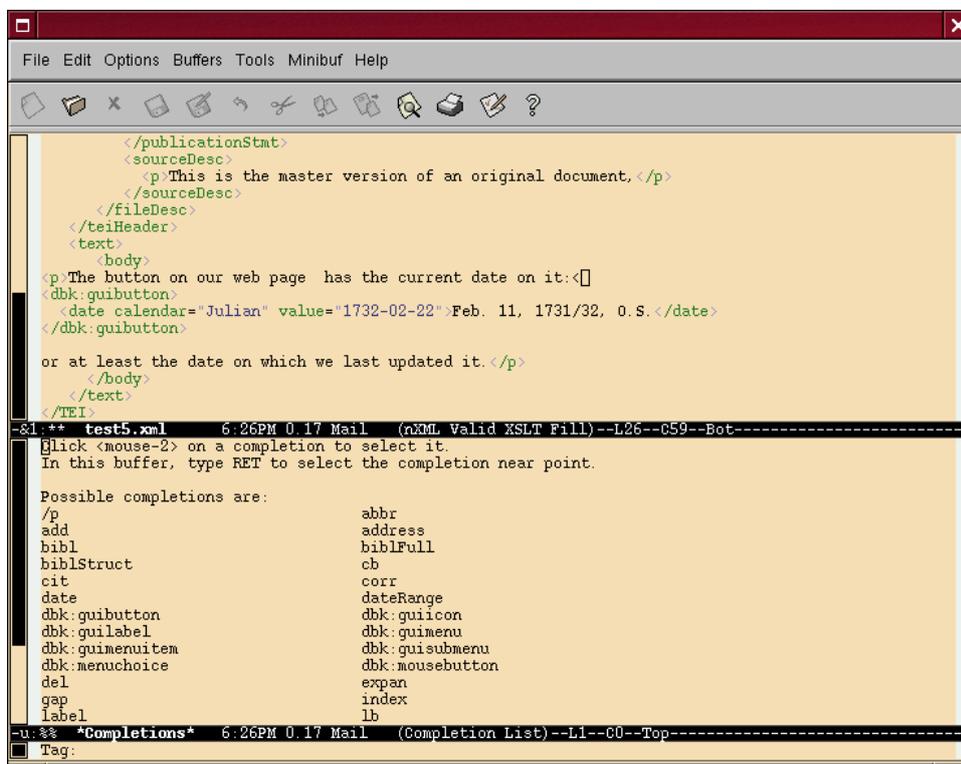


Figure 1: Mixed Docbook/TEI editing in Emacs

We have not yet attempted to use W3C Schema validators on automatically-converted schemas.

6 Linking to the wide world

Technically, it is not hard to combine vocabularies in different namespaces. Our demonstrations thus far have been satisfactory, but still remain at the level of informal agreement about semantics of classes. To go beyond this stage requires us to grasp and resolve problems which are fundamentally linguistic in nature: how are the meanings of technical terms in different languages related? how do new meanings emerge? how are meanings transferred between different languages? These are not new problems in the language research community, and it is reasonable therefore to look to that community for insight into their solution. There is an interesting synergy between the current fascination within the data processing

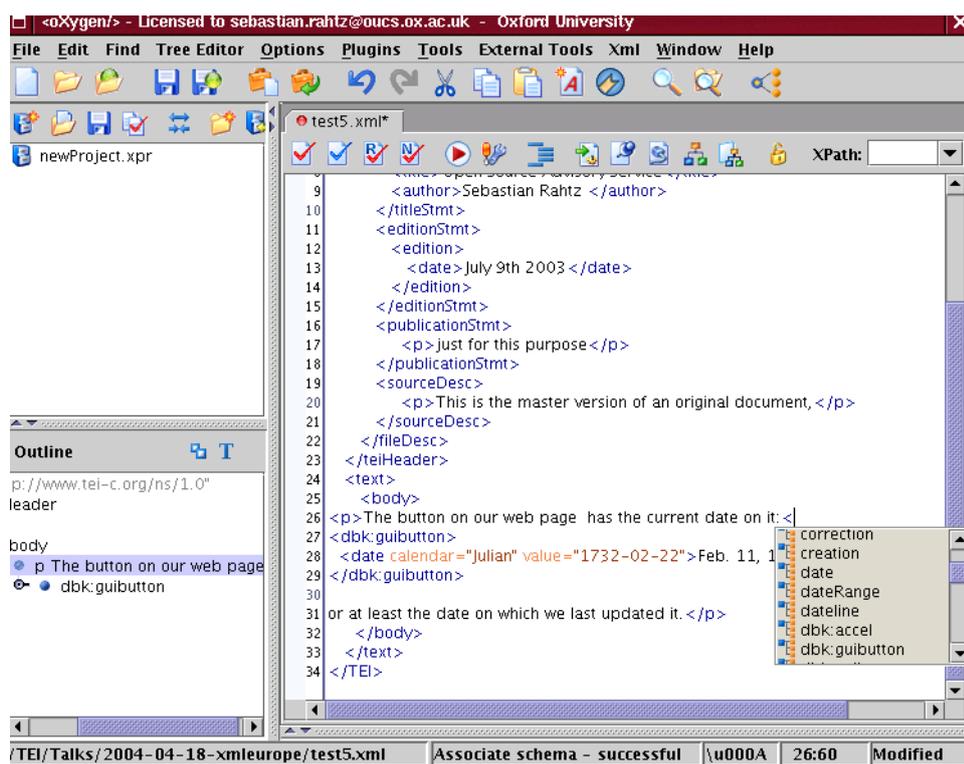


Figure 2: Mixed Docbook/TEI editing in oXygen

community for conceptual modelling tools and ontologies on the one hand, and the growing awareness within the natural language processing community of the potential of tools such as RDF and XML for modelling fundamental linguistic structures on the other. The TEI, which has a long history of enabling linguistic research, is well poised to leverage that synergy.

For example, if both the TEI and Docbook terminologies were mapped to some standard conceptual model, ‘round-tripping’ between documents in either scheme would be feasible (or at least, its feasibility would be demonstrable). No such universal conceptual model exists, and previous attempts to build universal vocabularies have been notoriously unsuccessful. Where we see real scope for progress however is in the specific area of technical terminology. Current work in ISO/TC 37/SC4 is working towards the definition of a Linguistic Annotation Framework comprising a number of related standards, one of which, ISO DIS 12620/1, relates to the definition of Data Category Registries for language resources. The idea behind this standard is to collect existing linguistic terminology from as many different fields as possible (term banks, online dictionaries, existing NLP systems etc) and construct a common concept base, which terminological systems can use to describe the data they hold. By contrast with other ontologies, such as Wordnet, which specifically exclude technical and specialized vocabulary since their goal is to model ‘common sensed’ knowledge of the world, the Data Category registry focuses on the concepts underlying linguistic description itself. As such, it provides the natural place to look for an interlingua on to which other specialized descriptive vocabularies such as TEI or Docbook can be mapped.

Other possibilities are offered by standards like the CIDOC Conceptual Reference Model (CRM) (<http://cidoc.ics.forth.gr/>), which has been developed as a way of defining a common and extensible semantic framework within which complex cultural heritage information can be expressed and interchanged. Systems which have been developed in the context of the CRM can interchange data by relating the categories used to express it to the CRM’s underlying model, independently of the terminology used. Again, there is an area of overlap with the concerns of the TEI, since many specialized TEI elements relate to concepts which are central to the CIDOC CRM.

To take advantage of these, and other possibilities, the TEI has built into its own model the concept

of *equivalence* at every level. For any element, attribute, value, and class defined in the TEI scheme, an equivalent concept in some other scheme may be referenced. We plan to continue the work reported here by identifying and defining such links between a substantial proportion of the concepts defined in the Docbook, TEI, CIDOC CRM and ISO 12620-1 standards. Amongst the challenges to be addressed will be finding ways of modelling partial equivalence and subset relations, and also ways of using the equivalences identified in document-understanding systems.

A Notes and Acknowledgements

This work was carried out as part of the technical work programme of the Metalanguage Taskforce (<http://www.tei-c.org/Activities/META/>) of the TEI Council in 2003/2004. The authors are both members of this Taskforce. We are greatly indebted to Lou Burnard and Laurent Romary, also on this taskforce, for important insights into wider linking.

Many thanks are due to David Tolpin, who made fixes to his excellent RNV Relax NG validator at very short notice, and provided vital support for some of the more complicated parts of the TEI literate programming process.

B Bibliography

- [1] Sebastian Raetz, 'Converting to schema: the TEI and RelaxNG', paper presented at XML Europe 2002, Barcelona, May 2002.
- [2] Sebastian Raetz, 'Building TEI DTDs and Schemas on demand', paper presented at XML Europe 2003, London, May 2003.
- [3] Association for Computers and the Humanities, Association for Computational Linguistics, and Association for Literary and Linguistic Computing, *Guidelines for Electronic Text Encoding and Interchange (TEI P4)*. Ed. C. M. Sperberg-McQueen and Lou Burnard. University of Virginia Press, 2001.
- [4] N. Walsh and L. Muellner, *DocBook The Definitive Guide*, O'Reilly, Sebastopol, CA, USA, 1999.
- [5] Donald E. Knuth, *Literate Programming*, Stanford University Center for the Study of Language and Information (CSLI Lecture Notes Number 27), Stanford, CA, USA, 1992.