Converting to schema: the TEI and RelaxNG

Sebastian Rahtz March 2002

The Text Encoding Initiative *Guidelines* are the product of an ambitious international research project dating from the early 1990s, the goal of which was to provide generic but detailed recommendations for the mark-up of electronic documents, in particular texts from the literary and linguistic domains. The project was one of the first large-scale attempts to apply then-emerging markup technologies to traditional scholarly and research concerns, and has had considerable impact, both within academia and beyond. The original TEI project concluded with a detailed publication, presented at SGML Europe in 1994, and subsequently revised in 1996. In 2001, the TEI was reorganized as a membership Consortium, and began work on a new XML-based release of its work, due for publication in April 2002. This release (P4) will be the last to retain compatibility with the original SGML version of the Guidelines. Future releases of the Guidelines, it seems probable, will be expressed using some form of schema-based markup rather than XML DTDs. In this paper we report on preliminary work undertaken into the feasibility of using the existing abstractions underlying the Guidelines to generate XML schemas, conventional DTDs, or the Relax NG schemas, according to need.

We describe in some detail the literate programming system used by the TEI, in which a single XML file contains both the text of Guidelines, and structured information from which DTDs (or other formalizations) are created. We also outline how the modularity of the TEI scheme is implemented, and how users follow the 'Chicago pizza model' to generate an instance DTD. Finally, we describe how the current TEI Guidelines document, which was converted from SGML to XML in 2001, can now be transformed using Relax NG, and how DTDs and schemas can be derived from it. We provide examples of how the resulting system can be used, and discuss how schema features such as namespaces and datatyping can be used to make the TEI both more modular and more rigorous.

The TEI is probably the most detailed and best documented public markup system yet devised. This paper reports on a further stage in its evolution.

1 The Text Encoding Initiative (TEI)

The Text Encoding Initiative's *Guidelines for electronic text encoding and interchange* ([1]) were the result of an extensive international research project, which aimed to provide exhaustive recommendations for the encoding of key features in literary and linguistic textual materials. These recommendations, taking the form of a modular SGML-based architecture in which DTD fragments and documentation are combined according to user-specified requirements, proved very effective and have been widely adopted in digital library, language engineering, and many other projects (the TEI now maintains a website of current TEI applications which gives some indication of their range: see `http://www.tei-c.org/Applications/`). A new membership Consortium (`http://www.tei-c.org/`) was formed in 2001, based at four universities worldwide, with the goal of maintaining and revising the Guidelines. Its first deliverable was to re-express the whole of the Guidelines in XML, and to update their recommendations in light of changes in the technological infrastructure. The TEI's technical programme focuses on further work on harmonization with new related standards and a new schema-based version of the underlying DTDs.

In the next two sections we will consider the features of the current TEI system which make it possible to consider switching to schema-based constraints.

2 TEI Literate Programming

The abbreviation DTD has two distinct expansions in the SGML standard: it may be regarded as short for document type *declaration*, or for document type *definition*. To quote the standard itself: '*(document) type declaration:* A markup declaration that formally specifies a portion of a document type definition... A document type declaration does not specify all of a document type definition because part of the definition, such as the semantics of elements and attributes cannot be expressed in SGML.'[1]. Corresponding to this distinction, most popular markup languages, such as the popular DocBook ([2]), require distinct sets of documentation: firstly, a set of SGML (or XML) declarations, catalogs etc, suitable for immediate use with document instances; and secondly, associated descriptions which say what the elements are intended to represent, and gives examples of their use. Of course, the DTD files may contain comments, but in general the model is that of traditional computer programming, which regards documentation as a separate (and often later) stage from writing code.

The TEI developed differently, partly because it did not start specifically as a project to write SGML DTDs; the initial effort concentrated on abstract models of markup, which the editors used SGML to instantiate, that being the best method available to them at the time. However, they did their best to insulate themselves from future changes by formalizing the markup constraints in a small SGML-based markup language, rather than directly writing DTD code. In this respect, they followed the 'literate programming' WEB model developed and popularized by Donald Knuth ([3], one of many references) in which a single document is produced which contains both formal code and its documentation. This single source can then be processed to produce several outputs:

1. SGML or XML documentation for the SGML DTD modules, containing embedded DTD fragments (in the same way that Pascal code is intermingled with prose in the Web system); this corresponds to the output from Knuth's *Weave* processor.

2. SGML or XML documentation for the elements, attributes, and other items in the markup language constituting an alphabetic reference manual.

3. the actual DTD modules defining the markup language; this corresponds to Knuth's *Tangle* processor.

The TEI literate programming system (jocularly named ODD, for 'One Document Does it all') as originally specified is documented in an internal TEI working paper (`http://www.tei-c.org/Vault/ED/edw29.sgm`). This design underwent several modifications during the process

[1] ISO/IEC 8879-4, clause 4.103, cited from Goldfarb, Charles F. *The SGML Handbook* (OUP, 1990), p. 264

of implementing and using it for production of the TEI Guidelines; a paper presented at the ALLC-ACH Conference in Paris in 1994 described the production system more fully cmsm-lb:1994 and it was substantially generalized by C.M. Sperberg-McQueen. Other XML/SGML-based literate programming schemes are neatly summarized at `http://xml.coverpages.org/xmlLitProg.ht`.

As noted above, a key feature of the TEI scheme is its modular design. The ODD system was developed also in part as a testbed for the effectiveness of that design: it consists of a small suite of additional declarations for special purpose elements which are embedded within a special subset of the same basic TEI DTD designed for authoring of new documents. [2] The embedding procedure used is discussed further below.

Amongst the basic building blocks of the ODD extension to TEI markup are the `<tagDoc>`, a special-purpose element for documentation of a single TEI element, and the `<classDoc>`, an element which documents a single TEI class. Elements in the TEI scheme are assigned to one or more classes of three distinct kinds: *model classes*, which group elements that have similar structural properties, i.e. they can appear at the same point in the document hierarchy; *attribute classes*, i.e. they share common attributes; and *semantic classes*, i.e. they have similar semantic properties. As a simple example, we present the tagdoc for the `<bibl>` element:

<tagDoc id="BIBL" usage="opt"> <gi>bibl</gi> <name>bibliographic citation</name> <desc>contains a loosely-structured bibliographic citation of which the sub-components may or may not be explicitly tagged. </desc> <attList/> <exemplum> <bibl>Blain, Clements and Grundy: Feminist Companion to Literature in English (Yale, 1990)</bibl> </exemplum> <exemplum> <bibl> <title level="a">The Interesting story of the Children in the Wood</title>. In <author>Victor E Neuberg</author>, <title>The Penny Histories</title>. <publisher>OUP</publisher> <date value="1968">1968</date>. </bibl> </exemplum> <remarks/> <part type="base" name="core"/> <classes names="CLBIBL DECLABL"/> <dataDesc>Contains phrase-level elements, together with any combination of elements from the <term>biblPart</term> class</dataDesc> <elemDecl> %om.RO; (#PCDATA | %m.phrase; | %m.biblPart; | %m.Incl;)*</elemDecl> <ptr target="COBITY"/><ptr target="HD3"/><ptr target="TETA"/> <ptr target="CCAS2"/> </tagDoc>

The start of this is straightforward; a description of the element, a list of attributes (here empty), and some examples. The next part is more interesting, as it places `<bibl>` into the overall scheme of the TEI. Firstly, the `<part>` element says to which module of the TEI `<bibl>` belongs (it forms part of the core module), and secondly the `<classes>` element indicates of which *classes* `<bibl>` is a member. In this case, bibl is a member of the semantic class bibl and the attribute class declarable. The last part of the `<tagDoc>` supplies the intended content model in a heavily parameterized form, and some pointers to further relevant parts of the Guidelines.

When run through the DTD generator, the following declaration is produced from this tagdoc element:

<!ELEMENT %n.bibl; %om.RO; (#PCDATA | %m.phrase; | %m.biblPart; | %m.Incl;)*> <!ATTLIST %n.bibl; %a.global; %a.declarable; TEIform CDATA 'bibl' >

Note here how the name of the element itself has been parameterized (%n.bibl;), thus permitting its renaming to another language. An attribute (TEIform) preserves the canonical name. The attribute list has been generated, using information from the `<classes>` element of the `<tagDoc>`. The values CLBIBL and DECLABL serve as pointers to the appropriate `<classDoc>` elements, on which these same values are used as identifiers: CLBIBL, for example, links to the following `<classDoc>` element:

<classDoc id="CLBIBL" type="model"> <class>bibl</class> <desc>bibliographic elements. </desc> <attList/> <remarks/> <part type="base" name="core"/> <classes names="INTER COMMON"/> <ptr target="COBI"/> </classDoc>

Although this defines no attributes itself, it does declare itself to be a member of the class COMMON, which is in turn a member of the class GLOBAL, within which the global attributes are actually declared; similarly, the class DECLABL supplies another group of attributes. The same class mechanism also

[2] This 'tiny dtd' eventually became the very widely used subset of TEI known as TEI Lite: see `http://www.tei-c.org/Lite/`

determines which DTD parameter entities the current element will be in: the <classDoc> for CLBIBL creates an entity %m.bibl; which includes bibl (or, more precisely, %n.bibl;).

An example of an element which has 'local' attributes is <formula>:

<tagDoc id="FORMULA" usage="rwa"> <gi>formula</gi> <desc>contains a mathematical or other formula.</desc> <attList> <attDef usage="opt"><attName>notation</attName> <desc>supplies the name of a previously defined notation used for the content of the element.</desc> <datatype>%formulaNotations;</datatype> <valDesc>The name of a formal notation previously declared in the document type declaration.</valDesc> <default>#REQUIRED</default> <remarks/> </attDef> </attList> <part type="top" name="ft"/> <classes names="PHRASE"/> <dataDesc>The content model for this element is specified by the parameter entity <term>formulaContent</term>, the default value of which is <mentioned>CDATA</mentioned>.</dataDesc> <elemDecl> %om.RR; %formulaContent;</elemDecl> <ptr target="FTFOR" type="div1"/> </tagDoc>

The result of this in the DTD is:

<!ELEMENT %n.formula; %om.RR; %formulaContent;> <!ATTLIST %n.formula; %a.global; notation %formulaNotations; #REQUIRED TEIform CDATA 'formula' >

in which the <attDef> element is translated into DTD notation.

The main body of the Guidelines is a conventional TEI document with chapters and sections which describe different parts of the TEI in great detail. Mixed in with informal prose are references to the <tagDoc> and <classDoc> elements, by means of <dtdFrag>, <tagDecl>, and <claDecl> elements; so the chapter describing formulae ends with:

The formula element itself is defined as follows: <dtdFrag id="DFTFOR" n="Formulae"> <tagDecl tagDoc="FORMULA"/> </dtdFrag>

This inserts a summary description of the element at this point in the text; the DTD files themselves are created by a variant on the above, e.g.

<dtdFrag id="DVE" n="Base Tag Set for Verse" url="teivers2.dtd"> <commDecl>First, declare the default text structure, which is the same for verse as it is for other kinds of text. (But declare it only if verse is the only base.)</commDecl> <msection keywords="%TEI.singleBase;"> <entDecl entDoc="TEISTR"/> <peRef n="TEI.structure.dtd"/> </msection> <commDecl>Finally, declare the elements specific to this base tag set</commDecl> <dtdRef dtdFrag="DVEST"/> <dtdRef dtdFrag="DVESE"/> </dtdFrag>

This determines in what order elements are placed in separate files of the DTD suite, and in what marked sections (using <msection>) etc. There are many small extra features of the system, but this covers the essential operations.

It should be clear from this brief overview of TEI literate programming that the current model has three main characteristics:

1. A class system for classifying elements and attributes which is essentially independent of SGML or XML;
2. Markup for describing attributes and DTD files which is expressed in XML, but is quite closely tied to DTD notation (e.g. the datatype of %formulaNotations; in <formula>)
3. A bottom-level element, <elemDecl>, which contains literal DTD code

The original aim of the TEI editors in trying to make the scheme independent of the current SGML/XML instantiation is not, in our view, completely successful; nevertheless, it provides a plausible framework for the generation of versions of the TEI Guidelines expressed using other metalanguages. We will return to this later in the paper.

3 The TEI Pizza Chef

It is essential to understand that the TEI does not provide a single DTD which covers all documents within its remit. Instead, it provides about 450 well-defined elements which can be combined together to produce a custom DTD for a particular project. At the simplest level, this customization consists of

choosing a base module[3] together with zero or more optional additional modules. The choice of base modules is:

prose Mainly prose texts

verse Texts containing verse

drama Texts containing drama

spoken Transcriptions of spoken texts.

dictionaries Print dictionaries.

terminology Terminological data files.

general and mixed Texts which combine bases.

while the additional modules are

linking Elements for linking, segmentation, and alignment (`http://www.tei-c.org/Guidelines/SA.html`)

analysis Elements for simple analytic mechanisms (`http://www.tei-c.org/Guidelines/AI.html`)

fs Elements for feature structure analysis (`http://www.tei-c.org/Guidelines/FS.html`)

certainty Elements for indicating uncertainty and probability in the markup (`http://www.tei-c.org/Guidelines/CE.html`)

transcr Elements for encoding the physical appearance of manuscripts or other primary sources (`http://www.tei-c.org/Guidelines/PH.html`)

textcrit Elements for critical editions (`http://www.tei-c.org/Guidelines/TC.html`)

names.dates Elements for names and dates (`http://www.tei-c.org/Guidelines/ND.html`)

nets Elements for graphs, digraphs, trees, and other networks (`http://www.tei-c.org/Guidelines/GD.html`)

figures Elements for graphics, figures, illustrations, tables, and formulae (`http://www.tei-c.org/Guidelines/FT.html`)

corpus Elements for language corpora (`http://www.tei-c.org/Guidelines/CC.html`)

As well as mixing and matching modules from this list, the TEI user can supply their own modifications or extensions.

The result is a fairly complex scheme, in which a document which wanted to use more or less all of the TEI elements[4] could start like this:

<!DOCTYPE TEI.2 SYSTEM "dtd/tei2.dtd" [<!ENTITY % TEI.prose "INCLUDE"> <!ENTITY % TEI.verse "INCLUDE"> <!ENTITY % TEI.drama "INCLUDE"> <!ENTITY % TEI.spoken "INCLUDE"> <!ENTITY % TEI.dictionaries "INCLUDE"> <!ENTITY % TEI.certainty "INCLUDE">

[3]Earlier discussion of the TEI uses the term *tagset* for each of the various groups of element and attribute list declarations making up the whole scheme. In this paper we have adopted the term *module* for this purpose, as being less misleading, despite its comparative unfamiliarity

[4]It is not actually possible to use *all* the TEI at the same time; some of the extra modules can only be used in isolation, and there are a few conflicts between some of the optional tag sets.

<!ENTITY % TEI.corpus "INCLUDE"> <!ENTITY % TEI.mixed "INCLUDE"> <!ENTITY % TEI.figures "INCLUDE"> <!ENTITY % TEI.analysis "INCLUDE"> <!ENTITY % TEI.fs "INCLUDE"> <!ENTITY % TEI.certainty "INCLUDE"> <!ENTITY % TEI.transcr "INCLUDE"> <!ENTITY % TEI.textcrit "INCLUDE"> <!ENTITY % TEI.names.dates "INCLUDE"> <!ENTITY % TEI.nets "INCLUDE"> <!ENTITY % TEI.linking "INCLUDE"> <!ENTITY % TEI.XML "INCLUDE">]> <TEI.2>

While perfectly acceptable in theory, this imposes a certain burden on the author and on the processing software (the DTD files make very heavy use of parameter entities and conditional sections, in ways which not all supposedly compliant processors support). There are also circumstances in which a group of users might wish to remove the parameterization, to *stop* document instances overriding things. To deal with with this, the TEI produced two related pieces of software:

Carthage The brain-child of Michael Sperberg-McQueen, this is a DTD pre-processor, which takes a parameterized DTD and 'compiles' it. In the resulting DTD, all parameter entities are expanded in place, and all content models are tidied so that they contain only elements declared in the DTD.

The Pizza Chef This is a web form which asks the user questions about which features from the TEI they want to include, lets them supply some user modification files if desired, and then calls the carthage program to make a customized DTD.

The Pizza Chef (so-called because of the supposed similarity to requesting a variety of toppings on a pizza) interface can be seen in figures , and . Note especially the last figure, which shows how individual elements within a module can be turned on or off. Elements turned off at this stage are entirely removed from the the final DTD by the carthage program.

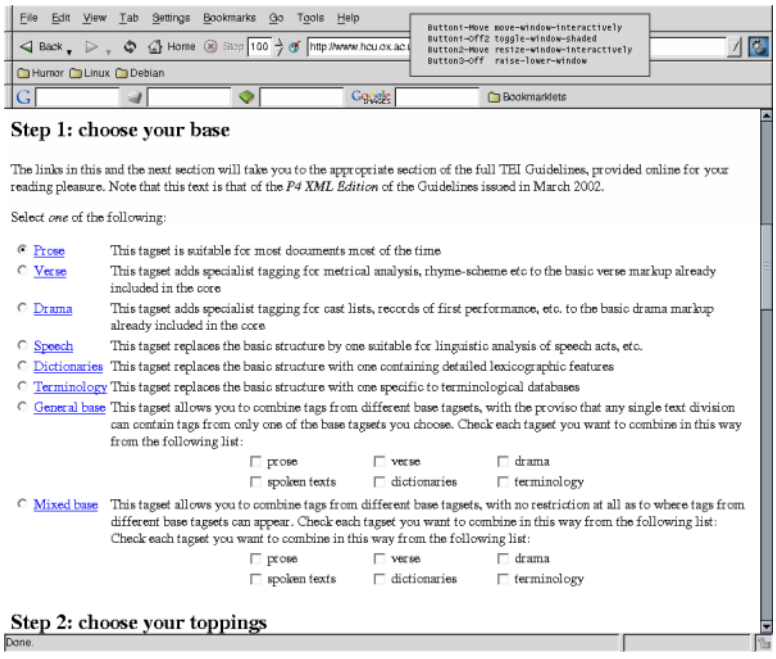

Figure 1: Choosing your base tag set in the TEI Pizza Chef

The importance of the TEI Pizza Chef is that it provides another level of abstaction on top of the world of the DTD. While, of course, detailed knowledge of the DTD syntax is needed to write modification files, the casual user can create a view of the DTD by filling in a simple form.

4 Converting the TEI to Relax NG

We have seen how the TEI tries, in two ways, to insulate the Guidelines from their current instantiation as a set of XML/SGML DTD fragments. If this insulation is effective, it should be possible to use the

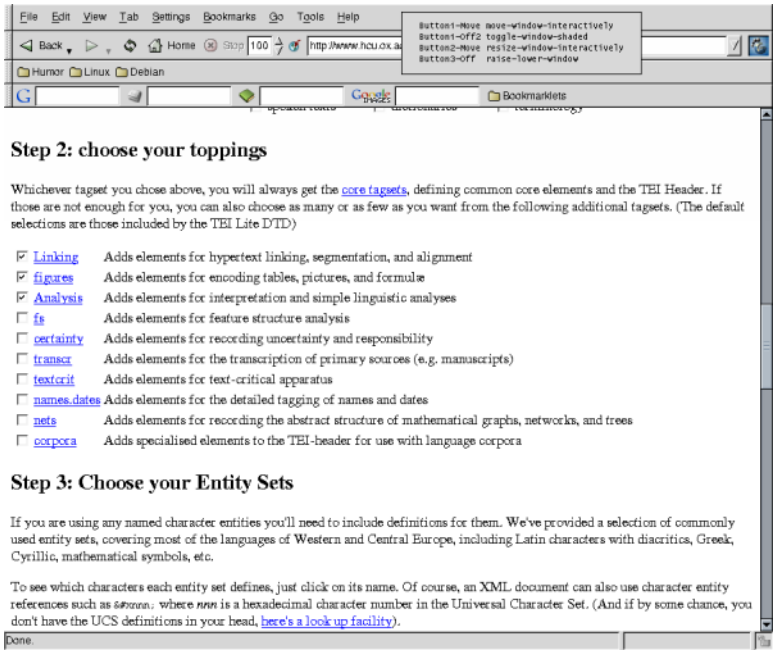

Figure 2: Choosing your optional tag set in the TEI Pizza Chef

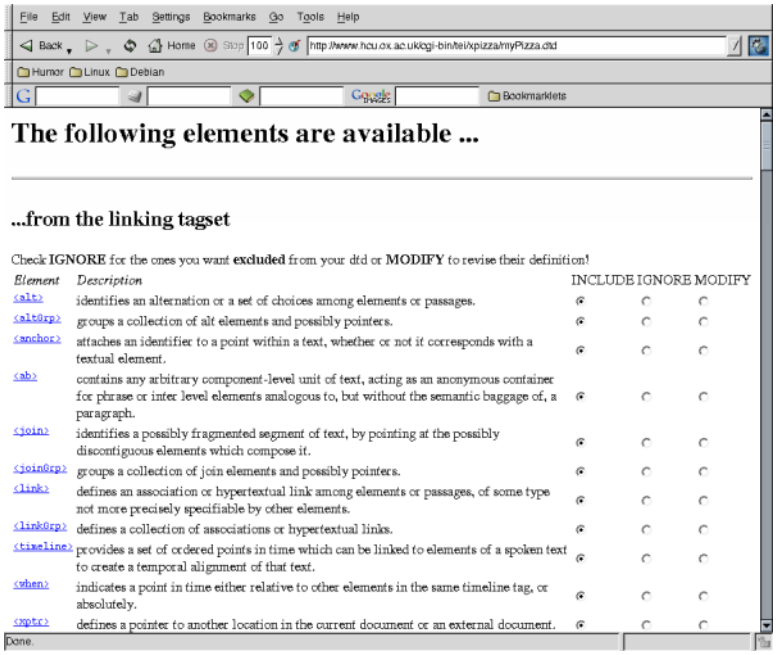

Figure 3: Deciding whether to include individual elements in the TEI Pizza Chef

same source to generate an different set of constraint rules, not expressed as XML or SGML, but (for example) as one of the XML schema languages. Such a capability has several obvious attractions:

1. We can experiment with using other XML vocabularies alongside the TEI, by using XML namespaces; obvious examples are MathML and SVG. Selective validation should be possible.

2. We have an opportunity to add more extensive datatyping and subtler constraints than are currently possible, possibly allowing us to re-express some of the descriptive text as formal constraints.

3. The Guidelines become even more amenable to internal cross-checking.

In addition, removing remaining traces of SGML-dependency from the TEI will stand us in good stead for coping with future developments.

Which constraint language should we change to? The choices we considered were UML, W3C Schema, and Relax NG. In many ways, the more abstract nature of UML is attractive, but our problem was lack of familiarity with the tools, and some problems with the initial attempt at modelling. W3C Schema looks very forbidding at present, so it was decided to experiment with Relax NG (`http://www.relaxng.org/`), which has the advantage of being small, complete, and with at least two implementations immediately available. We will not try to explain the syntax of RelaxNG here, as it already has an excellent tutorial.

As all good projects should, the conversion of the TEI Guidelines to express the constraints in Relax NG was done twice: initially as a rapid set of hand edits, and then in a more formal, repeatable, process. The constraints on the work were:

- That it be repeatable, so that the TEI editors could continue work on the current source and not have to make changes in two places.

- That it be possible to generate DTDs as well as Relax NG schemas, for backward compatibility.

The major job in the conversion is the extraction of `<elemDecl>` elements from the source, and converting each one to an equivalent fragment of Relax NG. This was done with a Perl script which converted, for example, a content model of

%om.RO; (#PCDATA | %m.dictionaryParts; | %m.phrase; | %m.inter; | %m.Incl;)*

to an intermediate XML form of

<schema> <group> <text/> <seqbar/> <element name="%m.dictionaryParts;"/> <seqbar/> <element name="%m.phrase;"/> <seqbar/> <element name="%m.inter;"/> <seqbar/> <element name="%m.Incl;"/> </group> <repstar/> </schema>

This can be converted, using an XSLT program, to

<rng:zeroOrMore xmlns:rng="http://relaxng.org/ns/structure/1.0" xmlns:ext="http://www.tei-c.org/relaxng/1.0"> <rng:choice> <rng:text/> <rng:ref ext:rend="entity" name="m.dictionaryParts"/> <rng:ref ext:rend="entity" name="m.phrase"/> <rng:ref ext:rend="entity" name="m.inter"/> <rng:ref ext:rend="entity" name="m.Incl"/> </rng:choice> </rng:zeroOrMore>

An extra bit of data has been added here (in its own namespace), the 'rend' attribute; this is put in to allow us to recreate the DTD at a later stage. In our use of RelaxNG, `<rng:ref name="foo"/>` would turn simply into the name 'foo' in a DTD, whereas `<rng:ref rend="entity" name="foo"/>` turns into '%foo;'. There are a number of small inelegancies like this in the system we have developed, dictated either by the need to allow for back-translation to DTD, or by the presence of partial content models like `| foo | bar`.

When the fragments of content model have been put back into the main text, the `<bibl>` element we saw earlier now has an `<elemDecl>` as follows:

<elemDecl omit="RR"> <rng:group xmlns:rng="http://relaxng.org/ns/structure/1.0"> <rng:optional> <rng:ref name="head"/> </rng:optional> <rng:oneOrMore> <rng:choice> <rng:ref name="bibl"/> <rng:ref name="biblStruct"/> <rng:ref name="biblFull"/> </rng:choice> </rng:oneOrMore> <rng:optional> <rng:ref name="trailer"/> </rng:optional> </rng:group> </elemDecl>

The 'omit' attribute is added to allow for recreating SGML DTDs.

The remaining task was to rewrite the existing DTD generator to reconstruct content models from the Relax NG elements, and to write a new processor which extracts all the Relax fragments and creates a set of useable schema files. This was straightforward, but ugly, as we did nothing to change the datatypes, for instance, from their SGML notations. Similarly, the Guidelines use such SGML/XML literals as `#IMPLIED` or `#REQUIRED`, which need interpreting. Ideally, these would be re-expressed in some more neutral form. However, a relatively simple XSLT script suffices to produce schemas which are acceptable to the Relax NG processors. This final stage was debugged by comparing the results with conversions of the TEI DTDs made by James Clark's *rngconv* program, which gave many useful hints.

5 Using TEI in Relax mode

Following suggestions by James Clark, the TEI Relax NG schemas use a 'marked section' mechanism similar to that in DTDs. The linking elements in the TEI, for example, are dealt with by an object called TEI.linking, initially defined as follows:

<define name="TEI.linking"> <ref name="IGNORE"/> </define>

The definition of IGNORE is

<define name="IGNORE"> <notAllowed/> </define>

while the opposite INCLUDE is

<define name="INCLUDE"> <empty/> </define>

so in a top-level schema we can say

<define name="TEI.linking"> <ref name="INCLUDE"/> </define>

which overrides the initial definition of IGNORE. In an element definition in the linking tagset, each element uses a TEI.linking reference:

<define name="link"> <ref name="TEI.linking"/> <element name="link"> <ref name="al.link"/> <empty/> </element> </define>

If TEI.linking expands to `<notAllowed/>`, the `<link>` element becomes unavailable. This allows us to construct a driver schema file like this, with additional declarations for datatype libraries, following Clark's lead:

<?xml version="1.0" encoding="iso-8859-1"?> <grammar xmlns="http://relaxng.org/ns/structure/1.0" xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0" datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"> <include href="rng/tei2.dtd.rng"> <define name="TEI.prose"> <ref name="INCLUDE"/> </define> <define name="TEI.figures"> <ref name="INCLUDE"/> </define> <define name="TEI.linking"> <ref name="INCLUDE"/> </define> </include> </grammar>

This says, informally, 'read all the Relax NG specifications referenced by rng/tei2.dtd.rng, and turn on the prose, figures and linking tagsets'. The schema can now be used to validate an XML instance file.

Let us consider a more complex example, which shows use of non-TEI namespaces, and extensions. The problem is to validate web pages written in TEI, but enhanced with XSP tags for processing by AxKit (`http://www.axkit.org/`), as this simple example shows:

<xsp:page language="Perl" xmlns:xsp="http://apache.org/xsp/core/v1"> <TEI.2> <teiHeader>...</teiHeader> <text><body> Page delivered <xsp:expr>scalar localtime</xsp:expr>. For further details, see <xptr url="http://www.oucs.ox.ac.uk/"/>. </body></text> </TEI.2> </xsp:page>

The `xsp` namespace contains Perl code which is to be interpreted on the web server before the page is delivered, and the `<xptr>` element uses a new 'url' attribute instead of the standard TEI entity reference. How can we express this in Relax NG? Our wrapper schema file first has to say that `<xsp:page>` can occur as the top level element, which means adding to the `<start>` element:

<grammar xmlns="http://relaxng.org/ns/structure/1.0" xmlns:xsp="http://apache.org/xsp/core/v1" xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0" datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"> <start combine="choice"> <choice> <ref name="xsp"/> <ref name="TEI.2"/> </choice> </start>

We can now define what 'xsp' means, in terms of some useful Relax NG wildcarding elements. This

means that we do not have to specify the precise rules about the 'xsp' namespace, just allow anything from it within <xsp:page>:

<define name="xsp"> <element name="xsp:page"> <oneOrMore> <attribute><anyName/><text/></attribute> </oneOrMore> <ref name="anyXSP"/> <ref name="TEI.2"/> </element> </define> <define name="anyXSP"> <element> <nsName ns="http://apache.org/xsp/core/v1"/> <zeroOrMore> <choice> <attribute> <anyName/> </attribute> <text/> <ref name="anyXSP"/> </choice> </zeroOrMore> </element> </define>

The additional attribute to <xptr> can be dealt with by redefining its attribute list inside an <include> which pulls in the main TEI schema. Each of the elements has a corresponding 'al.XX' object which is the equivalent of the XML ATTLIST:

<include href="rng/tei2.dtd.rng"> <define name="al.xptr" combine="interleave"> <ref name="a.global"/> <ref name="a.xPointer"/> <optional> <attribute name="url"><text/></attribute> </optional> </define> </include>

Aficionadoes of DTD hacking will perhaps find this a little more verbose, but the advantage of being able to deal with non-TEI namespaces is huge. It means that we no longer have to incorporate every possible markup scheme into the TEI.

So far we have gone little beyond what DTDs can do. But now the datatype libraries which can be used with Relax can be brought into play. The <birth> element currently has a promising attribute 'date', defined as

<attribute name="date"> <ref name="ISO-date"/> </attribute>

but ISO-date turns out to be simply defined as

<define name="ISO-date"> <text/> </define>

ie plain text. If we change that to

<define name="ISO-date"> <data type="date" datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/> </define>

the schema processor will check our birth dates to see if they are valid.

A great deal of work remains to be done to work out appropriate schema datatypes for the TEI elements and attributes.

Kohsuke Kawaguchi's Sun Multi-Schema XML Validator (http://www.sun.com/software/xml/developers/multischema/) has an even more interesting add-on (http://www.sun.com/software/xml/developers/schematronaddon/) which supports Schematron constraints on element content; these are expressed in a different namespace. To take a trivial example, if we have an element <email> whose content should contain an '@' sign, we can write a Relax NG rule as follows:

<define name="email"> <element name="email"> <ref name="a.global"/> <ref name="paraContent"/> <s:assert test="contains(.,'@')"> email address must contain an @ sign </s:assert> </element> </define>

The <s:assert> adds a test beyond the normal element content with a test (expressed in XPath) which must be passed.

We have as yet barely scratched the surface of the exciting ways in which TEI projects can start to enhance their data validation within the context of schemas.

6 Future work

It should be clear that while this work has derived a set of useable TEI schemas, there is plenty more to do. Not all remnants of SGML/XML have been removed (eg #IMPLIED), and the processing pipeline is sufficiently tortuous as to render it only useful to the TEI editors.

The work took place in parallel with the major changes involved in making the TEI Guidelines fully XML compliant (version P4). With the completion of this work, the TEI Technical Council can start on specifying the next release (P5), and the work described in this paper is one input to that process. It is to be hoped that the schema-ized TEI will become the mainstream development. If this is accepted, we will have two big tasks, and one important decision. Firstly, we will have to completely rewrite the

TEI Pizza Chef to produce either DTDs or schemas on demand; and secondly, we will have to examine all the attribute datatyping in the TEI to see whether it can make use of the richer schema datatypes. The *question* is whether we should consider rewriting the element content models (which are often very convoluted in the TEI) to take advantage of some of the extra facilities which Relax NG offers.

Readers interesting in getting copies of the TEI Relax NG schemas should contact the author. They will also appear on the TEI web site.

A Notes and Acknowledgements

This work was carried out as an informal exercise by the author alongside the technical work programme of the TEI Consortium in 2001-2002. It was not requested, and has not been validated, by the TEI Technical Council. Any views expressed in this paper are *not* necessarily those of the TEI.

I would like to thank: Lou Burnard (European Editor of the TEI) for encouraging this experimental work, having endless arguments about it, and improving the text; James Clark for stimulating us by showing how the TEI could be converted to Relax NG with a DTD conversion, and finding several errors in the DTD; Kohsuke Kawaguchi for extremely responsive development of his excellent Relax NG processor; and Leonor Barroca for detailed discussions about, and experiments with, UML. I also thank Norm Walsh, who first put the idea in my head when he mentioned that he planned to convert Docbook to Relax NG.

B Bibliography

[1] 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 P3)*. Ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 1994.

[2] N. Walsh and L. Muellner, *DocBook The Definitive Guide*, O'Reilly, Sebastopol, CA, USA, 1999.

[3] Donald E. Knuth, *Literate Programming*, Stanford University Center for the Study of Language and Information (CSLI Lecture Notes Number 27), Stanford, CA, USA, 1992.

[4] C.M. Sperberg-McQueen and Lou Burnard *The ODD System of Tag Set Documentation*, in *Consensus ex machina?* (Resumés du colloque internationale: , Laboratoire "Lexicométrie et textes politiques", École Normale Supérieure de Fontenay-Saint Cloud, 1994, pp 221-222.)

[5] C.M. Sperberg-McQueen and Lou Burnard. 'The Design of the TEI Encoding Scheme' in N. Ide. and J. Veronis, eds. *The Text Encoding Initiative: Background and Contexts*, special triple issue of *Computers and the Humanities*, 29:1, 1995, 17-39