Building TEI DTDs and Schemas on demand

Sebastian Rahtz March 2003

The Text Encoding Initiative Guidelines provide generic but detailed recommendations for the mark-up of electronic documents, in particular texts from the literary and linguistic domains. The TEI guidelines, converted to XML in 2001, are maintained in a high-level markup which mixes elements combination and content model rules with text documentation. A project to convert this to use RelaxNG internally was described at XML Europe 2002. Because the TEI is modular and extensible, it is accompanied by a web application which assists the user to define a subset and/or extension of the schema and creates an ad hoc DTD. This paper describes a new version of the program, which will enable users to generate DTDs, RelaxNG schemas, and W3C schemas on demand according to their specification, along with instance documentation.

The application, named Roma (the previous DTD-only incarnation was called Carthage), lets the user choose which TEI modules are needed, and allows them to include or exclude elements individually. It also supports the modification of existing elements, and the definition of new elements, with appropriate changes to TEI model classes. Standard components from other namespaces (SVG and MathML) can be included. Most of this can be done without commitment to which of the output formats is desired. At the end a 'flattened' schema or DTD is produced, containing only the necessary elements.

1 Introduction

The Text Encoding Initiative's *Guidelines for electronic text encoding and interchange* ([2]) provide exhaustive recommendations for the encoding of key features in literary and linguistic textual materials. These recommendations, instantiated by a modular XML-based architecture in which DTD fragments and documentation are combined according to user-specified requirements, are very effective and are widely adopted in digital library, language engineering, and many other projects (`http://www.tei-c.org/Applications/`).

One of the projects (`http://www.tei-c.org/Activities/META/`) of the TEI's Technical Council is to rewrite the Guidelines so that underlying metalanguage is independent of SGML or XML DTD language, allowing for automatic generation of schemas, DTDs, or any future constraint languages. The first stage of this work resulted in a set of RelaxNG (`http://wwww.relaxng.org/`) grammar files automatically generated from the Guidelines (available from `http://www.tei-c.org/Schemas/RelaxNG/P4X/`). This work was described at XML Europe 2002 ([1]), so we will only provide a summary description here, but we will have to recap some of the explanation.

Manipulation of the TEI is possible because the TEI is not maintained as DTD files, but in a literate programming (cf [4]) system which documents and describes elements in a largely abstract manner, and describes their interdependence using an independently-documented set of element classes. This is probably best demonstrated by an example. The `<persName>` element is specified by the following markup:

```
<tagDoc id="PERSNAME" usage="opt">
<gi>persName</gi>
<name>personal name</name>
<desc>contains a proper noun or proper-noun
phrase referring to a person, possibly including any or all of the
person's forenames, surnames, honorifics, added names, etc.</desc>
<attList>
<attDef usage="mwa">
<attName>type</attName>
<desc>describes the personal name more fully using an open-ended
list  of words or phrases which help to indicate the function, e.g.
<q>married name</q>, <q>maiden name</q>,
<q>pen name</q>, <q>religious name</q>, etc.</desc>
<datatype>CDATA</datatype>
<valDesc>Any string of characters.</valDesc>
<default>#IMPLIED</default>
</attDef>
</attList>
<exemplum>...</exemplum>
<remarks/>
<part type="top" name="ND"/>
<classes names="DEMOG NAMES DATA"/>
<elemDecl> %om.RR; ( #PCDATA | %m.personPart;
| %m.phrase; | %m.Incl; )* </elemDecl>
<ptr target="NDPER"/>
</tagDoc>
```

The key features here are

1. The general description of the purpose of the element, including examples (in `<exemplum>`, the contents of which are omitted here)
2. The list of attributes, specified using name, datatype, default etc
3. The module of the TEI to which `<persName>` belongs (`ND`, ie the module covering names and dates)

4. The classes to which this element contributes (DEMOG, NAMES, and DATA)

5. The content model for the element; this is also expressed in terms of classes, using the DTD markup `%m.personPart;`—any elements which say they are members of the 'personPart' class are allowed here.

This information allows a processor to construct a DTD fragment for the element as follows:

```
!ELEMENT persName ( #PCDATA | %m.personPart;
| %m.phrase; | %m.Incl; )* >
<!ATTLIST %n.persName;
%a.global;
%a.names;
type CDATA #IMPLIED
TEIform CDATA 'persName'  >
```

Note here the addition of more attributes, from the classes of which this element is a member.

The problem with the system described above is the dependence on explicit DTD content models, which are not amenable to processing using standard XML tools. We therefore replace the `<elemDecl>` with the following:

```
<elemDecl>
<rng:zeroOrMore xmlns:rng="http://relaxng.org/ns/structure/1.0">
<rng:choice>
<rng:text/>
<rng:ref name="m.personPart"/>
<rng:ref name="m.phrase"/>
<rng:ref name="m.Incl"/>
</rng:choice>
</rng:zeroOrMore>
</elemDecl>
```

This is much easier to analyze, and is (reasonably!) easy to turn back into DTD markup if needed. A processor can now assemble all the information needed to construct a complete RelaxNG grammar.

The translation of the TEI Guidelines to use RelaxNG markup to encode content models is fairly stable, and the challenge now is to find ways of making use of the extra power provided by schemas.

2 From Pizza Joint to Sushi Bar

In addition to the class system for maintaining relationships between elements, the TEI also works on the basis of a set of mutually exclusive[1] basic tag sets. The choice is between:

Prose This tagset is suitable for most documents most of the time

Verse This tagset adds specialist tagging for metrical analysis, rhyme-scheme etc to the basic verse markup already included in the core

Drama This tagset adds specialist tagging for cast lists, records of first performance, etc. to the basic drama markup already included in the core

Speech This tagset replaces the basic structure by one suitable for linguistic analysis of speech acts, etc.

[1] This statement is not *entirely* true.

Dictionaries This tagset replaces the basic structure with one containing detailed lexicographic features

Terminology This tagset replaces the basic structure with one specific to terminological databases

A normal TEI document will start with one of these scenarios, and then add modules from the following list:

Linking Adds elements for hypertext linking, segmentation, and alignment

Figures Adds elements for encoding tables, pictures, and formulae;

Analysis Adds elements for interpretation and simple linguistic analyses

FS Adds elements for feature structure analysis

Certainty Adds elements for recording uncertainty and responsibility

Transcription Adds elements for the transcription of primary sources (e.g. manuscripts)

Textcrit Adds elements for text-critical apparatus

Names & Dates Adds elements for the detailed tagging of names and dates

Nets Adds elements for recording the abstract structure of mathematical graphs, networks, and trees

Corpora Adds specialised elements to the TEI-header for use with language corpora

It is important to understand that a user *must* make sort of choice—there is no one TEI DTD or schema which is the default. In addition, the TEI has a clear system for extending the tagset, which again utilises the class system by allowing new elements to be added to classes, and to refer to existing classes.[2]

How does a casual user make sense of this complexity? It requires a good understanding of DTD or Schema languages to manipulate the right parameter entities or pattern definitions, so the TEI offers an interface for building customized views of the system. In the DTD-only release of the TEI, this is done using a web form and a utility called carthago; the job of this program was to 'compile' DTDs, expanding all parameter entity references and removing references elements which were not available.[3]

The web application is known as the TEI Pizza Chef, because it allows the customer to choose what toppings they want for a particular base. However, it has to leave most of the work to the user, by creating a pair of skeleton DTD extension files which the user downloads, edits, and uploads again. Editing these DTD files by hand is error-prone, fairly forbidding, and cannot be used to modify schemas. A revised system has therefore been built which attempts to keep all the knowledge or DTD or schema in the application itself, and simply ask the user to select options on web forms. This is fancifully known as the TEI Sushi Bar, following the model of an endless choice of clean, distinct, options continually being presented to the user, rather than a rather oily mound of congealing cheese and tomato. More precisely, the Sushi Bar is a web application running scripts known generically as roma.

Roma starts by asking the user to choose which base tagsets and extra modules they require. There are two interfaces, one verbose (Figure) and one for the expert (Figure). There are also two important choices to make:

[2] Adding new classes is a more complex exercise, not for the faint hearted

[3] Hence the name carthago; it builds of list of elements which are not needed, commenting as it goes *haec delenda sunt*, or *these must be destroyed*, echoing Scipio's repeated admonition to the Roman Senate of *Carthago delenda est*. Now, I hope, it is clear why the schema-based successor is called roma.

1. The user must indicate what sort of output is needed. The choice is between:
   - RelaxNG schema
   - compiled RelaxNG schema
   - compact RelaxNG schema
   - W3C schema
   - compiled DTD
2. The user must say whether they want to make modifications to the elements in the selected tagset. The choice is between:
   - Leave elements as they are
   - Configure elements, including them by default
   - Configure elements, excluding them by default
3. The user can say whether they want to add some new elements

The choices here affect the next stage. Firstly, if a DTD is requested, the user is allowed to choose some ISO entity sets to include (Figure). Secondly, if element configuration is requested, all the elements in the chosen tagsets are listed, with radio buttons which allow the element to be included in the result, or excluded (Figure). The links in this table are to the documentation of each element on the TEI web site. At this stage, the user can *rename* elements; the example shown in Figure has `<figure>` being renamed to `<graphic>`, and `<figDesc>` to `<caption>`, while `<table>`, `<row>`, `<cell>`, and `<formula>` are declared as unwanted. We will see shortly how this is implemented.

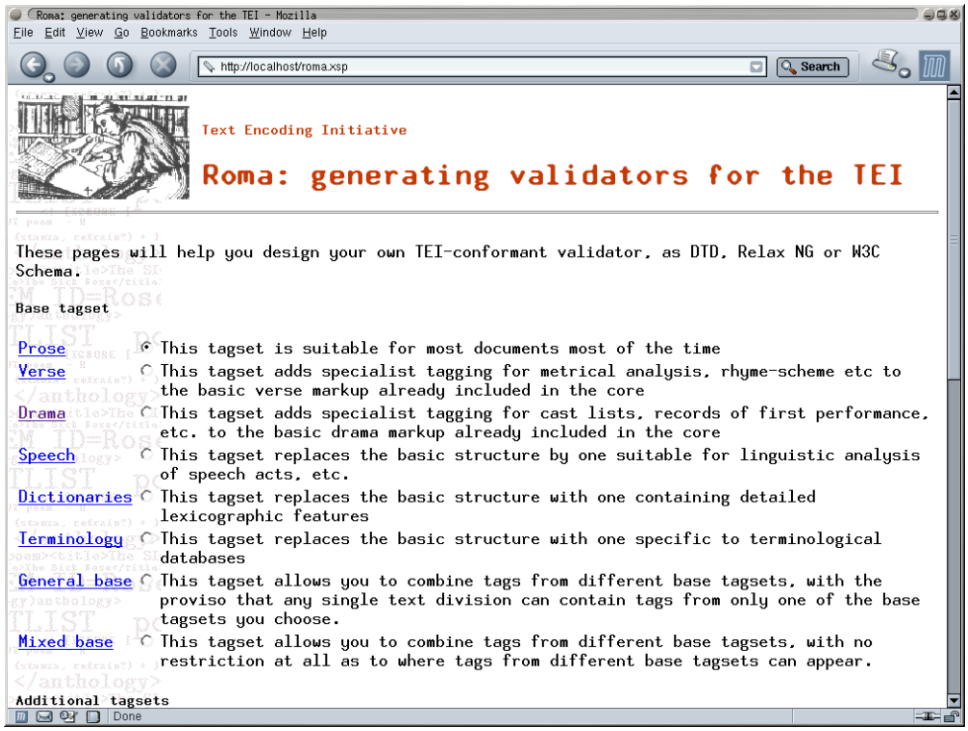

Figure 1: Roma stage 1, verbose mode

In the second stage of Roma, there is also a set of general options which can be turned on and off:

1. Whether date elements should be validated against an ISO date format (Schema only)

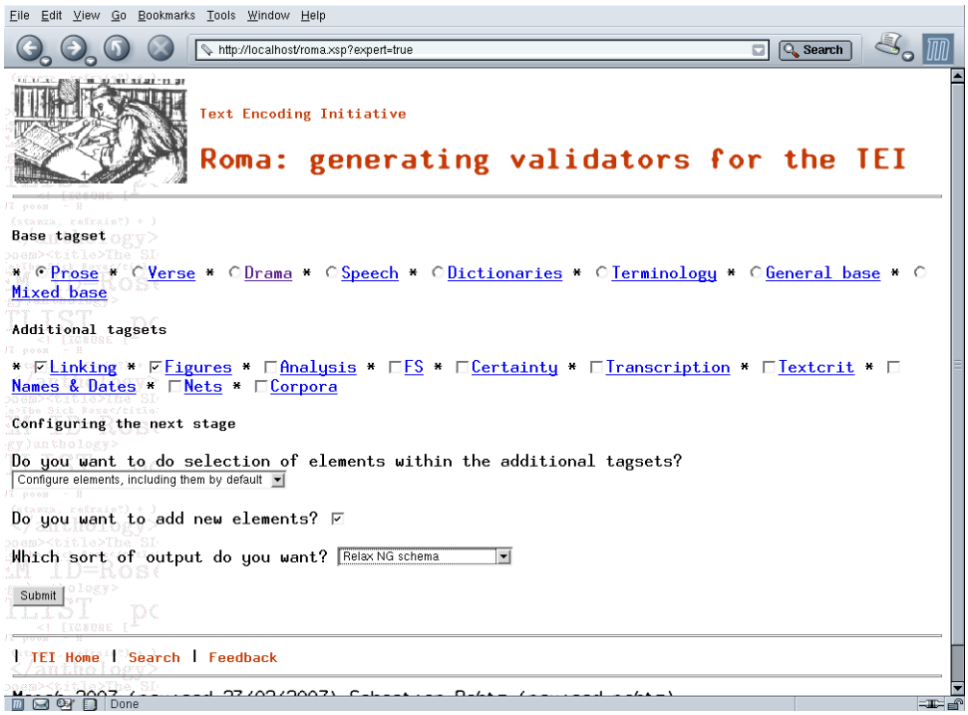

Figure 2: Roma stage 1, expert mode

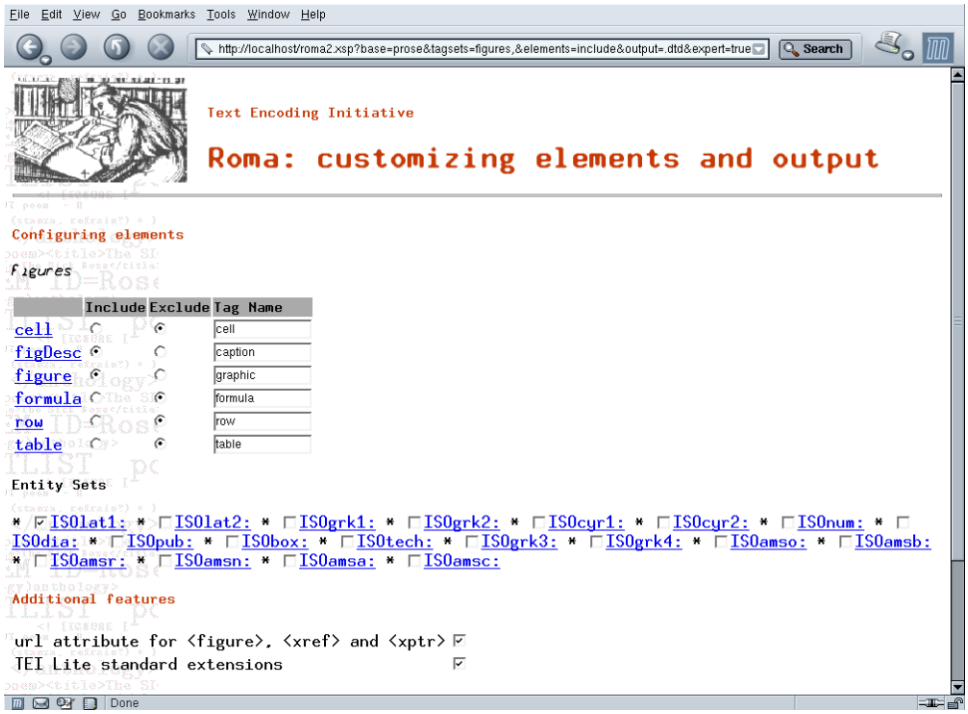

Figure 3: Roma stage 2, choosing entity sets

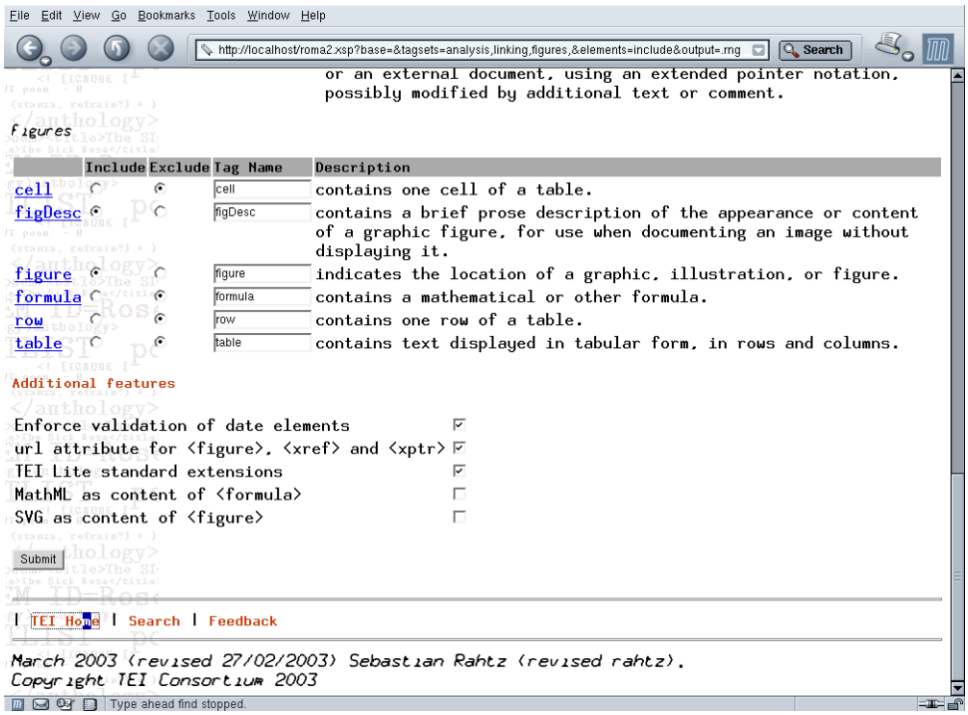

Figure 4: Roma stage 2, expert mode

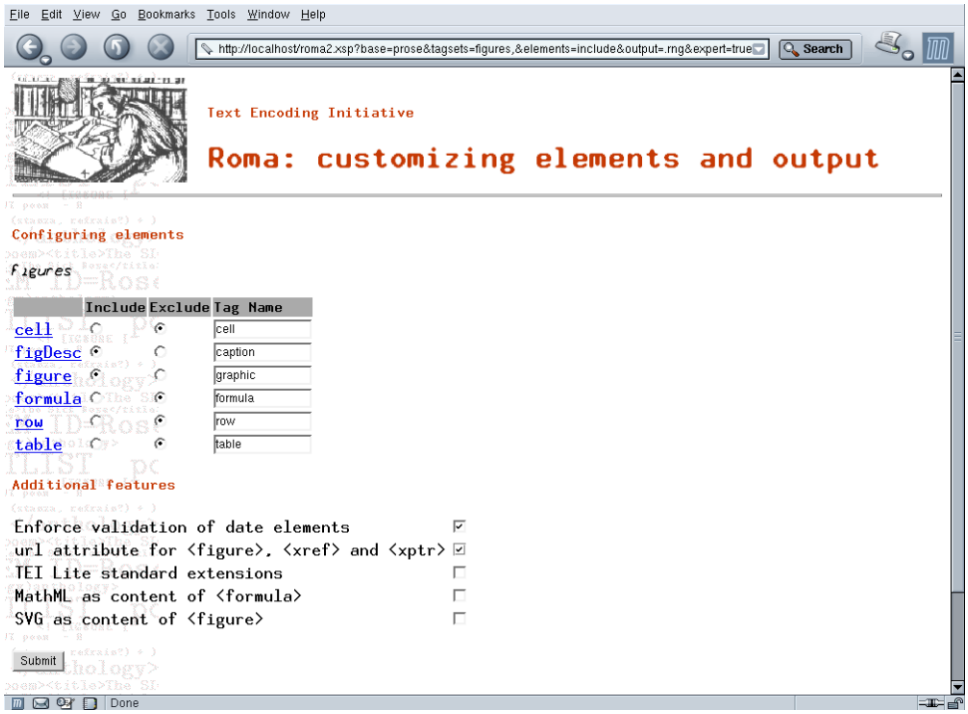

Figure 5: Roma stage 2, renaming elements

2. Whether <xptr>, <xref> and <figure> elements should support a url attribute to identify external resources[4]

3. Whether the standard extensions of the common subset of the TEI known as 'TEI Lite' should be activated

4. Whether the <formula> element should be redefined to insist on content being expressed as MathML (Schema only)

5. Whether the <figure> element should be redefined to allow a content of SVG (Scaleable Vector Graphics) elements (Schema only)

After all these choices are made, the Submit button prompts the user to download the resulting DTD or schema.

The look of the result depends on whether or not a compiled form has been selected. Given a simple set of choices, a RelaxNG grammar could result as follows:

```
<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="http://www.tei-c.org/Schemas/RelaxNG/P4X/tei2.dtd.rng">
<define name="TEI.prose"><ref name="INCLUDE"/></define>
<define name="TEI.figures"><ref name="INCLUDE"/></define>
<define name="formula"><notAllowed/></define>
<define name="table"><notAllowed/></define>
<define name="figDesc">
<element name="caption">
<ref name="c.figDesc"/>
</element>
</define>
<define name="row"><notAllowed/></define>
<define name="figure">
<element name="graphic">
<ref name="c.figure"/>
</element>
</define>
<define name="cell"><notAllowed/></define>
<!-- overrides to make ISOdate a formal datatype -->
<define name="ISO-date">
<data type="date"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
</define>
</include>
</grammar>
```

There are some important points to note here:

1. The basic structure is <include href="http://www.tei-c.org/Schemas/RelaxNG/P4X/tei2 *... a set of redefinitions of standard TEI patterns ...* </include>

2. TEI modules are turned on by redefining a pattern, eg <define name="TEI.figures"><ref name="INCLUDE"/></define>

3. In the same way, individual elements can be disallowed by setting their definition to <notAllowed>, eg <define name="table"><notAllowed/></define>

4. Elements are renamed by a redefinition of a pattern

[4]This is done using entities in 'traditional' TEI.

The last point deserves some more explanation. The original definition for <figure> is like this:

```
<define name="figure">
<element name="figure">
<ref name="c.figure"/>
</element>
</define>
```

that is, a pattern is defined called figure, which defines an element called <figure>, with a content model given in the pattern c.figure. By redefining figure as follows:

```
<define name="figure">
<element name="graphic">
<ref name="c.figure"/>
</element>
</define>
```

we define an element called <graphic>, which has the *same content model* as the old <figure>, and is inside athe pattern called figure. This is what other definitions will refer to; so anything which wants to include the 'figure' element will say <ref name="figure"/>, and it will not matter that the actual element is renamed. The *original* name of the element is preserved by an attribute called TEIform, defined as <attribute name="TEIform" a:defaultValue="figure"> <text/> </attribute>, so it is easy to relate this changed setup to the basic TEI. The renaming feature may be extended in future to allow complete translations of the TEI element names to predefined language sets, allowing the user to simply request "all elements in Spanish, please".

If a compiled output is requested, then the skeleton DTD or Schema will be put through a flattening process to remove redundant elements and references to external files. This has the advantage that a single file is produced, which considerably aids portability, and the removal of unused elements can make it much smaller.

DTD flattening is performed by the existing carthago application, and schema flattening is performed by an XSLT transform of a RelaxNG grammar. The other outputs (compact RelaxNG and W3C Schema) are done by calls to James Clark's trang program (http://www.thaiopensource.com/trang/).

MathML and SVG inclusion are managed by simplying <include>ing the relevant RelaxNG grammars, each in their own namespace.

3 Extending the TEI

We have so far seen examples of simply choosing subsets of the TEI, or adding standard new features. What if we want to add some elements? This may be for one of two reasons:

1. To add an element which is effectively a clone of an existing element, perhaps with an assumed attribute value, to make the text easier to edit and read. For example, we could mark a set of exercise steps with <list type='steps'>, but it would be friendlier to allow <steplist>, even though the processing would be identical.

2. To add a new element to an existing class. For example, the elements for describing an address do not include anywhere to put a personal URL, so we want to add a new element parallel to <postCode> and <street>.

If the user chooses to add elements, they are asked to decide which of these two situations they want to address, and to give the element a name and description. In Figure we show

the addition of the <homeurl> element, in the addrPart class. Of course, this assumes some familiarity with the TEI class system, (see section *4. TEI classes* for a summary of the TEI classes) and the interface is not yet friendly enough for someone completely new to the TEI. The list of elements and classes are derived, of course, dynamically from the TEI Guidelines.

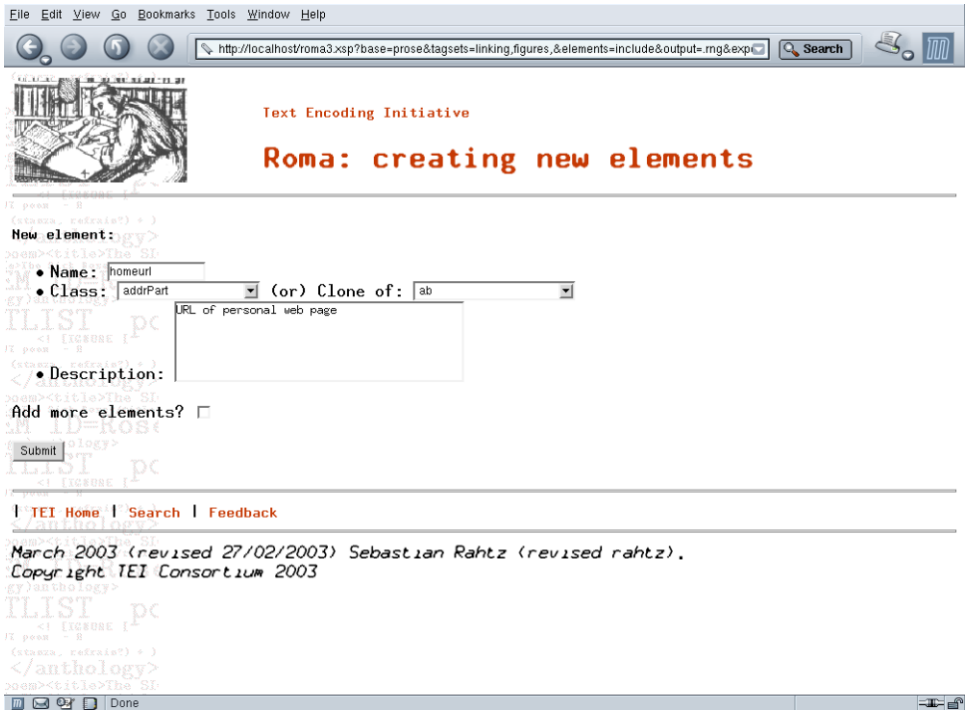

Figure 6: Creating new elements

There are three further facilities which Roma does not yet provide:

1. Adding elements which do not simply follow the class system, but have arbitrary content models and attribute lists. The problem here is how to ask the user to specify the new material without directly writings schema code. It remains to see how many requests we will receive for this feature.

2. Changing or limited the content model of elements which do not follow the class system fully. The correct answer to this may be to revise the TEI so that all elements *do* use the class system 100%, but in the short-term this is unrealistic. It may be possible to devise an interface to editing content models.

3. Adding entire classes to the TEI. This is a complex matter, which it is unlikely we can provide in a simple web interface.

4 TEI classes

Here is a list of the currently defined classes of the TEI system:

addrPart groups elements which may constitute a postal or other form of address.

agent groups elements which contain names of individuals or corporate bodies.

analysis default declaration for class *analysis*: when the additional tag set for simple analysis is not selected, no attributes are defined for this class.

analysis defines a set of attributes for associating specific analyses or interpretations with appropriate portions of a text, which are enabled for all elements when the additional tag set for simple analysis is selected.

baseStandard groups elements in a writing system which refer to some public or private standard as part of the basis for the writing system declaration

bibl groups elements containing a bibliographic description.

biblPart groups elements which can appear only within bibliographic citation elements.

binary elements which express binary values in feature structures.

boolean groups elements which express Boolean values in feature structures.

chunk groups elements which can occur between, but not within, paragraphs and other chunks.

common groups common chunk- and inter-level elements.

comp.dictionaries groups those component-level elements which are unique to the base tag set for dictionaries.

comp.drama groups those component-level elements which are specific to performance texts.

comp.spoken groups those elements which appear at the component level in spoken texts only.

comp.terminology groups component-level elements unique to the base tag set for terminological data.

comp.verse groups component level elements unique to the base tag set for verse.

complexVal groups elements which express complex feature values in feature structures.

data groups phrase-level elements containing names, dates, numbers, measures, and similar data.

date groups elements containing a date specifications.

declarable groups elements which may be independently selected (using the special purpose decls attribute) from a candidate list of declarations within a TEI header.

declaring groups elements which may be independently associated with a particular declarable element within the header, thus overriding the inherited default for that element.

demographic groups elements describing demographic characteristics of the participants in a linguistic interaction.

dictionaries default declaration for class *dictionaries*: when the base tag set for dictionaries is not selected, no attributes are defined for this class.

dictionaries defines a set of global attributes available on elements in the base tag set for dictionaries.

dictionaryParts groups all elements defined specifically for dictionaries.

dictionaryTopLevel groups related parts of a dictionary entry forming a coherent subdivision, for example a particular sense, homonym, etc.

divbot groups elements which can occur at the end of a text division; for example, trailer, byline, etc.

divn defines a set of attributes common to all elements which behave in the same way as divisions.

divtop groups elements which can occur at the start of any division class element.

dramafront groups elements which appear at the level of divisions within front or back matter of performance texts only.

edit defines a group of attributes common to the phrase-level elements used for simple editorial correction and transcription.

edit groups phrase-level elements for simple editorial correction and transcription.

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.

enjamb groups elements bearing the `enjamb` attribute.

entries groups the different styles of dictionary entries.

featureVal groups elements which express feature values in feature structures.

fmchunk groups elements which can occur as direct constituents of front matter, when a full title page is not given.

formInfo groups elements allowed within a `<form>` element in a dictionary.

formPointers groups elements in the dictionary base which point at orthographic or pronunciation forms of the headword.

fragmentary groups elements which mark the beginning or ending of a fragmentary manuscript or other witness.

front groups elements which appear at the level of divisions within front or back matter.

global defines a set of attributes available to all components of the writing system declaration.

global defines a set of attributes common to all elements in the TEI encoding scheme.

gramInfo groups those elements allowed within a `<gramGrp>` element in a dictionary.

hqinter groups elements related to highlighting which can appear either within or between chunk-level elements.

hqphrase groups phrase-level elements related to highlighting.

Incl groups empty elements which may appear at any point within a TEI text.

inter groups elements of the intermediate (inter-level) class: these elements can occur both within and and between paragraphs or other chunk-level elements.

interpret defines the set of attributes common to this group of interpretative elements.

linking default declaration for class *linking*: when the additional tag set for linking is not selected, no attributes are defined for this class.

linking defines a set of attributes for hypertext and other linking, which are enabled for all elements when the additional tag set for linking is selected.

lists groups all list-like elements.

loc groups elements used for purposes of location and reference

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.

metrical defines a set of attributes which certain elements may use to represent metrical information.

morphInfo groups elements which provide morphological information within the dictionary tag set.

names groups those elements which refer to named persons, places, organizations etc.

notes groups all note-like elements.

personPart groups those elements which form part of a personal name.

phrase.verse groups phrase-level elements which may appear within verse only.

phrase groups those elements which can occur at the level of individual words or phrases.

placePart groups those elements which form part of a place name.

pointer defines a set of attributes used by all elements which point to other elements by means of one or more `IDREF` values.

pointerGroup defines a set of attributes common to all elements which enclose groups of pointer elements.

readings defines a set of attributes common to all elements representing variant readings in text critical work.

refsys groups milestone-style elements used to represent reference systems

seg groups elements used for arbitrary segmentation.

sgmlKeywords groups elements whose content is an SGML or XML identifier or tag of some sort (generic identifier of an element type, name of an attribute, etc.).

singleVal group elements which express single feature values in feature structures.

stageDirection groups elements containing specialized stage directions defined in the additional tag set for performance texts.

temporalExpr groups component elements of temporal expressions involving dates and time, and defines an additional set of attributes common to them.

terminology default declaration for class *terminology*: when the base tag set for terminological data is not selected, no attributes are defined for this class.

terminology defines attributes for all elements in documents which use the base tag set for terminological data.

terminologyInclusions groups elements which may be included at any point within a terminology entry.

terminologyMisc groups elements which can appear together at various points in terminological entries.

timed defines a set of attributes common to those elements which have a duration in time, expressed either absolutely or by reference to an alignment map.

tpParts groups those elements which can occur as direct constituents of a title page (`<docTitle>`, `<docAuth>`, `<docImprint>`, `<epigraph>`, etc.)

typed defines a set of attributes which can be used to classify or subclassify certain elements in any way.

xPointer defines a set of attributes used by all those elements which use the TEI extended pointer mechanism to point at locations which have neither an SGML nor an XML ID.

5 Conclusions

The increasing power provided by schemas, and the stress on modularity, argue in favour of moving towards (conceptual) *two stage validation*. In the first phase, the important check is that the document uses the right vocabulary, in our case meaning the 441 elements currently described by the TEI. The structure here can be quite loose. In the second phase, which can depend on individual projects, validation can be a lot more precise, with detailed datatyping and inter-dependency validation. For example, the basic rule may say that an `<text>` must have a `<author>`, `<title>` and `<date>`, but be agnostic about their order. A particular project may wish to enforce a rule that they must occur in a fixed order; or it may wish to *more* limited than the base schema, and say that `<date>` is not permitted at all. Thus a typical document may be checked once to ensure that it uses TEI vocabulary and broad grammatical structure, and then checked again to make sure it talks the right dialect.

The relevance of this work is that it shows a way forward for XML users which does not involve low-level interaction with DTDs or Schemas. Unlike the graphic direct manipulation tools in eg XML Spy, the Roma tool works at the level of the TEI class system. Together with the support for other namespaces via schemas, these tools take the TEI one step further on the road to a universal markup language.

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/Council/tcw03.html`) of the TEI Council in 2003. It is still experimental and does not form a formal part of the TEI.

I am grateful to Norm Walsh and Lou Burnard, and the other members of the Taskforce, for stimulating discussion on this and related subjects; I was also delighted to discover Daniel Veillard's work on a new RelaxNG validator (now part of libxml2) while I was writing this paper, and to have the chance of contributing towards debugging the software with TEI examples.

B Bibliography

[1] Sebastian Rahtz, 'Converting to schema: the TEI and RelaxNG', paper presented at XML Europe 2002, Barcelona, May 2002.

[2] 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.

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

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

[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