Kyoto University, Institute for Research in Humanities

Summer Seminar 2005: The World of XML Markup

Christian Wittern

Contents

1 General Introduction to Markup **2**

1.1 The Concept of Text . . . 2
1.2 Text Encoding . . . 2
1.3 Markup . . . 2
1.3.1 Presentational Markup . . . 2
1.3.2 Procedural Markup . . . 3
1.3.3 Descriptive Markup . . . 3
1.4 Text as a Ordered Hierarchy of Content Objects (OHCO) . . . 4

2 Introduction to XML **4**

2.1 History . . . 4
2.2 HTML . . . 5
2.3 From SGML to XML and Beyond . . . 5
2.4 A Closer Look at XML . . . 5
2.5 XML Schema Languages . . . 6

3 Introduction to TEI **7**

3.1 Background and history . . . 7
3.2 Overview of the TEI encoding scheme . . . 8
3.3 General structure of a TEI document . . . 8
3.3.1 `<teiHeader>` . . . 9
3.3.2 `<text>` . . . 9
3.3.3 Marking Divisions within a Text . . . 9
3.3.4 Structural features . . . 9
3.3.5 Floating features . . . 10
3.4 Using the TEI scheme . . . 10

1 General Introduction to Markup

1.1 The Concept of Text

Depending on the context of its usage, text can be:

* In everyday usage, a broad term for something written to express something.
* In linguistics, a communicative act, fulfilling the principles of textuality.
* In literary theory, text is the object studied, be it a novel, poem, film, advertisement or anything else with a linguistic component. This broad use is inspired by semiotics and cultural studies of the 1980s.
* In information processing, text refers to character data.

Text can be very simple or complexely structured. Structure usually makes it easier to understand. Text depends on some notation, usually a script made up of characters.

1.2 Text Encoding

Text encoding is the process of transcribing a text in digital form. It is sometimes confused with character encoding, which is the act of assigning characters to the individual items observed in the stream of text. Text encoding comprises character encoding, but goes beyond that, since it is also concerned with re-creating the structure of a text in electronic form. Text encoding is sometimes also confused with markup, which is a methodology used in text encoding to express information about structure, status or other special features of a text.

1.3 Markup

The term markup derives from the annotations or symbols editors would use in traditional publishing to convey to the printers information about how the text should be printed: Size and weight of characters, position on the page, size of margins etc.

The notion of markup has been generalized by some theorists of markup to include certain aspects of written communications, for example page layout, typography and punctuation.

Markup of digital text is usually classified in different categories according to the way it is used: presentational, procedural, descriptive.

1.3.1 Presentational Markup

Presentational markup arranges the text in the electronic text in the same way as it is intended to appear on the page, using space characters, line-feed characters etc. This can be send to a output device (a printer or screen) without further processing.

1.3.2 Procedural Markup

Procedural markup inserts special codes into the electronic text to produce the desired effect in the output. A special software programm, called a ‘formatter’ is used to interpret these special codes and produce an intermediate version of the electronic text that is then send to the output device.

To give an example, there might be a special code that switches to an italic style, for example `.i` or `<i>` and also to switch back to the previous state, `.i` or `</i>`. A formatter will then interpret these codes and take the necessary actions to ensure that the desired result is achieved.

1.3.3 Descriptive Markup

Instead of inserting the desired formatting codes directly, descriptive markup inserts information into the text that describes or identifies features of a text. This introduces an additional layer of abstraction, and information about how to render these features can be held separately. A formatter then uses both the descriptive markup and the formatting information together to produce the desired result.

Instead of specifying the desired result directly, as above, we might for example observe that there is a title of a book mentioned in the text and mark it as `<title>`. We then specify that we want to have our titles rendered with an italic font for printing. On a screen however, we might find that we would prefer to render it with a color rather than italic, to improve legibility.

Descriptive markup offers not only more flexibility in rendering for different devices, the information contained in the descriptive markup can be used for many additional purposes, for example to provide a list of books mentioned, a topical index etc.

Descriptive markup offers many advantage, both for authoring composition or transcription of texts, and for publication. Some of the advantages are:

* Composition is simplified
* Structure-oriented editing is supported
* Natural editing tools are supported
* Alternative views of a text are possible
* Formatting can be generically specified and modified
* Indexes, appendices, etc. can be automated
* Many output devices can be supported
* Portability and durability is maximized
* Information retrieval is supported
* Analytical procedures are supported

Descriptive markup is thus the most versatile and flexible method of markup and this will be the form of markup that we will apply here.

1.4 Text as a Ordered Hierarchy of Content Objects (OHCO)

The many advantages of descriptive markup and the methodologies used to implement it were so successful, that for some people it seemed to suggest that it was not only a handy way of working with text, but deeply, profoundly the ‘correct’: ‘descriptive markup is not just the best approach ... it is the best imaginable approach’ (Coombs et.al., 1987). This view assumes that only the model used by the methodologies employed for descriptive markup reflect a correct view of ‘what text really is’ (DeRose et.al., 1990). In this model, a text is view is determined by its logical structure as a nested hierarchy of chapters, sections, paragraphs, sentences, and so on, but not as features of the physical representation of a text, like pages, columns, lines, font shifts, spacing and so on. According to this view then, a text is simply a ‘Ordered Hierarchy of Content Objects’ (OHCO) and descriptive markup works well, because it identifies that hierarchy and makes it explicit.

The OHCO view provides a powerful model to text encoding and allows elegant and convenient handling of many features and constraints found in real-life texts (for example a section header is always at the start of a section, not somewhere in the middle, lines of a poem are within a stanza etc.), but there are also limits and cases where it can not be applied, e.g. sentences do not nest within lines of poetry or quotations interrupted by an authorial voice. Nevertheless, it proved so successful and clearly superior to other views of a text (for example as a simple sequence of characters) that it became the dominating view of markup languages like SGML and XML, and is widely applied in text encoding.

2 Introduction to XML

2.1 History

Since the 1960s, efforts have been made to standardize and formalize methods of descriptive markup. While there were a number of projects, the one headed by Charles Goldfarb at IBM to develop a Generalized Markup Language proved to be most succesful and was further developed with international standards organizations to the SGML, which was published by the ISO in 1986 as ISO 8879: Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML). Besides its name, SGML is more a metalanguage (or ’grammar’ to be more precise) than a markup language, it allows to say things like what characters are used to distinguish the content of a text from the added information about the text. It can be used to define markup languages for specific purposes, these are then called applications of SGML. Such an application is constructed by developing a specification for a class of documents that lines out what elements can be used and at what place in the document. This specification, which is called DTD (Document Type Definition) can then be used to check a given document against the specification and thus quickly find out if the document conforms to the specification. The software used for this purpose is called a parser and the process of checking the document validation.

2.2 HTML

One such application of SGML is the ‘Hyper Text Markup Language (HTML)’, which is the underlying markup language of the World Wide Web. It defines a set of elements ('tags') that can be used to write Web pages, including things like `<p>` for paragraphs, `<h1>`, `<h2>` etc. for headers of different levels, `<img>` to include images into the page and `<a>` for anchors that provide links to other pages. While this is a very limited set, it was easy to write software to support it (this type of software was then called a browser or web-browser) and thus it proved good enough to drive the massive growth of the WWW and continues to support it.

The small set of elements available in HTML, however, was soon seen as a limitation and manufacturers of browsers tried to compete by inventing their own specialized tags, that would be unknown to other browsers. Another problem that soon arose was that the SGML specification was complicated and writing validation software was difficult. Most users did therefore not validate their HTML documents, but instead looked if the browser they used could display the document. The browsers on the other hand tried to be as permissive as possible to enlarge their base of users.

2.3 From SGML to XML and Beyond

To solve this situation and provide the WWW with a better path to growth, the World-Wide-Web Consortium (W3C), which had been founded to ensure that the WWW would continue to operate on open standards and technologies that were available to everybody, charged a workgroup of leading experts with the task of developing a simplified version of SGML that would be easier to implement and to use. The result of this was published as a W3C recommendation in February 1998 as the eXtensible Markup Language (XML). XML allowed easy definition of document types, but could also operate without. It was based on Unicode and had simple but efficient rules for validation. It proved an immediate success and a whole industry developed around XML and related standards, nowadays XML is not only widely used for text encoding, but also for metadata, data exchange and messaging. There are specifications of XML markup vocabularies for computer graphics, mathematical formulae, chemical fusions, geographical locations, bibliographical records, business transactions, news reports, weather and many others.

2.4 A Closer Look at XML

A XML document can contain seven different kinds of components:

* Elements
* Text
* Attributes
* Entities
* Comments
* Processing instructions
* CDATA

In addition to that, there are some components that can precede a document to indicate its DTD (the ‘DOCTYPE declaration’), and its encoding and to identify it as a XML document.

A fairly minimal XML document would look like this:

```
<firstDoc n="1">
  <p>This is an instance of a "firstDoc" document</p>
</firstDoc>
```

There are some additional rules that govern a XML file that had not been mentioned so far. One of them is that the first element has to contain all other elements, this is the ‘root elelemt’. In the above example, the root element is called firstDoc. It contains one other element, which is called <p> in this case. Another important rule is that all elements have to complete nest into each other. The following structure is thus not valid XML:

```
<p>Two <a>elements, <b>overlapping</a> each</b> other.</p>
```

It is not valid, because after the start of element <a>, element <b> is opened, but then element <a> is closed before <b>. To make this valid XML, b has to close first, then a:

```
<p>Two <a>elements, <b>overlapping</b> each</a> other.</p>
```

The element <firstDoc> in the example above contains an attribute with the name ‘n’ and the value ‘1’. Attributes usually convey additional information about the text contained in the element, there are also some rules that govern the way attributes are specified. The most important rule is that the beginning and end of the attribute has to be delimited with either the single or the double quote (e.g. ’ or ") and they have to be used in pairs. The name and the value have to be connected with the = character.

While this might seem confusing at the beginning, a dedicated XML editor can take care of most of the details, and one can used to it quickly.

2.5 XML Schema Languages

As mentioned above, the original XML specification allowed for the specifications of document structure and content (a socalled Schema) using a DTD. To show how this works, As an example, we can extend the example given above to include a definition of the expected document:

```
<!DOCTYPE firstDoc [
  <!ELEMENT firstDoc (p+) >
  <!ATTLIST firstDoc n CDATA #IMPLIED>
  <!ELEMENT p (#PCDATA)>
]>
<firstDoc n="1">
  <p>This is an instance of a "firstDoc" document</p>
</firstDoc>
```

This example says in DTD language that the firstDoc document type consists of one <firstDoc> element followed by one or more <p> elements. The <firstDoc> element might optionally have an attribute n. Details about the DTD language can be found in XML 入門 p.XX.

In the meantime, other schema languages have been developed. At the moment, there are three schema languages widely used with XML documents:

* DTD language, originally developed for SGML (*.dtd)
* XML Schema language, developed by the W3C (*.xsd)
* Relax NG Schema language, developed by James Clark and Murata Makoto, adopted by OASIS and the ISO (*.rng or *.rnc)

We will not detail the differences here, for the moment it will be enough to know that there exist several languages. They all serve the purpose of defining a classes of documents, and can be used to validate a given document, but they differ in the way constraints are expressed. In the table above, the file extension usually given to the files containing the schema are noted, so that the type of the schema usually can be guessed from the file extension. As shown in Figure 1, the XML editor ＜oXygen/＞ for example has a function that can be used to associate a schema with a given file to validate it. For furthe information on how to use ＜oXygen/＞, please refer to the Users Guide.

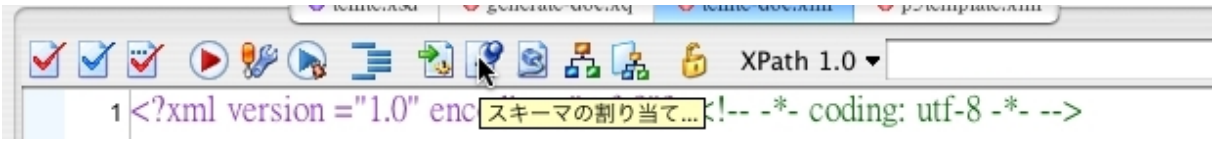

Figure 1

3 Introduction to TEI

3.1 Background and history

The creation of machine-readable texts for research in disciplines of the humanities began early and grew fast. The beginning of literary text encoding is usually dated to 1949, when Father Roberto Busa used IBM punched-card equipment to start work on his Index Thomisticus. In the mid-1960s, there were three academic journals focusing on humanities computing and a list of literary works in machine-readable form published in 1966 already required 25 pages (Carlson, 1967).

At a conference on computers and literature in 1965, a meeting was conveyed to discuss ‘the establishment of a standard format for the encoding of text’ (Kay, 1965). The first concerns were about a consistent identification and encoding of characters, but encoding of the structural and analytic features of a text was attempted as well.

With the passage of time, these issues became more pressing, and in 1987, the Association for Computers in the Humanities (ACH) convened a meeting at Vassar College, NY that was attended by 32 specialists from different disciplines, representing professional societies, libraries, archives and projects from North America, Europe and Aisa. The result was a set of principles, which became the design principles for the Text Encoding Initiative (TEI), a project that was sponsored by the ACH and two other learned societies [TEI ED P1, 1988].

The first draft of the recommendation (P1) has been released in June 1990. After that, the effort was reorganized in 15 working groups and a revised proposal (P3), intended for a larger audience was published for the first time also in print in May 1994 [TEI P3, 1994]. These guidelines have been a huge success and are now widely used around the globe in nearly every humanities computing project involving texts. In 2002, the TEI Consortium, which was formed in 2000 released ”P4”, which was a re-expression of the SGML-based P3 using XML. Another major revision (P5) is currently under way, which will support more XML features like namespaces, different schema languages and also add new modules for additional characters (’gaiji’) and manuscript descriptions. The significance lies in the fact that it is actively developed and actively maintained through a very large international collaboration of scholars from many disciplines and nations, with many different interests and viewpoints, and with the active participation of other specialists from many different professions and institutions. The work is now carried out in a transparent way, with the current state of proposal and documentation available to the public on a sourceforge website at [`http://tei.sf.net`].

3.2 Overview of the TEI encoding scheme

To accommodate the wide variety of different types of texts that can be encoded using TEI, the TEI encoding scheme has been divided into several modules, that can be selected and combined as needed. Some of these modules provide the basic tag set for a specific type of document, such as prose works, poetry, drama and dictionaries. Other modules provide additional, special purpose tag sets, for example for text-critical encoding of different text witnesses, transcription of primary sources, figures, linking, gaiji, names and dates and so on. In addition to selecting modules, individual elements can be added or deleted from the resulting set, existing elements can be modified and new elements added. This is all done through a customization mechanism defined within the TEI Guidelines and implemented in software available from the TEI Consortium (Note: There are two implementation of customization programs as web applications, one for P4 at [`http://www.tei-c.org/Pizza`], the other for P5 at [`http://www.tei-c.org/Roma`].) A given customization is then called a view of the TEI encoding scheme.

For those who find the prospect of starting out with developing a customization of TEI intimidating, there is ‘TEI Lite’, a view of the TEI schema that has been created with the most frequenly used element to allow an easy start. It can also serve as an example for how customizations are created and comes with a general tutorial on the usage of TEI. It is available at [`http://www.tei-c.org/Lite/`] in English and has also been translated to some other languages, including Japanese at [`http://www.tei-c.org/Lite/teiu5_jp.html`].

3.3 General structure of a TEI document

Every TEI document has two general parts: the `<teiHeader>` and the `<text>`. The `<teiHeader>` holds the ‘metadata’, information about the electronic edition, while the `<text>` holds the text itself.

3.3.1 <teiHeader>

The <teiHeader> contains the kind of information that for printed books would be held in library catalogs: Information about the creator and publisher, date of publication, conditions of use etc. All this goes in the <fileDesc> element and its specialised sub-elelements; the <fileDesc> is the only child element of the <teiHeader> which is required for all TEI documents.

There are also some additional items that are only of interest to electronic texts and would not appear in a library catalog, information about these is contained in additional elements of the header.

* <encodingDesc> documents the relationship between an electronic text and the source or sources from which it was derived.
* <profileDesc> provides a detailed description of non-bibliographic aspects of a text, specifically the languages and sublanguages used, the situation in which it was produced.
* <revisionDesc> summarizes the revision history for a file.

3.3.2 <text>

The <text> element contains a single text of any kind, either unitary or composite, for example a poem or drama, a collection of essays, a novel, a dictionary, or a corpus sample.

Since many texts published as books in the european tradition have this structure, the <text> may contain mainly the three elements <front>, <body> and <back>. A <front> element holds content from the frontmatter of a book, i.e. from the title page, table of content, dedication etc. The <body> element will contain the main body of the text and the <back> element is set aside for anything that comes in the printed book after the main body of text. Documents that do not follow this tri-partite structure will use only the <body> element to hold the text.

3.3.3 Marking Divisions within a Text

The TEI recommendations categorise document elements as either ”structural” or ”floating”. Structural elements are constrained as to where they may appear in a document; for example a <head> or heading may not appear in the middle of a <list>. Floating elements, as the name suggests, are less constrained and may appear almost anywhere in a text: examples include <note> or <date>. Intermediate between the two categories are so-called ”crystals”: these are floating features the contents of which have an inherent structure, for example <list> or <cit> elements.

3.3.4 Structural features

The current TEI recommendations define a general purpose hierarchic structure, which has been found to be suitable for a very large (perhaps surprisingly large) variety of textual sources. In this, a text is divided into an optional <front>, a <body> and an optional <back>. The body of a text may be a series of paragraphs (marked with ...), or it may be divided into chapters, sections, subsections, etc. In the latter case, the <body> is divided into a series of elements known generically as ”div”s. The largest subdivision of a given text is tagged

(with an appropriate type attribute indicating the type of division), all the nested divisions will then also be marked as . Written prose texts may also be further subdivided into <p>s (paragraphs). For verse texts, metrical lines are tagged with the <l> tag, optionally grouped under a <lg> (line group).

3.3.5 Floating features

As mentioned above, the TEI Guidelines propose names and definitions for a wide variety of floating features. Examples include <head> for titles and captions (not properly floating, since they are generally tied to a particular structural element); <q> for quoted matter and direct speech; <list> for lists and <item> for the items within them; <note> for footnotes etc.; <corr> for editorial corrections of the original source made by the encoder; and, optionally, a variety of lexically 'awkward' items such as <abbr> for abbreviations, <num> for numbers, <name> for names, <date> date for dates, <cit> for bibliographic or other citations, <address> for street addresses and <foreign> for words or phrases in a language that differs from the language of the text.

It goes without saying that this is only a very rough overview of the TEI encoding scheme. For more information about the TEI Guidelines, please go to the TEI website at [http://www.tei-c.org/P4X/].

3.4 Using the TEI scheme

It might be appropriate now to give some general advice on using the TEI encoding scheme.

It is usually advisable to identify the structure of the document first, then procede with the encoding of phrase and word level elements as required.

There are always decisions that have to be made about a text, best to think about this before start.

* what to include in the encoding and what to exclude
* what elements to use
* what attributes to use

Coding is always, to some extent, interpretation of a text.

* Be clear in you choices and consistent in carrying them out.
* type attribute values should be standardized.
* elements should be used consistently throughout a document
* id attributes are required to be unique, but make sure they are encoded systematically

Can is not the same as should

* All elements don't need to be used
* All attributes don't need to be used
* Make sure there are clear reasons behind all of your tagging choices

TEI is meant to describe, not display a text. Encoding should be focused on embedding metadata, so that the text can become more flexible. Focusing on the display of the text will produce a text that will allow usage for fewer purposes.

References

Carlson, G.,
“ Literary Works in Machine Readable Form ”, *Computers and the Humanities*, p.75-102

Coombs, James H.; Renear, Allen H.; DeRose, Steven J.,
“ Markup Systems and the Future of Scholarly Text Processing ”,
Communications of the ACM, p.933-947,
URL: `http://xml.coverpages.org/coombs.html`

Kay, G.,
“ Report on an Informal Meeting on Standard Formats for Machine-readable Text ”, *Litarary Data Processing Conference Proceedings*, p.327-328

Mylonas, Elli; Renear, Allen H.; DeRose, Steven J.; Durand, David G.,
“ What is text, really? ”, *Journal of Computing in Higher Education*, p.3-26

Sperberg-McQueen, Michael; Burnard, Lou (ed.),
Guidelines for Text Encoding and Interchange (TEI P3), ACH/ALLC/ACL Text Encoding Initiative: Chicago Oxford, 1994

Text Encoding Initiative, *TEI ED P1 Design Principles for Text Encoding Guidelines*, 1988

Text Encoding Initiative Consortium, *TEI @ Sourceforge*,
URL: `http://tei.sourceforge.net/`

Text Encoding Initiative Consortium, *TEI Lite*,
URL: `http://www.tei-c.org/Lite/`