Course Pack for Advanced Topics in TEI

June 13–17, 2016, University of Victoria

Syd Bauman, Northeastern University (s.bauman@neu.edu)
James Cummings, University of Oxford (james.cummings@it.ox.ac.uk)
Julia Flanders, Northeastern University (j.flanders@neu.edu)

This course pack contains the following materials:

1. Schedule
2. Encoder Predilection Profiling Tool
3. Document and Project Analysis Guide
4. Element Inventory Form (sample)
5. Levels of XML Rectitude chart
6. XPath axes diagram (aka “XPath Butterfly”)
7. Crib sheet for ODDs
8. Readings

All materials are also available at the workshop web site:

http://www.wwp.northeastern.edu/outreach/seminars/uvic_advanced_2016/index.html

ADVANCED TOPICS IN TEI
UNIVERSITY OF VICTORIA, JUNE 13–17, 2016

course instructors: Syd Bauman and James Cummings

course developed by: Julia Flanders, Syd Bauman, and James Cummings

Mon, 13 Jun

Session 0	**10:15–10:30**	Welcome logistics, introductions, gameplan
Session 1	**10:30–12:00**	Text encoding as data modeling markup as modeling; document analysis
Session 2	**13:30–16:00**	Modeling and the TEI class system exploring the TEI class system as a modeling tool group hands-on work; discussion

Tue, 14 Jun

Session 3	**09:00–12:00**	Using pointers to model data pointers in theory and practice hands-on work (including development of sample encoding)
Session 4	**13:30–16:00**	ODD: the TEI customization language basics of ODD-writing hands-on: develop test files and customization

Wed, 15 Jun

Session 5	**09:00–12:00**	Super-schema checking Review of ODD; What a grammar can [not] do XPath
Session 6	**13:30–16:00**	Rule-based schemas Schematron Hands-on: Schematron, schema design

Thu, 16 Jun

Session 7	**09:00–12:00**	ODDer Review of ODD; TEI Processing Model Individual hands-on practice and discussion ODD writing → schema generation → schema testing ⟲
Session 8	**13:30–16:00**	ODDest Individual hands-on practice and discussion ODD writing → schema generation → schema testing ⟲

Fri, 17 Jun

Session 9	**09:00–12:00**	Final hands-on; final discussion; wrap-up

Encoder Predilection Profiling Tool

1. Respond to each question below by circling one of the two responses (agree or disagree). Follow your gut; don't overthink it! Like the famous personality test this is modeled on, it will be somewhat irritating!

	Agree	Disagree
I like to know in advance what my document is going to look like when it's published	F	R
I know a lot about my documents before I start encoding them	E	N
Text encoding is a discovery process for me	E	N
The more encoding the better	P	G
I like to have a good schema to guide my encoding before I look at the documents	N	E
Exceptions are more interesting than rules	E	N
Vanilla TEI is too open-ended to be useful to me	N	E
Text encoding is like close reading	R	F
I'd rather have powerful analytic encoding than a perfect description of my documents	D	S
Each document challenges my ideas about my schema in exciting ways	R	F
Now that TCP exists, there's no point to encoding those texts again	G	P
Text encoding should be a faithful representation of the textual artifact	S	D
A truly successful encoding always eliminates information	D	S
If I had time, I would find it helpful to develop a really tight schema for my data	N	E
No matter how much I read, there will always be some new text with unfamiliar features	E	N
Simple languages like TEI Tite are better for collaboration and interchange	F	P
I always know how I'm going to use each piece of markup in my output	F	R
Whatever my schema says, my stylesheets ultimately determine the shape of my data	F	R
My output needs are too open-ended to predict	R	F
I think people who encode at the word level need to get a life	G	P
I don't see the point of text encoding if you don't take it seriously as a form of research	P	G
My encoding is an important expression of my scholarship	P	G
I see text encoding as a means to an end	F	R
If my colleagues understood the TEI, they would understand my research better	R	F
Encoding an entire text by hand fills me with ennui—I wish it could be automated.	G	P
Encoding a text really expands my understanding of it	R	F
When my document doesn't match my schema, I always change the schema	E	N
I would do anything to avoid changing my schema.	N	E
I feel most at home with metadata	D	S
Other people could learn a lot about me by examining my schema	R	F
Every time I start a new project, I feel like a learn a whole new perspective on the TEI	P	G
I know it's necessary, but mixing elements and text feels weird to me	D	S
Constraining attribute values is really important to me	D	S
My job as an encoder is to capture the words on the page and what they looked like	S	D

It's not data unless it's consistent	N	E
TEI Lite could probably meet 90% of my encoding needs	G	P
Encoding tends to erase what's distinctive about my documents	S	D
Ultimately, the encoded text is just an impoverished surrogate for the original	S	D
It's important to me that my encoding represent a theory of the text	P	G
My documents are so idiosyncratic that it's almost impossible to generalize about them	S	D
The only person who will really need my schema is my programmer	F	R
The whole point of encoding is to let the document teach us about itself	E	N
Beyond the paragraph level, text encoding is unsustainably expensive	G	P
The TEI needs to keep evolving as projects learn more and more about their data	P	G
Personographies and other highly structured data are the most powerful part of the TEI	D	S
I think <fw> provides more interesting data than <pb>	S	D
Schemas are an important part of my production work flow	F	R
I think schemas are an important way of regulating data	N	E

2. Total up your responses here:

Total D responses: _____

Total S responses: _____

Total E responses: _____

Total N responses: _____

Total P responses: _____

Total G responses: _____

Total F responses: _____

Total R responses: _____

3. Do a little math:

Your DS Score: D - S = _____. Divide the result by 12: _____ Multiply by 10: _____

Your EN Score: E - N = _____. Divide the result by 12: _____ Multiply by 10: _____

Your PG Score: P - G = _____. Divide the result by 12: _____ Multiply by 10: _____

Your FR Score: F - R = _____. Divide the result by 12: _____ Multiply by 10: _____

4. Interpreting the results:

DS score: Negative numbers locate you on the “S” end of this spectrum and indicate that you tend to be most interested in representing the details of the source document. Positive numbers locate you on the “D” end and indicate that you tend to be more oriented towards extracting data from the document, or representing it in a data-like, highly structured way.

S: Source-oriented  D: Data-driven

EN score: Negative numbers locate you on the “N” end of this spectrum and indicate that you tend to be more interested in normative, top-down, schema-driven approaches to modeling. Positive numbers locate you on the “E” end and indicate that you tend to be more interested in descriptive, bottom-up, document-driven approaches to modeling.

N: Normative  E: Descriptive

PG score: Negative numbers locate you on the “G” end of this spectrum and indicate that you tend to be more interested in general-purpose forms of modeling with the goal of maximizing reuse. Positive numbers locate you on the “P” end and indicate you tend to be more interested in project-specific forms of modeling that are oriented more towards specific anticipated forms of usage.

G: General-purpose  P: Project-specific

FR score: Negative numbers locate you on the “R” end of this spectrum and indicate that you tend to be more interested in exploratory, research-driven forms of modeling**.** Positive numbers locate you on the “F” end and indicate that you tend to be more interested in function-driven forms of modeling that have a specific tool set or functional outcome in mind

R: Research-driven  F: Function-driven

DEPF Data-driven, descriptive, project-specific, function-driven	**DEGF** Data-driven, descriptive, general-purpose, function-driven	**DNPF** Data-driven, normative, project-specific, function-driven	**DNGF** Data-driven, normative, general-purpose function-driven
DEPR Data-driven descriptive, project-specific, research-driven	**DEGR** Data-driven descriptive, general-purpose, research-driven	**DNPR** Data-driven, normative, project-specific, research-driven	**DNGR** Data-driven, normative, general-purpose, research-driven
SEPF Source-oriented descriptive, project-specific, function-driven	**SEGF** Source-oriented descriptive, general-purpose, function-driven	**SNPF** Source-oriented, normative, project-specific, function-driven	**SNGF** Source-oriented, normative, general-purpose, function-driven
SEPR Source-oriented descriptive, project-specific, research-driven	**SEGR** Source-oriented descriptive, general-purpose, research-driven	**SNPR** Source-oriented, normative, project-specific, research-driven	**SNGR** Source-oriented, normative, general-purpose, research-driven

Document and Project Analysis for Schema Design

This worksheet is intended to guide you through some of the analytic process needed for an effective text encoding plan (and by extension effective schema design). The questions below can be applied both to an individual document, in preparation for encoding it, and also to a collection of documents, in preparation for designing an encoding plan for the entire collection.

There are many different ways to work through this analysis. It can serve simply as a thought experiment or a way of gaining an understanding of what would be involved in planning your data modeling work in detail. More practically, if you are actually working through the questions below, you may find it helpful to create a formal document recording the information you generate. Experimentation will help you learn what works best for you, but here are some suggestions:

- In a table or spreadsheet, create an inventory of document features as you respond to the questions in Section 1: Document Analysis.
- For each feature you've listed, list the elements and attributes needed to represent that feature. (For instance, if you list bibliographic citations as a feature, then you might next list the specific bibliographic elements needed for your representation.) You should also include in this list any new elements you will need to create.
- In a separate table, create an inventory of elements and attributes. As you go through the questions in Section 2: Project Analysis, you can refine this inventory based on a clearer sense of project needs (perhaps demoting or removing some features and their elements). In Section 3: Schema Design, this inventory can then be used to guide the creation of a test document and a test TEI customization.
- For each element and attribute in the list, include a brief description of how it will be used (what specific features it will be used to encode, how they are to be recognized). These notes can later serve as the basis for your encoding documentation.
- You may find it helpful to add a column in which you identify the TEI module for each element (which will enable you to quickly identify the set of TEI modules you'll need to include in your schema) and the attribute class for each attribute.
- You could also include a column indicating what special customization is needed for specific elements (for instance, an added attribute or a change of model class).
- As you develop your schema, you can use this document as a checklist, checking off the elements you've taken care of and keeping track of what remains to be done (and keeping notes on any remaining problems or loose ends).

A sample form is included at the end of this document.

1. Document Analysis

Overall structure and genre

[These questions focus on how your markup will represent the overall structural architecture of the document.]

What is the overall structure of the document? Does it consist of a single textual object or an aggregation (e.g. a complete works, a multivolume document, etc.)? Does it have front matter or back matter? What textual unit will be represented in your encoding as an individual <TEI> structure?

Will you be capturing a documentary version of the text (using <sourceDoc>) or a facsimile (using <facsimile>? (The analysis below focuses on a conventional <text> but if you are using one of these additional representations you can extend the analysis to include them.)

What are the major structural components of the document? If you were creating a high-level outline of the document, would it contain subdivisions (sub-subdivisions, etc.)? If so, what are these? (E.g. chapters, poems, acts and scenes, generic sections, entries, etc.)

What kind of classification of these structural components is appropriate for your project? (I.e. what are the values for @type on ?) Consider here whether a fine-grained or coarse-grained classification is most likely to be useful, given your audience and the use to which you'll be putting this information (analysis, navigation, formatting…?)

Structural details

[These questions focus on lower-level structural features of the text.]

Within the structure of the text, what features of the text will you need to represent in order to provide for display, searching, analysis, or other processing? (For instance, if you are encoding a diary and want to be able to index and sort the entries by date, you would need to represent those dates in a way that allows them to be reliably identified and processed, for instance as <date> with @when inside <dateline>.) Examples include:

- Structurally important dates (such as dates for log entries, letters, or articles)
- Structurally important names (such as bylines, signatures, or names on title pages)
- Structurally important places (such as place names in letter headings or title pages)
- Bibliographic information (e.g. associated with quotations)
- Material that is rhetorically distinct: quotations, dialogue, asides, editorial notes, speech bubbles or captions in figures

This is just a set of illustrations, not an exhaustive list. Look through your documents and consider what features are structurally important to the way they'll be presented and used. For each feature you identify, consider what level of consistency in the representation you'll need in order to do the anticipated processing or discovery. What components are essential? Where do you need controlled vocabularies? Are there features of your text that are very consistent in their structure, such that you might benefit from a schema constraint to require them? For instance, should a letter be required to always begin with a dateline and end with a signature? List the specific elements and attributes that would be required, and any constraints on the order of elements that you would need to impose or test for.

Transcription

[These questions focus on how you will use markup to represent significant aspects of the transcription process.]

What forms of editorial intervention into the text will you be making as part of the transcription? Will these interventions be represented in the markup or done silently? For instance:

- Regularization or modernization of spelling, typography, or punctuation
- Correction of typographical errors
- Supplying variant readings from multiple witnesses

If the text is not perfectly legible, how will you handle illegible or difficult-to-read passages? What criteria will you apply in determining whether to treat a passage as illegible or simply unclear? (I.e. what level of certainty do you need to have in order to propose a tentative reading?)

Do you need to differentiate between different levels of certainty in your transcription?

Do you need to assign responsibility for specific readings and transcriptional decisions?

Document appearance

[These questions focus on how you will represent the appearance of the source document and the details of its material properties.]

What aspects of the source document's appearance are significant for your project and need to be represented explicitly as part of the markup? For instance: italics, typeface, alignment and justification, type size (absolute or relative), ink color, location on the page (for features like notes and handwritten additions), ornamentation.

What material properties of the source document are significant for your project and need to be represented explicitly (e.g. in the <msDesc>)? When considering what needs to be represented, think carefully about what pieces of information will actually be used in retrieval or analysis, and think about what form you'll need it in, to support those activities.

To what extent can this information be represented using a formal system (such as CSS or rendition ladders)? Are there any aspects of the document's appearance that cannot be formally described but still need to be represented? Would a note on the text suffice for these?

Components not captured

[These questions focus on features that you are explicitly excluding from your representation of the document.]

Are there any components of the document that can be omitted from your transcription altogether, based on your intended usage of the document: for instance, front matter, advertisements, running heads, non-authorial sections, footnotes, etc.?

Will these components simply be silently omitted or do you need to account for their absence in some manner (e.g. with <gap> or in the document metadata)?

Annotation

[These questions focus on editorial commentary and annotation that are represented as part of the markup (not on user annotations that are stored separately from the document).]

What forms of annotation or commentary, if any, will be included in the TEI encoding (whether in the transcription itself or in a linked document)? For instance:

- Commentary on specific words or passages (such as glosses, explanatory notes)
- General notes on the text as a whole

- Biographical information about people mentioned in the text (such as might be represented in a personography)
- Interpretive keywords, qualitative analysis codes

Consider whether these different kinds of information need to be handled differently in your output (for instance, authorial footnotes distinguished from editorial footnotes; biographical information linked from personal names; interpretive keywords visible only as part of the search interface). What encoding mechanism makes sense for each one, given the way it will be used?

2. Project Analysis

[These questions are aimed both at helping you assess the value of each markup feature you are considering, and also at helping you strategize the markup process. In this section, when we talk about "encoding this feature" we mean both "transcribing the content of this feature" and also "adding the markup necessary to explicitly represent this feature." In many cases, the important question is not whether you will transcribe a given feature (such as personal names) but what level of markup detail you will use.]

Treating the features you've identified in the document analysis above as a kind of target, consider each feature in light of your project's functional goals:

- Will encoding this feature directly contribute to the functions you have identified as essential?
- Will encoding this feature contribute to functions that are highly valuable (or may possibly prove important in the future)?
- Will encoding this feature now (rather than later) save time or money, or enable you to take advantage of resources you have now and may not have later on?

Consider each feature in light of your project's financial resources:

- Will encoding this feature add very significantly to the effort required to capture the text? (Will it require any additional steps that take the encoder away from the text to do look-ups? Will it require extra layers of error checking or increase the overall likelihood of error?)

Consider each feature in light of your project's available staff and their expertise:

- Will it be possible for your project to identify and encode this feature adequately given the staff expertise you have available? Does this feature require specialized subject knowledge? (If so, would the encoding process require two separate passes through the text by two different people, or would it make sense for one person to do all of the encoding?)
- Will encoding this feature require additional training of your staff?
- Will encoding this feature adequately require information that must be looked up or researched (e.g. checking whether a quotation is exact or not; checking the correct regularization of old-style dates; adding biographical information about people named in the text)?

Consider each feature in light of your project's likely horizon of completion or activity:

- If your project has a short horizon of completion (e.g. imposed by a grant or personnel availability), can you identify a set of features that would make sense as a first encoding pass through the text, leaving more advanced features for a subsequent project phase? Features that "make sense" in this context might be features that would support a basic display of the text (to enable you to build a user base with a simple initial publication) or features that would enable

you to demonstrate proof of concept for a prototype analytical tool (for instance, basic personography data to support a network diagram of a correspondence network).

- If your project has a phased funding model (e.g. a student worker every summer), can you identify a sensible sequence of encoding activities that would focus on capturing additional specific features over time? What would be the highest-priority features to focus on? What groupings of features would make sense (from the perspective of both training and error-checking)?

3. Schema Planning

Features of the transcription

[These questions are aimed at helping you identify the features that will be included in your schema, and how they should be represented.]

For each of the features you identified in the document analysis above, if you haven't already identified the specific elements needed to produce an appropriate encoding, do so now. You may find it helpful to create a test document with sample encoding at this stage.

For each of these features, consider what specific TEI attributes will be required, and whether you will need to specify a controlled vocabulary or datatype, or alter an existing TEI controlled vocabulary or datatype for that attribute.

In any cases where you identified a specific configuration of elements, consult the TEI Guidelines (or your test document) to see whether this configuration can be encoded using the TEI in unmodified form, or whether a customization is needed to allow specific elements in specific places.

Features not present

[These questions are aimed at helping you eliminate unnecessary components from your schema]

What TEI elements will never be needed in your encoding? Are they in TEI modules (i.e. chapters of the Guidelines) that contain elements you do plan to use? Or can you eliminate the entire module that contains them?

What TEI attributes will never be needed in your encoding? Are there entire attribute classes that can be removed from the schema?

Sample Element Inventory Form

This form is just an example; you'll want to create a version of it in a spreadsheet so that you can expand as needed.

Element or attribute name	Intended usage	TEI module	Customization notes	Done?

The 4 "Levels" of XML Rectitude

Syd Bauman, *Balisage* 2010

The following table lists English "sentences" and "TEI" constructs that fail to meet the indicated level of rectitude.
For the moment, this is just a 2-minute thought experiment. More later, maybe.

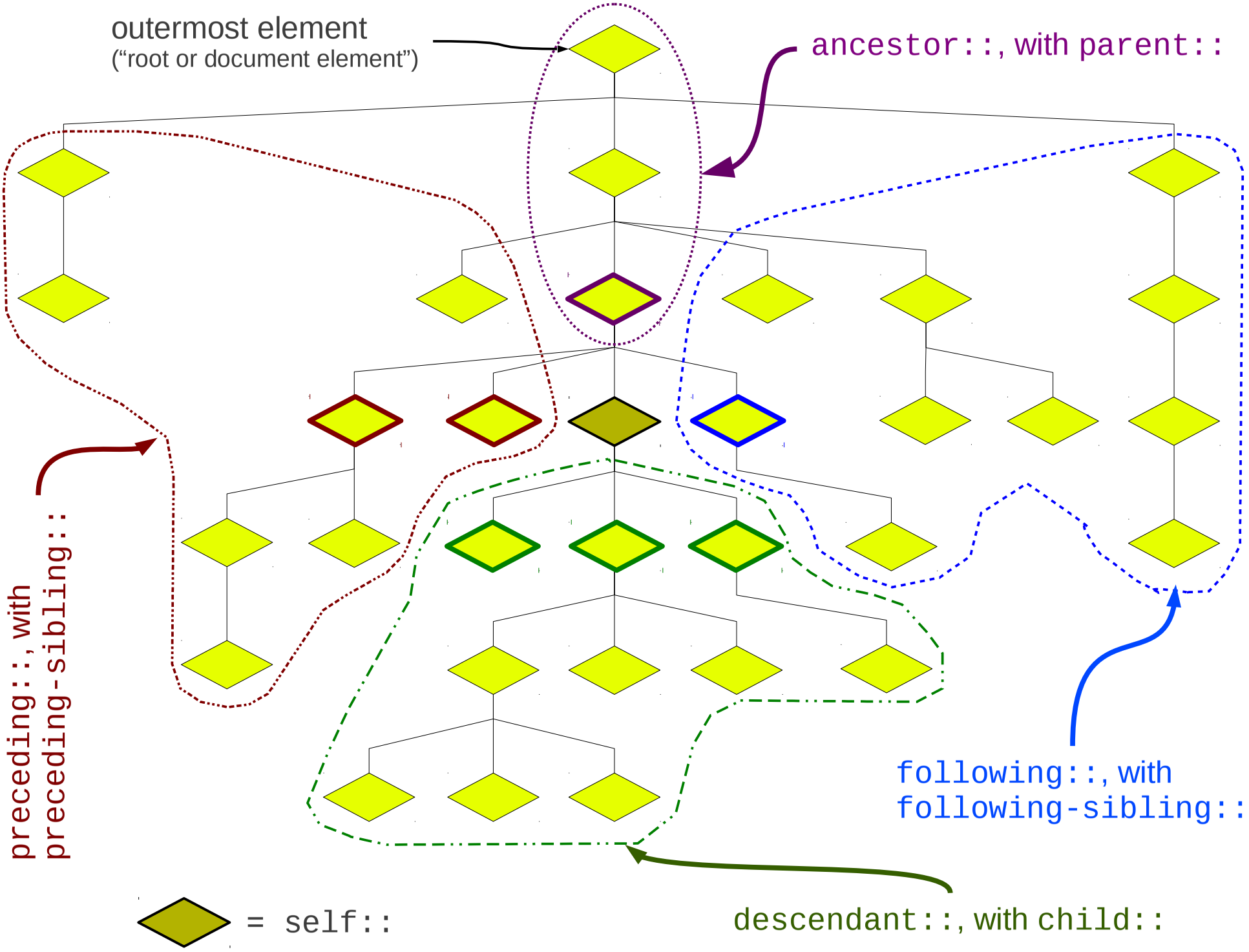
name	description	English	TEI

well-formedness	Standard rules of XML well-formedness. Sample failure includes mal-formed Unicode characters.	ǵ
		Fu cr n ee er g u ahr ruh ot nti otnn e ain ocie nlbry n eiae otepooiinta l e r rae qa.
			`<titleStmt>`
			`  <title>Fun!</head`
			`</titleDesc>`

validity	Does the syntax of the XML match the formal schema(s)?	I gave her a walked.
		
		The chewed rawhide puppy the.
			`<note href="#there">`
			`  <div>`
			`  What? In TEI, <gi>div</gi> is`
			`  not allowed in <gi>note</gi>.`
			`  </div>`
			`</note>`

sensibility	Does the construct make sense, e.g. as a proposition or assertion?	The virus sipped the tractor casually.
		
		Schematron rules cogitate purple.
			`<caesura xml:space="preserve"/>`
			
			`<respStmt>`
			`  <name><catchwords/></name>`
			`  <resp><height unit="cm"/></resp>`
			`</respStmt>`

veracity	Does the construct correctly represent reality?	The magnificent markup conference *Balisage* is held in Victoria, BC each year in June.	`<quote who="#Washington">`
			`  <time dur="PT4H7M">Four score and`
			`  seven years</time> ago our fathers`
			`  brought forth on this continent a`
			`  new nation …`
			`</quote>`


outermost element
(“root or document element”)
ancestor::, with parent::
preceding::, with
preceding-sibling::
following::, with
following-sibling::
= self::
descendant::, with child::

Element list for ODD

This is a brief reference sheet listing the most essential elements used in writing a TEI ODD file.

Ordered by Function

High-level ODD Structures

<schemaSpec>
: The element that contains the formal schema specifications within the ODD file. All the schema customization elements listed below are children of <schemaSpec>.

ident=
: Used on <schemaSpec> indicates the name of the schema that will be created; required by roma.

<moduleRef>
: A reference to a TEI module (i.e. a grouping of elements representing a single chapter of the TEI Guidelines), by which the module is included in the custom schema.

key=
: Used on <moduleRef>, gives the short name of the module to be included

except=
: Used on <moduleRef>, lists which elements from the module which should **not** be included

include=
: Used on <moduleRef>, lists which elements from the module should be included (thus excluding all others)

mode=
: Used on almost all ODD elements, indicates the nature of the change being made in relation to unmodified TEI:

  - "add" indicates that the feature in question originates with the customization and is **additional to** the unmodified TEI schema
  - "delete" indicates that the feature in question is being deleted from the custom schema
  - "change" indicates that the feature in question is being changed in some detail, and that the changed portion should

override the corresponding portion of the unmodified TEI schema, leaving other portions unchanged
- `"replace"` indicates that the feature in question is being completely changed, and that the portion designated in the custom schema should entirely replace the specification for that feature in the unmodified TEI

Element Management

<elementSpec>
A specification of a single element in the schema, which can be used to delete an element from an included module (with `mode="delete"`) or to modify some aspect of the element's definition (with `mode="change"` or `mode="replace"`).

module=
Used with <elementSpec>, indicates the module in which the element in question is defined.

ident=
Used with <elementSpec>, indicates the name of the element in question

<altIdent>
Used inside <elementSpec> to change the name of an element; indicates the new name by which the element being specified will be known in the custom schema

<classes>
Used within <elementSpec> (or <classSpec>) to change the class membership of the element (or class). Contains one or more <memberOf> elements which, by means of their key= attributes, indicate the specific classes to which the element (or class) is being added or from which it is being deleted.

<memberOf>
Designates a specific model class to which an element is being added, or from which it is being deleted.

key=
used on <memberOf>, contains the name of the class being designated.

mode=
used on <memberOf>, with value `"add"`, indicates that the element is being added to the class designated by the <memberOf> element.

<content>
Contains the content model for an element in RELAX NG (XML syntax) or Pure ODD elements, for which see “Content”, below

<constraintSpec>
Contains a further syntactic constraint, typically expressed in ISO Schematron.

Content

<elementRef>
A reference to an element (typically a TEI element, but perhaps one you've added in your customization), indicating that it is permitted or required at this point

key=
used on <elementRef>, indicates the element being refered to.

minOccurs=
indicates the smallest number of times the indicated element may occur at this point (default is 1).

maxOccurs=
indicates the largest number of times the indicated element may occur at this point (default is "unbounded", i.e. an infinite number of times).

<classRef>
A reference to an entire model class, all of whose members may (or must) occur at this point

key=
used on <classRef>, indicates the model class being referred to.

include=
used on <classRef>, indicates which members of the model class are being referred to

except=
used on <classRef>, indicates which members of the model class are not being referred to

<textNode>
Used to indicate that a string of zero or more characters is allowed at this point.

<sequence>
Indicates that each of the items referred to by the references inside must occur; if preserveOrder="true" then furthermore they must occur in the order specified

<alternate>
Indicates that only one of the items referred to by the references inside must occur (the number of times a selected alternate must occur is specified by minOccurs= and maxOccurs=

Attribute Management

<attList>
A list of attribute definitions; used to contain the <attDef> elements that are used to add, delete, or modify the attributes for an element.

<attDef>
Used to designate an attribute that is being added, deleted, or modified.

ident=
Used on <attDef>, indicates the name of the attribute in question.

<valList>
A list of values for an attribute (used to add or alter a set of constrained values).

<valItem>
A single value in a value list for an attribute.

<datatype>
Contains a reference to a TEI datatype, used when constraining the contents of an element or attribute value.

<constraintSpec>
Contains a further syntactic constraint, typically expressed in ISO Schematron.

Schema Documentation and Management

<gloss>
Used inside <elementSpec>, <attDef>, and <valItem> to provide the full natural-language equivalent of the name (i.e. ident=) of the construct. E.g., the <gloss> for <elementSpec ident="att"> is “attribute” (as opposed to “attention”)

<desc>
Used inside <elementSpec>, <attDef>, and <valItem> to document the new or altered element, attribute, or attribute value.

<exemplum>
Contains an example of usage of the attribute (in <attDef>), class (in

<classSpec>), element (in <elementSpec>), or macro (in <macroSpec>) along with optional commentary

<remarks>
Contains commentary or discussion about usage of the attribute (in <attDef>), class (in <classSpec>), element (in <elementSpec>), or macro (in <macroSpec>)

<classSpec>
A specification of a TEI class (either a model class or an attribute class); often used (with a mode= of "change") to change the membership of an attribute class

<macroSpec>
A specification of a TEI datatype or a chunk of arbitrary RELAX NG code

Processing Model Documentation

<model>
Documents potential processing for a specified element

behaviour=
Names the process or function which this processing model uses in order to produce output

predicate=
The XPath predicate expression giving the condition under which this model applies

useSourceRendition=
Whether to obey any rendition attribute which is present (by default no)

<outputRendition>
Description of the rendering of this element (in selected context)

<modelGrp>
A grouping of <model> elements with common output

<modelSequence>
A sequence of <model> elements intended as a single set of actions

Readings and Links

The following readings all offer different perspectives on the central question of this workshop: “How can we use TEI markup more precisely and powerfully to convey information and meaning about texts?” Each one is attacking a different specific challenge, and those challenges influence the way these authors frame the problem.

Bauman, Syd. “Freedom to Constrain: where does attribute constraint come from, mommy?” Presented at Balisage: The Markup Conference 2008, Montréal, Canada, August 12–15, 2008. In Bauman, Syd. “Freedom to Constrain: where does attribute constraint come from, mommy?” Presented at Balisage: The Markup Conference 2008, Montréal, Canada, August 12 - 15, 2Bauman, Syd. “Freedom to Constrain: where does attribute constraint come from, mommy?” Presented at Balisage: The Markup Conference 2008, Montréal, Canada, August 12–15, 2008. In *Proceedings of Balisage: The Markup Conference 2008*. Balisage Series on Markup Technologies, vol. 1 (2008). `doi:10.4242/BalisageVol1.Bauman01`.

This article offers a thoughtful examination of how different levels of information within a TEI system are or could be constrained.

Dodds, Leigh. “Schematron: validating XML using XSLT.” Presented at XSLT UK 2001, Oxford, UK, April 8–9 2001.
`http://www.ldodds.com/papers/schematron_xsltuk.html`

This paper discusses Schematron (in its early form) in some detail. I have extracted the overview of how it works here. The rest of the paper dives into implementation details.

Flanders, Julia. “Challenges of Collaborative Standards for Digital Humanities.” In *Collaborative Research in the Digital Humanities,* ed. Marilyn Deegan and Willard McCarty. Ashgate Publishing, 2012.

This article considers the TEI customization mechanism as a way of modeling the relationship between individual usage and community standards.

Sperberg-McQueen, C. M., David Dubin, Claus Huitfeldt, and Allen Renear. “Drawing Inferences on the Basis of Markup.” B. Tommie Usdin and Steven R. Newcomb (eds.) *Proceedings of Extreme Markup Languages 2002: Montreal, Canada*. 2002.
`http://conferences.idealliance.org/extreme/html/2002/CMSMcQ01/EML2002CMSMcQ01.html`

This somewhat technical article explores the ways in which markup can be understood as a carrier of formal information, and illustrates how we might approach the problem of meaning and interpretation as an information processing problem.

Pichler, Alois. "Transcriptions, Texts, and Interpretation." In: *Culture and Value*. Beiträge des 18 Internationalen Wittgenstein Symposiums. 13–20 August 1995. Kirchberg am Wechsel. Ed. Kjell S. Johannessen and Tore Nordenstam. Pp. 690–695.

This article offers a detailed look (from the perspective of a project focusing on philosophical manuscripts) at how the work of transcription and the work of interpretation intersect, and (by extension) how different layers of interpretive work can be represented in markup.

Piez, Wendell. "Beyond the "Descriptive vs. Procedural" Distinction. *Markup Languages: Theory and Practice* 3.2 (April 2001): 141–172.

This article examines the role played by constraint systems (schemas, work flows, expectations about semantics) in determining the function of markup as an information-bearing system.

Renear, Allen, and Jerome McGann. "What is text? A debate on the philosophical and epistemological nature of text in the light of humanities computing research." ACH/ALLC Conference, University of Virginia, June 1999.

This short piece serves as a very sketchy framing of what has proved to be a long-standing debate about the nature of text (and hence about the role markup might legitimately play in representing text).

Burnard, Lou and Syd Bauman, eds. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. 3.0.0. 2016-03. TEI Consortium. `http://www.tei-c.org/release/doc/tei-p5-doc/en/html/index-toc.html`.

Anyone planning to use the TEI seriously should read the *TEI Guidelines*. This workshop will be drawing on material from the following chapters; reviewing these would be good preparation, but no need to read them cover to cover (the are not included int his packet):

Chapter v: A Gentle Introduction to XML

Chapter 3: Elements Available in All TEI Documents

Chapter 4: Default Text Structure

Chapter 7: Performance Texts

Chapter 11: Representation of Primary Sources

Chapter 13: Names, Dates, People, and Places

Chapter 16: Linking, Segmentation, and Alignment

Chapter 20: Non-hierarchical Structures

Chapter 22: Documentation Elements

Balisage: The Markup Conference 2008
Proceedings

Freedom to Constrain

where does attribute constraint come from, mommy?

Syd Bauman
Senior Programmer/Analyst
Brown University Women Writers Project
<Syd_Bauman@Brown.edu>

Balisage: The Markup Conference 2008
August 12 - 15, 2008

How to cite this paper

Bauman, Syd. “Freedom to Constrain: where does attribute constraint come from, mommy?” Presented at Balisage: The Markup Conference 2008, Montréal, Canada, August 12 - 15, 2008. In *Proceedings of Balisage: The Markup Conference 2008*. Balisage Series on Markup Technologies, vol. 1 (2008). doi:10.4242/BalisageVol1.Bauman01.

Abstract

Where should attribute constraints live? In an external schema? In the document’s own metadata? In a separate file? Several possibilities are examined, raising lots of questions and offering a few answers.

Table of Contents

Use Case
Background
    Open vs Closed vs Extensible Schemas
    Literate Encoding
In the Closed Schema (RELAX NG file)
    how
    advantages
    disadvantages
In the Open Schema (ISO Schematron)
In the Metaschema (ODD file)
    how
    advantages

disadvantages
In the Metadata (<teiHeader>)
how — pointing
advantages
disadvantages
how — co-reference
In the Metadata (separate file)
Appendix A. <codeGrp> to Schematron

It is clear that constraining document structure is a very important part of document production. We test whether or not our XML documents are properly constrained through the process of validation. “The … purpose of validation is to subject a document … to a test, to determine whether it conforms to a given set of external criteria. … Our need to test is simply explained and understood (so much so that it rarely needs to be explicated): if there exists a point in a process where it is less expensive to discover and correct problems than it is to save the work of testing and fix at later points, it is profitable to introduce a test.”[1]

Michael Sperberg-McQueen may have summed this importance up best when he advised “constrain your data early and often”, which he often did.[2] (It helped that he lived in Chicago at the time.)

So it is obvious that constraints need to be expressed in a formal language of some sort. Many such general-purpose formal languages are available, including closed schema languages like DTDs and RELAX NG, and open schema languages like Schematron and CLiX. Furthermore at least one literate encoding language exists in which such constraints along with documentation about them can be expressed. This language is called ODD (for “one document does it all”) — constraints expressed in other languages (DTDs, RELAX NG, or XML Schema; in theory others as well) can be derived from a set of constraints expressed in ODD.[3][4][5][6] Furthermore there are systems of constraint based on special-purpose languages, rather than general-purpose languages. The feature system declaration created by the Text Encoding Initiative (TEI) and now being incorporated into ISO 24610-2 is an example — a set of XML elements (the feature system declaration) that can be used to constrain the expression of another set of XML elements (the feature structure itself).[7]

So the choice of *how* to express a particular constraint is not always obvious. But a related question is perhaps just as important: *where* should these constraints be expressed? What are the consequences of expressing them in different places?

This paper will attempt to shed light on these general questions by taking an in-depth look at the possible locations for the expression of one particular kind of constraint, and the consequences of those different locations. The constraint discussed will be that of limiting the value an attribute may take to one of an enumerated list of possible values. For simplicity the presumed setting for this constraint will be in a TEI document, but the principles should be equally applicable to any other encoding language that separates the document from its metadata, including DocBook or XHTML. The locations considered will be

- the “normal” way, in the formal closed schema (RELAX NG will be used as the example)
- in a formal open schema (ISO Schematron will be used as the example)
- in the metadata element (i.e. `<teiHeader>`)
- in a separate metadata file
- in the metaschema file (i.e. the ODD file)
- no formal constraint

Each of the latter methods will be compared to and contrasted with the first.

Use Case

There are lots of reasons to wish to constrain markup constructs, in particular attribute values. One case worth considering is the markup project which has tens or hundreds of occurrences of a particular attribute in each of tens or hundreds of files, where the list of possible values for the attribute is different for each file.

Imagine, e.g., an epigraphy project transcribing thousands of inscriptions on various objects. Imagine further that the inscriptions are divided among 27 separate files, organized by some criteria other than the kind of object that bears the inscription (e.g. date the object was discovered, current museum in which it is held, whatever). That which the text bearing object is made of is recorded in a TEI manuscript description on the `material=` attribute of the `<supportDesc>` element. Possible values might include `"bronze"`, `"marble"`, `"limestone"`, `"plaster"`, `"wood"`, etc.

Such a typical humanities computing project is likely to have:

- a subject matter expert
- an XML expert
- encoders — getting the extant text into XML-encoded digital form may

be accomplished in a variety of ways:

- typed from source
- post-OCR editing
- via an external vendor

- proofreaders, managers, web designers, research assistants, etc.

Background

Open vs Closed vs Extensible Schemas

Formal schema languages can generally be categorized into one of two types: open or closed. A closed schema language like RELAX NG specifies a complete document grammar. Only those documents that meet all of the constraints of the grammar are considered valid; all others are rejected as invalid.

An open schema language, like Schematron, specifies particular rules. Documents that violate the specified rules are rejected as invalid; all others are accepted as valid.

One can think of closed schema languages as a white list spam filter, and closed schema languages as a black list spam filter. Using a white list (closed schema language) only e-mail from the addresses specified get through, all others are rejected as spam. Using a black list (open schema language) any e-mail that is on the list of problematic addresses is rejected as spam, all others are allowed through.

Of course the situation is not as simple as that. One can specify some open constructs in many closed schema languages, and one can write sufficiently tight rules in most open languages that they behave like a closed language.

For example, validation against the following complete RELAX NG grammar will permit any XML document as long as it has a `<foo>` element with a `bar=` attribute as the first child of the root element.

```
start = element * { any_attribute*, foo, any_element* }
any_attribute = attribute * { text }
any_element = element * { any* }
any = ( any_attribute | any_element | text )
any_sans_bar = ( attribute * - ( bar ) { text } | any_element | text )
foo = element foo { attribute bar { text }, any_sans_bar* }
```

Conversely, validation against the following Schematron rule will permit only those documents that have one `<platypus>` element with a `bill=` attribute that has the value `"duck"` as the only child of the root `<enigma>` element.

```
<pattern>
  <rule context="/*">
    <assert test="name(.)='enigma'">Root element must be "enigma"</assert>
    <report test="@*">Root "enigma" element can not have attributes</report>
    <assert test="count(child::*)=1">"enigma" can only have one child
    ("platypus")</assert>
    <assert test="count(child::platypus)=1">"enigma" can only have one
    "platypus" child</assert>
    <report test="child::text()[not(normalize-space(.)='')]">"enigma" is
    not allowed to have text, just "platypus"</report>
  </rule>
  <rule context="/enigma/platypus">
    <assert test="@*[name(.)='bill']">"platypus" must have a bill=
    attribute</assert>
    <report test="@*[not(name(.)='bill')]">"platypus" must not have any
    attributes other than bill=</report>
    <report test="child::*">"platypus" must be empty (i.e., can not have
    child elements)</report>
    <assert test="string-length( normalize-space(.) ) = 0">"platypus"
    must be empty (i.e., can not contain text)</assert>
  </rule>
  <rule context="/enigma/platypus/@bill">
    <assert test="normalize-space(.)='duck'">The value of bill= of
    "platypus" must be 'duck'</assert>
  </rule>
</pattern>
```

These reverse uses of open and closed schema languages may be thought of as analogous to black-list or white-list spam filters that permit wildcards.

Neither of the above examples are particularly good ways of performing the desired validation, but they serve as proofs-of-concept that when we refer to a schema language as “open” or “closed”, we may be referring to its default, and not its only, behavior.

There is one further twist worth mentioning. Some modular XML document systems, including DocBook and TEI, permit a user of the system to generate (closed) schemas that contain not only the element and attribute declarations native to the system, but also additional declarations for constructs added by the user.

Literate Encoding

Literate programming is a style of programming intended to make computer documentation better by, among other things, placing the documentation and source code in the same computer file. The TEI has applied this concept to the schemas used to validate documents to help ascertain whether or not they conform to the TEI Guidelines. The source code from which the schemas are generated and the prose documentation that make up the bulk of the TEI Guidelines are stored in one computer document.

In order to facilitate this, and in order to help make it easy to extract formal

schemas in any of a variety of popular languages, the formal constraints are (for the most part) expressed in the TEI language, rather than any particular schema language.

Thus the TEI Guidelines proper (some 32 chapters of prose documentation), formal schemas expressed in RELAX NG, the XML DTD language, or the W3C Schema language, and reference documentation for those schemas, are all extracted from the same single document. We say that this “one document does” it all, and thus it is referred to as an ODD document.

In the Closed Schema (RELAX NG file)

how

Many are probably quite familiar with the mechanism for constraining an enumerated attribute in a formal closed schema language. E.g., in RELAX NG (compact syntax), the possible values of the `type=` attribute (in this case, of the `<name>` element) could be constrained with a construct like

```
attribute type { "person" | "place" | "ship" | "sword" }
```

A variety of readily available off-the-shelf software will test whether or not a document is valid with respect to a RELAX NG schema.

advantages

This method is extremely common for a reason: it makes a lot of sense. In many, many cases XML document structure is already governed by an external closed schema. These external schemas, at least when written in one of the three major languages (DTD, RELAX NG, W3C XML Schema) are generally easy to read and process. They describe the constraint in a standard formal language that has wide software support, including open source validators.

These languages typically provide the capability to specify a variety of structural and content constraints on XML documents. In particular, they provide the capability needed here: to constrain the set of possible values of the `type=` attribute to one of a list of possibilities. [8]

disadvantages

In many cases, the person or persons who write and maintain the external schema is not the same as the person or persons who create the XML instances (or the programs that write the XML instances) that conform to it.

In these cases, those who create the instances often do not have either the necessary knowledge (e.g., knowing the schema language) or capability (e.g., having read-write access to the schema) to make changes to it.

Furthermore in many cases (whether the instance creator is the same as the schema maintainer or not), a single external schema governs the validity of dozens or even tens of thousands of XML instances. But the desired constraints on a particular attribute may be different in different instances. Typically in these cases the schema limits the attribute to one of a set that is the union of all possible values in all governed documents. Here adding the additional constraint of “only these values in *this* document” requires making a separate schema that is like the original in all respects except for the declaration of the `type=` attribute of `<name>`.

In the Open Schema (ISO Schematron)

Many are probably quite familiar with the mechanism for constraining an enumerated attribute in a formal open schema language. E.g., in Schematron (DSDL part 4), the possible values of the `type=` attribute of the TEI `<name>` element could be constrained with a construct like

```
<pattern>
  <rule context="tei:name/@type">
    <assert test="normalize-space(.)='person'
                or normalize-space(.)='place'
                or normalize-space(.)='ship'
                or normalize-space(.)='sword'">
      Names can only be of people, places, ships, or swords
    </assert>
  </rule>
</pattern>
```

While the use of open vs closed schemas have a lot of advantages and disadvantages to the schema designer, with respect to this particular question, the advantages and disadvantages are primarily the same: while the constraint can be expressed in a formal, widely supported language, and can be tested with readily available tools, it is still in a separate file that may support many documents, that may not be accessible, and that uses a language that may be foreign to those who would like to change it.

There is one additional disadvantage of Schematron in particular with respect to RELAX NG: it is harder to annotate the Schematron schema than the RELAX NG schema. RELAX NG deliberately permits elements from other namespaces to be mixed in with the RELAX NG specifications, and defines where annotations relating to particular structures should go. Furthermore, because the four tokens against which we are trying to validate are expressed

as four separate elements (in the XML syntax), there is a place to annotate each separately (the <a:documentation> element follows the <rng:value> element to which it refers). Schematron also has a built-in documentation feature (a <p> element), but because all four tokens are tucked into a single XPath expression, it is a bit harder to discuss them individually. This is partially confounded because <p> is not permitted in <rule>, <assert>, or <report>, making it difficult to put the documentation close to the code. This is partially alleviated because elements from foreign namespaces are permitted in those spaces, and inside <p>. Thus something like the following construct could be used to provide documentation of such a constraint.

```
<pattern>
  <p class="annotation">The various values for <tei:att>type</tei:att> of
    <tei:gi>name</tei:gi> came about as follows: <tei:list type="gloss">
      <tei:label>
        <tei:val>person</tei:val>
      </tei:label>
      <tei:item>Added 2007-04-17 when we removed <tei:gi>persName</tei:gi></tei:item>
      <tei:label>
        <tei:val>place</tei:val>
      </tei:label>
      <tei:item>Added 2007-04-17 when we removed <tei:gi>placeName</tei:gi></tei:item>
      <tei:label>
        <tei:val>ship</tei:val>
      </tei:label>
      <tei:item>Added 2007-04-17 in order to accommodate the various ship names</tei:item>
      <tei:label>
        <tei:val>ship</tei:val>
      </tei:label>
      <tei:item>Added 2007-10-02 when we found a reference to "Excalibur" that the
        professor needed to annotate</tei:item>
    </tei:list>
  </p>
  <rule context="tei:name/@type">
    <tei:note><tei:att>type</tei:att> of <tei:gi>rs</tei:gi> is matched elsewhere.</tei:note>
    <assert test=".='person' or .='place' or .='ship' or .='sword'"> Names may only be
      of people, places, ships, or swords </assert>
  </rule>
</pattern>
```

In the Metaschema (ODD file)

how

The same constraint might be expressed, at a slightly higher level of abstraction and combined with some documentation, using the ODD literate encoding language:

```
<attDef ident="type">
  <valList type="closed">
    <valItem ident="person">
      <desc>The name refers to a person</desc>
    </valItem>
    <valItem ident="place">
```

```
      <desc>The name refers to a political or man-made region, for example
        a city, country, hamlet, town, or neighborhood. For geographical
        places such as rivers or valleys, use <gi>geogName</gi></desc>
    </valItem>
    <valItem ident="ship">
      <desc>The name refers to a ship, whether sea-worthy, interplanetary,
        or interstellar</desc>
    </valItem>
    <valItem ident="sword">
      <desc>The name refers to a sword</desc>
    </valItem>
  </valList>
</attDef>
```

There exists software that will “tangle” ODD specifications like the above into formal declarations in one of several schema languages, including RELAX NG. Then any of the same variety of readily available off-the-shelf software could be used to test validity.

Furthermore, there exists software that will “weave” the same specification above into easily readable hyperlinked documentation.

advantages

The advantages of literate programming are well understood, and include more easily readable and understandable source code, and that documentation (because it is right next to the source code) is more likely to match the program and be updated when the source code changes.[9] These advantages apply here as well. In addition, at least for those familiar with TEI, there is the advantage that the language used to describe the constraints is a TEI language, so schema designers are likely to be familiar with at least the documentation paradigm for the specialized schema-description elements, if not the elements themselves; in addition, they are likely familiar with the generic TEI elements (like `<desc>`, above) that are used in addition to the specialized elements.

disadvantages

The disadvantages of the external schema (whether open or closed) are present here as well. Furthermore, an extra processing step is required to generate (i.e. “tangle”) a schema that itself can be used to validate instances using off-the-shelf software. In addition, at least for those who are not intimately familiar with TEI, there is the disadvantage that the language used to describe the constraints is primarily a TEI language, so schema designers may not be familiar with the specialized schema-description elements.

In the Metadata (`<teiHeader>`)

how — pointing

It should be quite feasible to develop a mechanism for expressing the list of possible values of an attribute in the same document in a rather abstract way. For example:

```
<codeGrp elementTypes="name rs" attributes="type">
  <codeDef xml:id="person">The name or string refers to a
    person</codeDef>
  <codeDef xml:id="place">The name or string refers to a
    political or man-made region, for example a city, country,
    hamlet, town, or neighborhood. For geographical places such as
    rivers or valleys, use <gi>geogName</gi></codeDef>
  <codeDef xml:id="ship">The name or string refers to a ship,
    whether sea-worthy, interplanetary, or
    interstellar</codeDef>
  <codeDef xml:id="sword">The name or string refers to a
    sword, <foreign xml:lang="fr">main-gauche</foreign>, switchblade,
    or other edged weapon</codeDef>
</codeGrp>
```

Given this encoding in the `<teiHeader>`, the `<name>` element could have `type=` values of `"#person"`, `"#place"`, etc. Software could be developed to validate that the value of `type=` of `<name>` is a URI that points to an element whose parent `<codeGrp>` has `"name"` in its `elementTypes=` list and `"type"` in its `attributes=` list. (I believe that Schematron code could probably be used for this test, but have not yet demonstrated this.) Note that the check does not specify the element type of the child of `<codeGrp>`. This gives the flexibility to have special-purpose `<codeDef>`-like elements that might provide structured information about the value. E.g., one can well imagine the TEI's `<handNote>` element being used in this way.

advantages

This mechanism has significant potential advantages, particularly in cases where one schema is used for many files which may have different attribute constraint requirements. For most users it is much easier to change something in the same file they are working on, rather then needing to make changes to an external schema, particularly an external schema that may be in a language the user does not know or in a file to which the user does not have write access, and particularly changes that might inadvertently invalidate other existing instances. Thus the encoder, as opposed to the schema-designer, can add, remove, or change a value quite easily.

Another advantage is that the information about to what values the attribute is constrained, and what those values mean, is an integral part of the document. This means that this information will survive in the situation where a document instance is sent along without its schema or documentation.

Furthermore the list of values in different files at a given project could be slightly different.

Moreover, the particular system shown here has the advantage that it uses a mechanism most users are already familiar with: `xml:id=` and relative URIs (i.e., bare name fragment identifiers). It is worth noting, though, that there is no requirement that the URIs be bare name fragment identifiers, which permits this system to quickly and easily be changed to that which is discussed in section “In the Metadata (separate file)”.

disadvantages

This system has obvious inefficiencies when multiple, perhaps thousands, of document instances share the same constraints — the same information is repeated in each file.

Another significant disadvantage of this method is that we are using a non-standard language for constraint and documentation. The question, then, is whether or not this system is demonstrably significantly better than what can be obtained using standard languages.[10]

Lastly the fact that this system uses the URI pointing mechanism produces a disadvantages, one of which is severely problematic:

- of minor annoyance is that the user needs to encode a hash-mark (“`#`”, U+0023) at the beginning of each value;
- the fact that values are restricted to XML Names could be a problem in some situations;
- but far more problematic, because `xml:id=` needs to be unique within the document, any given possible attribute value can only occur on one attribute (although that attribute could be on multiple elements) — furthermore, no other element elsewhere in the document can use the same string as one of these attribute values as its identifier.

how — co-reference

Those last disadvantages that are the result of using `xml:id=` and URIs could be circumvented by matching the attribute values, rather than using a true pointer (e.g. ID/IDREF or URI). In the `<teiHeader>` the enumeration of the possible attribute values would look almost the same, but would use a different attribute for storing the actual value.

```
<codeGrp elementTypes="name rs" attributes="type">
  <codeDef attrVal="person">The name or string refers to a
    person</codeDef>
```

```
  <codeDef attrVal="place">The name or string refers to a
    political or man-made region, for example a city, country,
    hamlet, town, or neighborhood. For geographical places such as
    rivers or valleys, use <gi>geogName</gi></codeDef>
  <codeDef attrVal="ship">The name or string refers to a ship,
    whether sea-worthy, interplanetary, or
    interstellar</codeDef>
  <codeDef attrVal="sword">The name or string refers to a
    sword, <foreign xml:lang="fr">main-gauche</foreign>, switchblade,
    or other edged weapon</codeDef>
</codeGrp>
```

Software could be developed to validate that the value of `type=` of `<name>` is a string that matches the `attrVal=` attribute of an element whose parent `<codeGrp>` has `"name"` in its `elementTypes=` list and `"type"` in its `attribute=` list. (I believe that Schematron code could probably be used for this test, but have not yet demonstrated this. Certainly XSLT 1.0 can transform this into simple Schematron; this I have demonstrated, see Appendix A.) Note that the check does not specify the element type of the child of `<codeGrp>`. This gives the flexibility to have special-purpose `<codeDef>`-like elements that might provide structured information about the value. E.g., one can well imagine the TEI's `<handNote>` element being used in this way.

This system avoids the disadvantages of using `xml:id=`, and yet has several advantages over external schema files. E.g., encoders can quickly and easily add values to closed lists, in a manner that does not run the the risk that they might break the rest of the schema. I find the case of the encoder who wishes to quickly and easily express stricter constraints on her attribute values in a given file than those that come with the generic external schema very compelling.

In the Metadata (separate file)

In the method described in section "how — pointing" the values of the `type=` attribute of `<name>` are URIs. Because of this, it would be feasible to store the `<codeGrp>` element with `xml:id=` attributes in a project-wide "attribute_definitions.xml" file. While this has the advantage of flexibility and reusability, it presents the sizable disadvantage that the attribute values would now depend on details of system features external to the document. E.g., the ability to validate `<name type="../attribute_definitions.xml#sword">` breaks if the current file is moved to a sub-directory.

Furthermore, if the `<codeGrp>` is stored in a separate file, the maintenance issues are almost the same as those for a separate closed schema (e.g., a RELAX NG grammar), open schema (e.g., a Schematron schema), or

metaschema (e.g., a TEI ODD): those who have reason to change the constraints expressed may not have the write-permissions necessary to do so, and if they do may be at risk for invalidating files other than the one being worked on.

So in some cases (in particular, the scenario sketched out in section “Use Case”) it makes lots of sense to leave the formal constraints for some aspects of a document in the metadata section of that document itself, e.g. in the `<teiHeader>`. But having convinced ourselves there is a need to be able to express constraints in a different *place* than is usual, why require a separate formal construct to express the constraint? Why not include RELAX NG, Schematron, or ODD markup constructs in the `<teiHeader>` directly?[11] This is worthy of consideration, but is outside the scope of the current paper.

Appendix A. <codeGrp> to Schematron

The following XSLT 1.0 stylesheet is a proof-of-concept demonstration for transforming the `<codeGrp>` elements discussed above into Schematron that could be used to validate that an XML instance used only the mentioned possible values of the attribute specified.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Tranform my mythical <codeGrp> elements into a Schematron schema -->
<!-- Copyleft 2008 Syd Bauman -->
<!-- Last updated: 2008-08-31 -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron">

  <xsl:template match="/">
    <!-- only mess with <codeGrp> elements; if there are none, we do nothing -->
    <!-- Note that we presume each <codeGrp> has both elementTypes= and  -->
    <!-- attriubtes= specified and that their values are lists of one or more -->
    <!-- XML Names. No error-checking for this here, schema validation should -->
    <!-- have already flagged any that don't have both required attributes or -->
    <!-- have inappropriate values. -->
    <xsl:if test="//codeGrp">
      <!-- if there is one (or more) we write out a Schematron schema -->
      <sch:schema>
        <sch:ns uri="http://www.tei-c.org/ns/1.0" prefix="tei"/>
        <!-- and process each <codeGrp> into it -->
        <xsl:apply-templates select="//codeGrp"/>
      </sch:schema>
    </xsl:if>
  </xsl:template>

  <!-- Each <codeGrp> becomes a Schematron <pattern> -->
  <xsl:template match="codeGrp">
    <sch:pattern>
      <!-- append a blank to the GI list for easier parsing later -->
      <xsl:variable name="elementTypes" select="concat(normalize-space(@elementTypes),' ')"/>
      <!-- append a blank to the attribute name list for easier parsing later -->
      <xsl:variable name="attributes" select="concat(normalize-space(@attributes),' ')"/>
      <!-- Each GI/attribute pair becomes a Schematron <rule> -->
```

```
    <!-- A little more detail: each paired combination of -->
    <!-- 1. a GI listed on my elementTypes= attribute, and -->
    <!-- 2. an attribute name listed on my attributes= attribte -->
    <!-- becomes a <rule>. We do this by processing each GI in  -->
    <!-- a recursive template, which in turn calls another recursive -->
    <!-- template for the list of attributes. -->
    <xsl:call-template name="elementTypes">
      <xsl:with-param name="gis" select="$elementTypes"/>
      <xsl:with-param name="attrs" select="$attributes"/>
    </xsl:call-template>
  </sch:pattern>
</xsl:template>

<!-- Each GI listed on the elementTypes= attribute gets processed separately -->
<xsl:template name="elementTypes">
  <xsl:param name="gis"/>
  <xsl:param name="attrs"/>
  <!-- Taking advantage of that ending blank, parse off the 1st GI -->
  <xsl:variable name="this_gi" select="substring-before($gis,' ')"/>
  <xsl:variable name="rest" select="substring-after($gis,' ')"/>
  <!-- call attributes template to do the work for this particular GI -->
  <xsl:call-template name="attributes">
    <xsl:with-param name="gi" select="$this_gi"/>
    <xsl:with-param name="attrs" select="$attrs"/>
  </xsl:call-template>
  <!-- and do the same thing (via recursion) for the rest of the GIs, if any -->
  <xsl:if test="string-length($rest) > 1">
    <xsl:call-template name="elementTypes">
      <xsl:with-param name="gis" select="$rest"/>
      <xsl:with-param name="attrs" select="$attrs"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<!-- Each attibute name on the attributes= attribute gets processed in combination -->
<!-- with the current GI -->
<xsl:template name="attributes">
  <xsl:param name="gi"/>
  <xsl:param name="attrs"/>
  <!-- Taking advantage of that ending blank, parse off the 1st attribute -->
  <xsl:variable name="this_attr" select="substring-before($attrs,' ')"/>
  <xsl:variable name="rest" select="substring-after($attrs,' ')"/>
  <!-- make a rule out of it -->
  <xsl:element name="sch:rule">
    <xsl:attribute name="context">
      <!-- There must be a better way to do this ... -->
      <xsl:text>tei:</xsl:text>
      <xsl:value-of select="$gi"/>
      <xsl:text>/@</xsl:text>
      <xsl:value-of select="$this_attr"/>
    </xsl:attribute>
    <xsl:variable name="numVals" select="count(child::*/@attrVal)"/>
    <!-- if I have no children with attrVal= specified, then don't -->
    <!-- generate any assertions (luckily an emtpy <rule> is valid -->
    <!-- in Schematron). -->
    <xsl:if test="$numVals > 0">
      <xsl:element name="sch:assert">
        <!-- Probably would be better to generate this test (i.e., the expression -->
        <!-- that is the value of this output test= attribute) only once per attrVal=, -->
        <!-- rather once for each attrVal= for each GI/attr combination. -->
        <xsl:attribute name="test">
          <xsl:for-each select="child::*/@attrVal">
            <xsl:text>.='</xsl:text>
            <xsl:value-of select="."/>
            <xsl:text>'</xsl:text>
```

```
            <xsl:if test="$numVals > 1  and  position() != last()">
              <xsl:text> or </xsl:text>
            </xsl:if>
          </xsl:for-each>
        </xsl:attribute>
      </xsl:element>
    </xsl:if>
  </xsl:element>
  <!-- and do the same thing (via recursion) for the rest of the attributes, if any -->
  <xsl:if test="string-length($rest) > 1">
    <xsl:call-template name="attributes">
      <xsl:with-param name="gi" select="$gi"/>
      <xsl:with-param name="attrs" select="$rest"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

---

[1] Piez, Wendell, “Beyond the ‘descriptive vs. procedural’ distinction”, presented at Extreme Markup Languages 2001, Montréal, Canada. http://www.idealliance.org/papers/extreme/proceedings/html/2001/Piez01/EML2001Piez01.html.

[2] Sperberg-McQueen, C. Michael. Oral conversation, and multiple oral presentations throughout the 1990s. See, e.g., http://www.w3.org/People/cmsmcq/2001/darmstadt.html.

[3] Burnard, Lou and Syd Bauman, eds. “4.3.2 Floating Texts.” *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version 1.1.0. 2008-07-04. TEI Consortium. http://www.tei-c.org/release/doc/tei-p5-doc/html/DS.html#DSFLT 2008-08-30

[4] Burnard, Lou and Syd Bauman, eds. “23.4 Implementation of an ODD System.” *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version 1.1.0. 2008-07-04. TEI Consortium. http://www.tei-c.org/release/doc/tei-p5-doc/en/html/USE.html#IM 2008-08-30

[5] Sperberg-McQueen, C. Michael and Lou Burnard. “The Design of the TEI Encoding Scheme.” *Computers and the Humanities* 1995. 29 (1) p. 17–39.

[6] Burnard, Lou, Sebastian Rahtz. “RelaxNG with Son of ODD”, presented at Extreme Markup Languages 2004, Montréal, Canada. http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Burnard01/EML2004Burnard01.pdf.

[7] Burnard, Lou and Syd Bauman, eds. “18 Feature Structures” *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version 1.1.0. 2008-07-04. TEI Consortium. http://www.tei-c.org/release/doc/tei-p5-doc/en/html/FS.html 2008-08-30

[8] DTDs impose greater restrictions on what the members of that list can be

than the others: each possible value must be an XML Name.

[9] Knuth, Donald. *Literate Programming*, ISBN 0-9370-7380-6.

[10] What some call *Syd's rule*, and I have begun to call my *wheel re-invention prevention convention*: "unless your method is significantly and demonstrably superior to the standard, you should be using the standard.".

[11] Indeed, James Cummings and I have suggested this on more than one occasion. See, e.g., http://lists.village.virginia.edu/pipermail/tei-council/2005/005627.html.

Author's keywords for this paper: XML; attribute; TEI; ODD; constraint

Syd Bauman
<Syd_Bauman@Brown.edu>
Senior Programmer/Analyst
Brown University Women Writers Project

Syd Bauman is the technical person at the Brown University Women Writers Project, where he has worked since 1990, designing and maintaining a significantly extended TEI-conformant schema for encoding early printed books. He has served as the North American Editor of the Text Encoding Initiative Guidelines, has an AB from Brown University in political science, and has worked as an Emergency Medical Technician since 1983.

Balisage Series on Markup Technologies

Schematron Overview

excerpted from Leigh Dodds' 2001 *XSLT UK* paper, "Schematron: validating XML using XSLT"

Leigh Dodds, ingenta ltd, xmlhack.com

April 2001

Abstract

Schematron [Schematron] is a structural based validation language, defined by Rick Jelliffe, as an alternative to existing grammar based approaches. Tree patterns, defined as XPath expressions, are used to make assertions, and provide user-centred reports about XML documents. Expressing validation rules using patterns is often easier than defining the same rule using a content model. Tree patterns are collected together to form a Schematron schema.

Schematron is a useful and accessible supplement to other schema languages. The open-source XSLT implementation is based around a core framework which is open for extension and customisation.

Overview

This paper provides an introduction to Schematron; an innovative XML validation language developed by Rick Jelliffe. This innovation stems from selecting an alternative approach to validation than existing schema languages: Schematron uses a tree pattern based paradigm, rather than the regular grammars used in DTDs and XML schemas. As an extensible, easy to use, open source tool Schematron is an extremely useful addition to the XML developers toolkit.

The initial section of this paper conducts a brief overview of tree pattern validation, and some of the advantages it has in comparison to a regular grammar approach. This is followed by an outline of Schematron and the intended uses which have guided its design. The Schematron language is then discussed, covering all major elements in the language with examples of their

usage. A trivial XML vocabulary is introduced for the purposes of generating examples.

The later sections in this paper provides an overview of the open source XSLT framework used to implement the Schematron language. The Schematron conformance language for custom implementation is also introduced. The paper completes with some suggestions of possible future extensions.

Introducing Tree Patterns as a Validation Mechanism

During the last few years a number of different XML schema languages have appeared as suggested replacements for the ageing Document Type Definition (DTD). The majority of these have taken the basic premise of recasting DTD functionality in XML syntax with the addition, in some cases, of other features such as data typing, inheritance, etc [XMLSchema]. The use of XML syntax provides additional flexibility through leveraging existing tools for markup manipulation, while the 'value added' features satisfy the requirements of developers looking for closer integration with databases and object-oriented languages.

Yet the fundamental approach adopted by these languages does not diverge greatly from the DTD paradigm: the definition of schemas using regular grammars. Less formally, schemas are constructed by defining parent-child and sibling relationships [Jelliffe1999a]. For example in a DTD one might write:

```
<!ELEMENT wall EMPTY>
<!ELEMENT roof EMPTY>
<!ELEMENT house (wall+, roof)>
```

This defines three elements, `wall`, `root`, and `house`. The parent-child relationship between `house` and `wall` elements is defined in the content model for `house`. A house may have several walls.

The sibling relationship between `wall` and `roof` is derived from the same content model, which defines them as legal sibling children of the `house` element.

However this means that DTDs, and similar derivatives, are unable to define (and hence constrain) the other kinds of relationships that exist amongst markup elements within a document. As the XPath specification [XPath] shows, there are many possible kinds of relationship, known as 'axes'.

Example 1. Tree relationships (axes) defined by XPath

- child
- descendant
- parent
- ancestor
- following-sibling
- preceding-sibling

- following
- preceding
- attribute
- namespace
- self
- descendant-or-self
- ancestor-or-self

While XML does include an ID/IDREF mechanism which allows for cross-referencing between elements, and hence another form of relationships, it only weakly binds those elements. There is no enforcement that an IDREF must point to an ID on a particular element type, simply that is must point to an existing ID, and further that all IDs must be unique.

Having highlighted the fact that the existing schema paradigm can only express constraints among data items in terms of the child and sibling axes, it is natural to consider whether an alternate paradigm might allow a schema author to exploit these additional relationships to define additional types of constraint amongst document elements. Tree patterns do just that, and XPath provides a convenient syntax in which to express those patterns.

Validation using tree patterns is a two-step process:

- Firstly the candidate objects (in XPath terms, nodes) to be validated must be identified. i.e. identify a *context*
- Secondly assertions must be made about those objects to test whether they fulfill the required constraints.

Both the candidate object selection, and the assertions can be defined in terms of XPath expressions. More formally, the nodes and arcs within a graph of data can be traversed to both identify nodes, and then make assertions about the relationships of those nodes to others within the same graph. Assertions are therefore the mechanism for placing constraints on the relationships between nodes in a graph (elements and attributes in an XML document).

For example, we may select all `house` nodes within a document using the expression:

```
//house
```

And then assert that all `house`s have `wall`s by confirming that the following pattern selects one or more child nodes (within the context defined by the previous selection):

```
child::wall
```

Regular grammars, as used in DTDs, can then be viewed as tree patterns where the only available axis is the parent-child axis [Jelliffe1999e]. Full use of tree pattern validation provides the maximum amount of freedom when modelling constraints for a schema. This comes at very little cost: XPath is available in most

XML environments. For example the following types of constraint are hard, or impossible to express with other schema languages.

Example 2. Examples of 'difficult' constraints

- Where attribute X has a value, attribute Y is also required
- Where the parent of element A is element B, it must have an attribute Y, otherwise an attribute Z
- The value of element P must be either "foo", "bar" or "baz"

Tree patterns are the schema paradigm underpinning Schematron as a validation language.

There are reasons to believe that tree-pattern validation may be more suitable in an environment where documents are constructed from elements in several namespaces (often termed 'data islands'). As many consider that the future of XML document interchange on the Internet will involve significant mixing of vocabularies, a flexible approach may bring additional benefits.

Introducing Schematron

Background

Schematron [Schematron] is an XML schema language designed and implemented by Rick Jelliffe at the Academia Sinica Computing Centre, Taiwan. It combines powerful validation capabilities with a simple syntax and implementation framework. Schematron is open source, and is (at the time of writing) being migrated to SourceForge to better manage its development by a rapidly growing community of users.

Schematron traces its ancestry [Jelliffe1999f] indirectly from SGML DTDs via Assertion Grammars [Raggett], Groves and Property Sets [Arciniegas]. A recent review of six current schema languages [Lee] supports this view, declaring Schematron to be unique in both its approach and intent. Before discussing the details of the Schematron language it is worth reviewing the design goals which have been highlighted by its author.

Design Goals

There are several aims which Rick Jelliffe which believed were important during the design and specification of Schematron [Schematron], [Jelliffe2001]:

- Promote natural language descriptions of validation failures, i.e. diagnose as well as reject
- Reject the binary valid/invalid distinction which is inherent in other schema languages
- Aim for a short learning curve by layering on existing tools (XPath and XSLT)

- Trivial to implement on top of XSLT
- Provide an architecture which lends itself to GUI development environments
- Support workflow by providing a system which understands the phases through which a document passes in its lifecycle

Target Uses

Jelliffe has also suggested [Jelliffe2001] several target environments in which Schematron is intended to add value:

- Document validation in software engineering, through the provision of interlocking constraints
- Mining data graphs for academic research or information discovery. Constraints may be viewed as hypotheses which are tested against the available data
- Automatic creation of external markup through the detection of patterns in data, and generation of links
- Use as a schema language for "hard" markup languages such as RDF.
- Aid accessibility of documents, by allowing usage constraints to be applied to documents

How It Works

The implementation of Schematron derives from the observation that tree pattern based validators can be trivially constructed usings XSLT stylesheets [Jelliffe1999b], [Norton]. For example, a simple stylesheet that validates that `house`s must have `wall`s can be defined as follows:

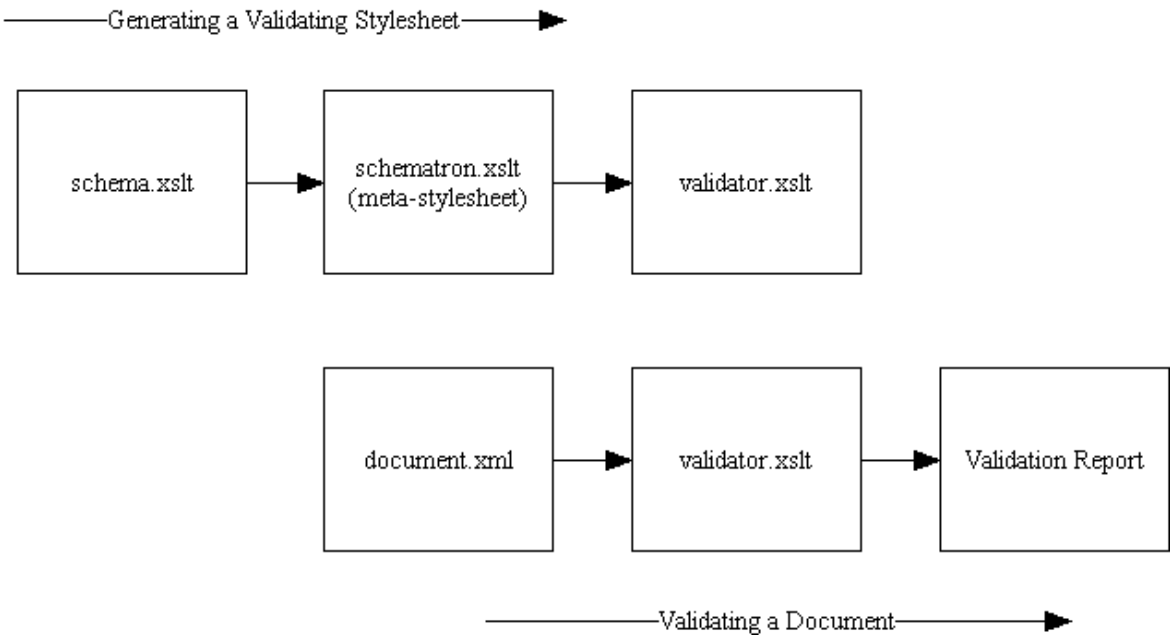
```
    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

      <!-- select all houses -->
      <xsl:template match="//house">
        <!-- test whether it has any walls -->
        <xsl:if test="not(wall)">
          This house has no walls!
        </xsl:if>
      </xsl:template>

  </xsl:stylesheet>
```

It should be obvious from the above that if a `house` does not have any `wall`s, a simple error message will be displayed to the user.

Schematron takes this a natural step further by defining a schema language which, when transformed through a meta-stylesheet (i.e. a stylesheet which generates other stylesheets), produces XSLT validators similar to the above. The following diagram summarises this process.

Figure 1. Generating Validators Using Schematron

Editor’note: most Schematron users would call the first box “schema.sch”.



Schematron is therefore a simple layer above XPath and XSLT allowing it to leverage existing tools, and benefit from a framework which is already familiar to XSLT developers. Yet from a user perspective, the details of XSLT are hidden; the end-user need only grapple with the XPath expressions used to define constraints.

The following section outlines the Schematron assertion language which is used to define Schematron schemas. The last section in the paper provides information on the Schematron implementation (i.e. the metastylesheet) which will be of interest to implementors seeking to customise Schematron for particular needs.

The Schematron Assertion Language

This section introduces the Schematron assertion language which can be used to generate XSLT validators using the Schematron implementation. All following examples conform to a simple XML vocabulary introduced in the next section.

DTD For Example Content

The examples used within this section will refer to a fictional XML language for describing building projects. While the examples could have been couched in terms of an existing schema language, the intention is to provide a simple vocabulary which does not assume any prior knowledge on behalf of the user. It should be stressed that, while the examples themselves may be trivial this should not be taken to indicate any specific limitation in Schematron, which is capable of handling much more complex schemas.

The following DTD defines the building project vocabulary:

Example 3. Simple XML language for illustrative purposes

```
<!ELEMENT wall EMPTY/>
 <!ELEMENT roof EMPTY/>
 <!ELEMENT street #PCDATA>
 <!ELEMENT town #PCDATA>
 <!ELEMENT postcode #PCDATA>
 <!ELEMENT firstname #PCDATA>
 <!ELEMENT lastname #PCDATA>
 <!ELEMENT certification EMPTY>
 <!ATTLIST certification number CDATA #REQUIRED>
 <!ELEMENT telephone #PCDATA>

 <!ELEMENT address (street, town, postcode)>
 <!ELEMENT builder (firstname, lastname, certification)>
 <!ELEMENT owner (firstname, lastname, telephone)>
 <!ELEMENT house (wall+, roof?, address, (builder|owner)?>
```

This schema allows us to describe a house consisting of a number of walls and a roof. The roof may not be present if the house is still under construction.

A house has an address which consists of a street name, town and a postcode.

A house should have either a builder who is currently assigned to its construction (and all builders must be certified), or an owner. Certification numbers of builders, and telephone numbers of owners are also recorded for adminstrative purposes.

A sample document instance conforming to this schema is:

Example 4. Sample document instance

```
<house>
  <wall/>
  <wall/>
  <wall/>
  <wall/>
  <address>
    <street>1 The High Street</street>
    <town>New Town</town>
    <postcode>NT1</postcode>
  </address>
  <builder>
    <firstname>Bob</firstname>
    <lastname>Builder</lastname>
    <certification number="123"/>
  </builder>
</house>
```

Core Elements: Assert and Report

The basic building blocks of the schematron language are the `assert` and `report`

elements. These define the constraints which collectively form the basis of a Schematron schema. Constraints are assertions (boolean tests) that are made about patterns in an XML document; these patterns and tests are defined using XPath expressions.

The best illustration is a simple example:

Example 5. A simple assertion

```
<assert test="count(walls) = 4">This house does not have four walls</assert>
```

This demonstrates a simple assertion which counts the number of `walls` in the current context. Recall that validation is a two step process of identification and followed by assertion. The identification step generates the context in which assertions are made. This is covered in the next section.

If there are not four walls then the assertion fails and a message, the content of the `assert` element, is displayed to the user.

Asserts therefore operate in the conventional way: if the assertion evaluates to false some action is taken. The `report` element works in the opposite manner. If the test in a report element evaluates to true then action is taken.

While reports and asserts are effectively the inverse of one another, the intended uses of the two elements are quite different. An assert is used to test whether a document conforms to a particular schema, generating actions if deviations are encountered. A report is used to highlight features of the underlying data:

Example 6. A simple report

```
<report test="not(roof)">This house does not have a roof</assert>
```

The distinction may seem subtle, especially when grapplying with a constraint which may be expressed simpler in one way or the other. However Schematron itself does not define the action which must be taken on a failed assert, or successful report, this is implementation specific. The default behaviour is to simply provide the user with the provided message. An implementation may choose to handle these two cases differently.

It is worth noting that there is a trade-off to be made when defining tests on these elements. In some cases a single complex XPath expression may accurately capture the desired constraint. Yet it is closer to the 'spirit' of Schematron's design to use several smaller tests that collectively describe the same constraint. Specific tests can more accurately provide feedback to a user, than a single general test and associated message.

Assert and Report elements may contain a `name` element which has an optional path attribute. This element will be substituted with the name of the current element before a message is passed to the user. When supplied the path attribute

should contain an XPath expression referencing an alternate element. This is useful for giving additional feedback to the user about the specific element that failing an assertion.

Schematron 1.5, released in January 2001, adds the ability to provide detailed diagnostic information to users. Assert and report messages should be simple declarative statements of what is, or should be. Diagnostics can include detailed information that can be provided to the user as appropriate to the Schematron implementation. Diagnostic information is grouped separately to constraints, and is cross-referenced from a diagnostic attribute.

Example 7. Diagnostic Example

```
<assert test="count(walls) = 4" diagnostics="1">This house does not have four walls</assert>

...

<diagnostics>
  <diagnostic id="1">
    Its an odd house which has more or less than four walls! Consult your architect...
  </diagnostic>

</diagnostics>
```

Writing Rules

As noted earlier, constraints must be applied within a context. The context for constraints is defined by grouping them together to form `rule`s.

Example 8. Defining the context for assertions

```
<rule context="house">
  <assert test="count(wall) = 4">A house should have four walls</assert>
  <report test="not(roof)">This house does not have a roof</assert>
</rule>
```

The context attribute for a `rule` contains an XPath expression. This identifies the candidate nodes to which constraints will be applied. The above example checks that a `house` contains 4 `wall` child elements, and provides feedback to the user if it is missing a `roof`.

Schematron 1.5 add a simple macro mechanism for rules which is useful when combining constraints. To do this, a rule may be declared as 'abstract'. The contents of this rule may be included by other rules as necessary. This is achieved through the use of the `extends` element.

Example 9. Using abstract rules and the extends element

```
<rule abstract="true" id="nameChecks">
   <assert test="firstname">A person must have a first name</assert>
```

```
   <assert test="lastname">A person must have a last name</assert>
</rule>

<rule context="builder">
   <extends rule="nameChecks"/>

   <!-- builder specific constraints -->
   ...
</rule>

<rule context="owner">
   <extends rule="nameChecks"/

   <!-- owner specific constraints -->
   ...
</rule>
```

In the above example an abstract rule is defined, and assigned the id "*nameChecks*". Two assertions are associated with this abstract rule: checking that an element has a firstname and a lastname. These assertions are imported by the other non-abstract rules and will be applied along with the other constraints specific to that element. An abstract rule may contain assert and report elements but it cannot have a context. Assertions from an abstract rule obtain their context from the importing rule.

Producing Patterns and Schemas

The next most important element in a Schematron schema is `pattern`. Patterns gather together a related set of rules. A particular schema may include several patterns that logically group the constraints.

Example 10. Grouping rules using patterns

```
<pattern name="Adminstration Checks"
         see="http://www.buildingprojects.org/admin/procedures.html">

   <rule abstract="true" id="nameChecks">
      <assert test="firstname">A person must have a first name</assert>
      <assert test="lastname">A person must have a last name</assert>
   </rule>

   <rule context="builder">
      <extends rules="nameChecks"/>

      <!-- builder specific constraints -->
      ...
   </rule>

   <!-- additional rules -->
   ...

</pattern>

<pattern name="Other Constraints">
   <!-- other rules -->
```

```
</pattern>
```

A pattern should have a name and may refer to additional documentation using a URL. A Schematron implementation can then furnish the user with a link to supporting documentation.

Patterns defined within a schema will be applied sequentially (in lexical order). Nodes in the input document are then matched against the contexts defined by the rules contained within each pattern. If a node is found to match the context of a particular rule, then the assertions which it contains will be applied. Within a pattern a given node can only be matched against a *single* rule. Rules within separate patterns may match the same node, but only the first match within a pattern will be applied. An example of an incorrect schema is given below.

Example 11. Incorrect use of rule contexts

```
<pattern name="Adminstration Checks">

   <!-- this rule WILL be matched -->
   <rule context="builder">
      <assert test="firstname">A person must have a first name</assert>
   </rule>

   <!-- this rule WILL NEVER be matched -->
   <rule context="builder">
      <assert test="lastname">A person must have a last name</assert>
   </rule>

   <!-- additional rules -->
   ...

</pattern>

<!-- rules in this pattern will be checked after the above -->
<pattern name="Other Constraints">

   <!-- this rule WILL be matched -->
   <rule context="builder">
      <assert test="certification">A builder must be certified</assert>
   </rule>

   <!-- additional rules -->
   ...

</pattern>
```

Care should be taken when defining contexts to ensure that these circumstances never arise.

The last step in defining a Schematron schema is to wrap everything up in a `schema` element.

Example 12. Grouping patterns to create a schema

```
<schema xmlns="http://www.ascc.net/xml/schematron">
   <title>A Schematron Schema for Validating Building Project Documents</title>

   <ns uri="http://www.buildingprojects.org/admin/" prefix="bld"/>

   <pattern name="Adminstration Checks">
   <!-- rules -->
   </pattern>

   <pattern name="Construction Checks">
   <!-- rules -->
   </pattern>

</schema>
```

There are several points to note about the above schema. Firstly it introduces the namespace for Schematron documents, which is "http://www.ascc.net /xml/schematron". Secondly a schema may have a title; this is recommended.

Conclusions

Schematron is unique amongst current schema languages in its divergence from the regular grammar paradigm, and its user-centric approach.

Schematron is not meant as a replacement for other schema languages; it is not expected to be easily mappable onto database schemas or programming language constructs. It is a simple, easy to learn language that can perform useful functions in addition to other tools in the XML developers toolkit.

It is also a tool with little overhead, both in terms of its learning curve and its requirements. XSLT engines are regular components in any XML application framework. Schematrons use of XPath and XSLT make it instantly familiar to XML developers.

A significant advantage of Schematron is the ability to quickly produce schemas that can be used to enforce house style rules and, more importantly, accessibility guidelines without alteration to the schema to which a document conforms. An XHTML document is still an XHTML document even if it does not meet the Web Accessibility Initiative Guidelines [WAI]. These kind of constraints describe a policy which is to be enforced on a document, and can thus be layered above other schema languages. Indeed in many cases it may be impossible for other languages to test these kinds of constraints.

Bibliography

[Arciniegas] Fabio Arciniegas. 19/04/2000. *Groves Explained* .

[Conformance] Rick Jelliffe. *Schematron Conformance Language* .

[Dodds] Leigh Dodds. 10/1/2001. *Old Ghosts: XML Namespaces* .

[Holman] Ken Holman. *Practical Transformations using XSLT and XPath* .

[Jelliffe1999a] Rick Jelliffe. 1999. *From Grammars To The Schematron* .

[Jelliffe1999b] Rick Jelliffe. 24/01/1999. *Using XSL as a Validation Language* .

[Jelliffe1999c] Rick Jelliffe. 11/06/1999. *How to Promote Organic Plurality on the WWW* .

[Jelliffe1999d] Rick Jelliffe. 30/07/1999. *Weak Validation* .

[Jelliffe1999e] Rick Jelliffe. 01/08/1999. *Axis Models & Path Models: Extending DTDs with XPaths* .

[Jelliffe1999f] Rick Jelliffe. 1999. *Family Tree of Schema Languages for Markup Languages* .

[Jelliffe2000] Rick Jelliffe. 19/10/2000. *Getting Information Into Markup: the Data Model Behind the Schematron Assertion Language* .

[Jelliffe2001] Rick Jelliffe. 31/1/2001. *The Schematron Assertion Language 1.5* .

[Lee] . *Comparative Analysis of Six XML Schema Languages* .

[Norton] Francis Norton. 20/5/1999. *Generating XSL for Schema Validation* .

[OgbujiC] Chimezie Ogbuji. *Validating XML with Schematron* .

[OgbujiU] Uche Ogbuji. *Introducing the Schematron* .

[Quick] Rick Jelliffe. *Schematron Quick Reference* .

[Raggett] Dave Raggett. May 1999. *Assertion Grammars* .

[RDDL] Jonathan Borden and Tim Bray. 22/1/2001. *Resource Directory Description Language* .

[Schematron] Rick Jelliffe. *Schematron Home Page* .

[WAI] Wendy Chisholm, Gregg Vanderheiden, and Ian Jacobs. 5/5/1999. *Web Content Accessibility Guidelines 1.0* .

[XHTML] Mark Baker, Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Ted Wugofski, and Toshihiko Yamakami. 19/12/2000. *XHTML Basic* .

[XMLNames] Tim Bray, Dave Hollander, and Andrew Layman. 14/1/1999. *Namespaces in XML* .

[XMLSchema] David Fallside. 24/10/2000. *XML Schema Part 0: Primer* .

[XPath] James Clark and Steven DeRose. November 1999. *XML Path Language (XPath) Version 1.0* .

[XSLT] James Clark. November 1999. *XML Transformations (XSLT) Version 1.0* .

[Zvon] Miloslav Nic. *Zvon Schematron Tutorial* .

Collaboration and Dissent: Challenges of Collaborative Standards for Digital Humanities (pre-print version)

Julia Flanders

Collaboration—literally a shared work—is always understood to carry with it some kind of sacrifice, a tradeoff between autonomy and synergy. In our collaborative relationships, we intensify the concessions we make to the demands of the social contract, and we voluntarily submit to norms of behavior and constraints on our freedom of action in order to gain the benefits of a group undertaking: a barn-raising, a collection of essays, a successful conference. But even before we as collaborators can adopt these norms, we (in a larger sense) also have to develop them. The collaboration of conversation is predicated on the norms of language; collaboration on a scholarly edition is predicated on a set of private agreements about the editorial mission, set against the backdrop of larger disciplinary expectations concerning what editing means and how it proceeds. So we might start by observing that collaboration takes place within frames of expectation that may be private, local, professional, or broadly social. The vectors of agreement and conformance are thus not solely between the collaborators themselves, but also between the collaborators and some standards that operate beyond their own sphere of activity.

What makes collaboration between projects in the digital humanities particularly interesting —and particularly challenging—is the way these different frames of expectation intersect. Digital humanities projects take place, strikingly, in a universe constrained by a set of technical norms that govern the informational and operational behavior of the digital environment. Because these collaborations are so often aimed at building something that works—a tool, a resource, an online collection—the collaborative activities are typically mediated through things like software tools and data standards that are quite uncompromising. For instance, XML documents must obey the rules of structural well-formedness prescribed by the XML standard; dates entered into a metadata record must follow the agreed-upon format or the sorting routines that operate on this data won't work. (Geoffrey Nunberg's 2009 critique of Google Books' metadata illustrates the consequences of failure vividly.) These technical standards mediate the collaboration by eliminating the need to create all of these conventions from scratch, and the more exacting they are (and the more exactly they are observed) the more powerful the technical outcome. High-function digital projects arise from strong adherence to the right standards.

But in addition, as these projects arise from humanities research, they also require agreement concerning disciplinary norms that shape the practices of digital representation. These include, for instance, acceptable practices of transcription, regularization and emendation; acceptable standards of authenticity and verification; the kinds of commentary and contextualization that are acceptable or required; and beliefs about the interpretive or analytical or critical goals that are at stake. These norms arise from detailed, ongoing debates concerning both the ultimate goals of scholarship and the methods and practices by which we achieve them.

A traditional humanities view of this picture might ask what these two sets of norms have to do with one another, any more than the editorial standards for a critical edition have to do with the

standards guiding the machinery that binds the book. But what the field of digital humanities as a whole has revealed is precisely how tightly interwoven and mutually consequential "technical" and "disciplinary" standards often are. Our concern in this discussion is in fact a domain in which disciplinary and technical norms overlap and where we see, locked in struggle, the drive towards absolute consistency and technical processability on the one hand, and the drive towards critical independence and disciplinary debate on the other. I am referring here to the arena of standards for digital representation of research materials, and in particular the domain of scholarly markup languages. Standards of this kind are an essential precondition for the development of collaborative projects in the digital humanities, because they permit us to exchange scholarly information about research materials in a digital form.

A markup language is a method of digital representation through which we can create highly complex models of textual artifacts. The markup constitutes an information structure which formalizes, labels, and enhances the source material: it makes explicit the nature and function of each piece of information, and through that explicitness makes it possible for non-human processes (such as computers) to "understand" and make use of the resulting digital object in ways that take advantage of its self-knowledge about its own structure and content. A manuscript letter marked in this way can carry with it the knowledge of which words have been deleted or replaced by the author, and a software system can use that knowledge (for instance, to generate a final edited version of the text). A collection of such manuscripts, if marked consistently, can yield a systematic analysis of their authors' revision practices.

Text markup of this kind, as practiced in the digital humanities world, sits at the juncture of humanities scholarship—textually nuanced, exploratory, and introspective—and digital technology, with its emphasis on formalism, consistency, and upward scalability. As a result its norms carry a double weight: they must achieve some kind of technically actionable uniformity, but they must also express useful scholarly concepts and differentiations. They draw their functional precision and power from the domain of technical norms, and also their capacity to broker collaborative exchange at a practical level. But they draw their relevance and intellectual usefulness from the domain of disciplinary norms, which as we have seen are fundamentally and on principle resistant to standardization, and in fact might be said to thrive like yeast in a state of fermentation (though without drowning in their own byproducts).

Debates about how to balance these competing concerns in the markup community—and particularly in the community of the Text Encoding Initiative—are lively and sustained. The goals of standardization and facilitated interchange were articulated by the planners of the TEI at the earliest stages of their discussion in the Poughkeepsie Planning Conference of 1987, as documented in the first of the so-called "Poughkeepsie Principles":

> Poughkeepsie Principle 1: The guidelines are intended to provide a standard format for data interchange in humanities research. (TEI, 1988)

These goals have remained strongly in view during the succeeding years, during which the TEI Guidelines have undergone sustained development and use and have become one of the most deeply researched features of the digital humanities landscape. At the same time, the challenges of maintaining the double force noted above—the power of both strong formalism and expressive nuance—have become much more vividly apparent. For some developers, the attempt to maintain

expressiveness simply undermines the goal of interchange and places the entire enterprise at risk. Characteristic of this view is the following quote from a 1997 posting by Mark Olsen to the Humanist discussion list, in which he is complaining about the laxity and permissiveness of the TEI Guidelines as a technical standard, a few years after the first public release of those Guidelines:

> The real test of an INTERCHANGE format, however, is … that the format can be automatically converted TO and FROM any number of systems with a minimum of effort. My principal objection to TEI is that it is by far the most difficult representation to convert into something else, because of its expressive power. The more tightly constrained a specification, the easier it is to write converters. It is a BALANCING act, which I do not believe the TEI community has -- because of its make-up and structure -- really tried to perform. (Olsen 1997, emphasis in original)

This position has also been strongly argued in subsequent debates about the nature and value of what is termed "TEI conformance": that is, the formal methods by which adherence to the TEI Guidelines can be assessed. The rationale for the current definition of conformance, in which conforming documents must be a strict subset of the unmodified TEI,[1] focuses on several practical goals involving the exchange of data:

- interchange or integration of documents amongst different researchers or users;
- software specifications for TEI-aware processing tools;
- agreements for the deposit of texts in, and distribution of texts from, archives;
- specifying the form of documents to be produced by or for a given project.

(TEI 2007, 23.3)

The countering position, equally strongly felt and argued, is that one can only achieve such strong standardization and uniformity by eliminating a degree of intellectual depth and nuance that is essential to the intellectual quality of the enterprise. Michael Sperberg-McQueen, one of the original editors of the TEI Guidelines, responded to an earlier version of Mark Olsen's plea above by offering him an extremely reduced and simplified subset of the TEI and

> demonstrating, by a reductio ad absurdum, how reducing a tag set to this size … forces one to omit too much material which can be useful in the encoding of virtually any text, and which is absolutely essential for dealing rationally with some texts. (Sperberg-McQueen 1995)

Sperberg-McQueen's point about textual complexity is even more true when we consider the role markup plays in representing not just texts but our views about texts: our methodological assumptions, our editorial decisions, our critical debates. The prospects for representing these in a

---

[1] That is, they must use only elements and element structures that are defined in the unmodified TEI; any conforming TEI-encoded document must be valid against the unmodified TEI schema or be transformable into a valid document with no loss of information.

simple and uniform manner are generally acknowledged on both sides of the debate to be extremely poor. Clearly these arguments would not apply to projects whose goals for text representation are quite simple (for instance, digital library projects where the transcription serves only to provide searchable full text to accompany a page image). But for projects in which the TEI functions as a representation of scholarly work, the accommodation of nuanced intellectual expression is clearly crucial.

In debates about this issue, it is usually assumed (as Mark Olsen assumes) that a strongly enforced uniformity will facilitate collaboration, and conversely that heterogeneity and dissent will militate against it. Our instinct may thus be, in the context of digital standards, to treat disciplinary debate as the opponent: something that needs to be eliminated or ignored in order for collaboration to proceed. I would like instead to suggest otherwise: that in fact the humanities dimension of digital humanities work makes this elimination impossible and also undesirable. But it will help if we can first establish a more nuanced and carefully structured view of the precise role of dissent within our collaborative ecology.

Collaboration and Dissent

One place where collaboration has been examined from a philosophical and political perspective (rather than a business-efficiency perspective) is in theories of collaborative learning, and John Trimbur in a 1989 essay entitled "Consensus and Difference in Collaborative Learning" provides a particularly thoughtful and critical analysis of some key points: he is considering the dynamics of consensus within learning environments but also making a larger point about the way that social negotiations take place. Trimbur marks consensus as central to collaboration, but for him, a crucial element of the discursive practice of consensus is dissent: not just the need to democratically acknowledge a minority view, but the need to base consensus on "collective explanations of how people differ" (610). This "dissensus" is a representation of fractures within the discourse: an acknowledgement of the existence of the periphery, the discourses out of power:

> We will need, that is, to look at collaborative learning not merely as a process of consensus-making but more important[ly] as a process of identifying differences and locating these differences in relation to each other. The consensus that we ask students to reach in the collaborative classroom will be based not so much on collective agreements as on collective explanations of how people differ, where their differences come from, and whether they can live and work together with these differences. (610).

Dissensus thus functions as a way of establishing a critical relationship between the consensus and the dissenting voices. In other words, dissent occupies an informationally important and illuminating position in the universe: "not as the goal of the conversation but rather as a critical measure to help students identify the structures of power." With this understanding of the role of dissent, Trimbur argues, we can finally see consensus not as a rational, realizable goal (akin to finding out that we really all do want the same thing) but rather as a recognition of the "inexhaustibility of difference" and a will to "organize the conditions in which we live and work accordingly" (615).

This argument offers a very interesting perspective on standards development, and one that understandably doesn't get much play in that domain. However, the digital humanities is in a position to attend to the fact that standards do arise from and represent power structures; they represent the functional homogenization that is one outcome of "community": the consent to being homogenized. And as a community-driven standard arising from the digital humanities, the TEI is rare (but perhaps also characteristic) in taking seriously the legitimacy of dissenting views while also seeking a technically functional outcome. It needs such views to exist—indeed, it relies on their existence as the driving force behind its own onward progress—but it wants them to exist *in relation to the community at large*, as part of the discourse rather than apart from it.

We can now usefully circle back to comment on the word "standard" in the context of Trimbur's observations, especially since in an important sense the TEI is not in fact a standard according to a strict definition of the term. The sphere in which strict definitions operate with relevance to the digital humanities is that of governmental and para-governmental standards (such as the American National Standards Institute), international non-governmental organizations (such as the International Organization for Standards), and industrial standards bodies (such as the World Wide Web Consortium). Standards promulgated by such bodies have the force, if not always of law, at least of definitional hegemony: a standard for measuring photographic film speed or tuning frequencies sets the parameters within which all activities will effectively operate.

Standards by their nature thus impose a uniformity that, in effect, creates consensus by fiat. Once they are in place, we obey them because we value the consensus they represent (and the cohesion and practical efficiency they enable) more than our own individual right to do things differently. Furthermore, within the domain of digital humanities, adherence to (open) standards is framed as a kind of good citizenship, the necessary precondition for a free interchange of data. The world without standards is a world of chaos—a post-Babel cacophony of conflicting and mutually unintelligible voices. But it is also a world of plurality, and the transition from that plurality to the uniformity of the standard is worth noting as an important change of state. The web site of the British Standards Institution includes this succinct statement on the nature of consensus standards, which places its emphasis tellingly on this exact point:

> All formal standards are developed with a period of public enquiry and full consultation. They incorporate the views and expertise of a very wide range of interests from consumers, academia, special interest groups, government, business and industry. As a result, standards represent a consensus on current best practice. (BSI 2010)

The deftness with which we move from that "wide range of interests" (which has a double force: both a variety of differing interests, and a representative set of interests) to the "consensus on current best practice" elides what must necessarily have happened: the elimination or bypassing by one means or another of that original plurality of views.

In fact, hard-headedly, standards present us with a balance or tradeoff: they represent an operation of power, which may have been exercised with more or less attention to minority opinions, and may place more or less constraint on our meaningful freedom of action, and may provide more or less practical benefit to us and to society. Different standards bodies and different standards sit at different points along these three axes. It is hard to feel very put out about not being consulted

about the standard for measuring the twist of single-ply yarn, but many of us may feel a greater stake in the ISO standard for representing the sex of humans, in which the permitted values are 0 (not known), 1 (male), 2 (female), and 9 (not specified) (ISO 2004). But even the best arrangement of these factors—the most open process, the least constraint, the greatest practical benefit—in the end does represent as "consensus" an end-product about which there actually has been, and may still be, disagreement. There is no need to standardize what no one differs on.

Collaboration and the TEI

The TEI occupies a distinctive position within this landscape—first of all, because it does not claim the status of a standard for its Guidelines. They "make recommendations" about methods of representing textual sources in digital form, but these recommendations have their force solely within the sphere of activity of the TEI itself: in order to claim to be using the TEI Guidelines, you have to use the TEI Guidelines. This internal regulation of the TEI community positions the TEI as a standards body *for that community of usage*, and for this reason the TEI is often described as a "community standard," both in its own self-description and by others. But this terminology deserves some scrutiny because, even thus qualified, it aligns the TEI's guiding enterprise with goals of uniformity and consensus that, as we have seen above, have proven problematic from the outset. In order to understand how this "community standard" operates *as a standard* within the TEI community and hence as a collaborative tool, we need to understand the role that dissensus, in Trimbur's sense, may play in its formation and use.

Encoding standards like the TEI are foundationally collaborative technologies: they presume the need and the desire to coordinate shared work, to generalize individual insight to a community, and to support the extension, critique, and reuse of ideas and techniques. But how, concretely, does the TEI figure in a typical digital humanities project? A simple example would be a pair of scholars who are working together on a digital edition of a very long manuscript that must be transcribed and annotated: for instance the Almanacks of Mary Moody Emerson—a 1000-page manuscript currently being edited and encoded by Sandra Petrulionis and Noelle Baker in collaboration with the Women Writers Project (WWP) at Brown University. If they divide up the manuscript, each taking a section, their agreement on a shared encoding system such as the TEI makes it possible for them to work together while also working separately: the encoding standard diminishes the need for day-to-day consultation on details of digital representation. When they combine their work, it should form a consistent whole, and when they hand it at last to the WWP for publication, it should fit in with the other materials in our collection that have also been prepared in this way.

In this typical kind of collaboration, the parties to the shared effort know of each other, and their intention to work together is explicit; in effect, a collaborative vector is established between them that operates in real time, mediated by their use of the encoding standard. As a more complex example, however, take a project like the Electronic Archive of Greek and Latin Epigraphy, which is a federation of epigraphic databases: online collections of ancient inscriptions on stone, metal, clay, and other durable surfaces. Scholars in several different countries want to be able to contribute to such collections, knowing that in doing so they are contributing to a single, vast, comprehensive resource rather than to one of many that are small, isolated, and partial. In order to accomplish this goal, they need a common standard for transcription and editing, and in fact this group is now using the TEI as the basis for their digitization efforts: the TEI-based epigraphic

transcription standard is known as EpiDoc. In this collaboration, although the scholars involved know that they are contributing to a shared resource, they don't necessarily feel themselves to be working at all times directly *with* all of the other scholars: the model is more of a hub and spokes rather than a direct person-to-person network.

This kind of indirect collaboration leads us in turn to a third case which has particular importance for the digital humanities. Imagine the same hub-and-spokes model, but now let us think of the spokes as extending not only into space but into time as well. The recent rise of interest in humanities data curation demonstrates how crucial these longitudinal collaborative vectors can be. Digital resources that we create today do not exhaust their value in the first few years of use; on the contrary, unlike research in the sciences which has a fairly deterministic half-life, humanities research materials have a horizon of usefulness that is much more complex and unpredictable and prolonged. The British Women Romantic Poets project is a substantial collection of Romantic-era poetry by women created at University of California, Davis in the late 1990s; it has been available on the web in its original form for years.[2] But in addition, this project contributed texts from their collection to the WWP's online collection, Women Writers Online, and the BWRP collection is now also included in NINES, the large federated collection of resources for the study of nineteenth-century literature.[3] The WWP and NINES are taking advantage of the intelligibility of the BWRP data—its use of standards like the TEI—to do things with these materials that were not specifically envisioned by their creators. But the intelligibility of the data in itself—the intention that it be reused—constitutes an important collaborative gesture. The "collaboration" here is real and important and yet takes place across time and without direct interaction.

This kind of indirect collaboration is at present rare—and indeed, there is a strong anecdotal "literature" attesting to the difficulties in accomplishing it—but it exercises a powerful mythic influence on the ways we think about how we create digital humanities resources and why. As *digital* humanists we are committed to encoding our data in a standard way so that it can be shared, so that it will remain comprehensible: in fulfillment of an implicit contract with unknown scholars of the future who need to know what we know and understand what we have done. But at the same time, as we've already seen, as digital *humanists* we know that a crucially important dimension of that representation is precisely the disciplinary norms we adopt, and these we know to be founded on debate rather than on straightforward agreement. We can assume, in other words, that our future collaborators will disagree with us on some fundamental point, while nonetheless wishing to make use of what we have done. So the question then is: what needs to be expressed in order for this kind of longitudinal collaboration—one in which the first collaborator has to play their full hand before the other even comes into the game—to work? Is there any conduit through which such a negotiation can take place?

There is, but it takes a complex form. What is needed in this case is not simply a common language for data representation, but more importantly a common mechanism for *negotiating about data*

---

[2] See http://digital.lib.ucdavis.edu/projects/bwrp/.

[3] NINES is the Networked Infrastructure for Nineteenth-Century Electronic Scholarship, http://www.nines.org.

representation. And that, I would like to argue, is exactly what the TEI properly provides: not a perfect language for collaboration, but a—let us call it a functional—mechanism for negotiating about language: in fact, a mechanism for negotiating dissent.

TEI Customization

In order to give some concreteness to this assertion, we need to take a brief excursus into the details of the TEI customization mechanism, which is the form given to this negotiation process. The TEI encoding language is not a single, unitary thing: it is a vast landscape of possibility that covers domains as remote as manuscript description, dictionaries, drama, oral history, linguistic corpora. The TEI community as a whole needs to be able to represent all of these things, but any individual project needs just a selection: manuscripts and drama, say, or verse and scholarly editing. So the TEI language is represented at the most basic level as a set of possibilities: a single source file (called the "ODD"[4]) that represents the entire landscape of the TEI in potential terms.

To represent our own, more selective view of that landscape, we need another piece of information, a separate file which is called an ODD customization. The ODD customization file represents the world of an individual user or project: the set of choices through which the individual adapts the TEI schema to local usage. These choices may simply be selections from among the various parts of the TEI (elements to encode manuscripts, elements for scholarly editing, elements for representing dictionaries or drama or verse). They may also take the form of added constraint: for instance, a project may wish to limit the vocabulary used to describe poetic genres. But they may also be extensions: added elements for things that the TEI cannot yet represent, or changes to TEI elements to reflect local needs.

From these two files (the main TEI ODD, and the ODD customization), with appropriate processing, one can then generate a schema that expresses the TEI landscape as viewed through the lens of the individual application. And by comparing any two (or more) customization files, one can also in principle gain an understanding of how two projects differ in their representation of data. There are thus several potential vectors of difference or dissent that can be expressed here: between the individual researcher or project and the TEI community as a whole (expressed through actual changes and extensions to the unmodified TEI language), and also between different individual researchers or projects.

To give a concrete example, let us take the Women Writers Project again: a long-standing TEI project with a somewhat idiosyncratic set of texts, experimental views on text encoding, and a strong collaborative instinct. The WWP's TEI customization captures a number of important things:

- the specific sections of the TEI that the project actually uses (for instance, the TEI module for names and dates, the module for linking)
- the controlled vocabularies used in classifying things like verse structures

[4] The origin of this terminology is whimsical: because the ODD file represents both the schema itself and the documentation of the schema, "one document does-it-all".

- the changes the project has made to individual textual structures, to accommodate the idiosyncrasies of the WWP's texts or of their thinking about them

So in this case, the customization immediately serves two important functions: first, to express and document the rationale behind the WWP's encoding practice; and second, to express in formal terms the relationship between that practice and the TEI guidelines themselves.

Imagine, though, that in a few decades, after the project has run its course and its data is part of the Brown University repository, another group of researchers (perhaps working on women's writing in French) wants to use this data, in combination with their own, to compare ideas of genre across languages. In the wild, this project is utterly hopeless. But if both groups are using the TEI, the process is eased: the peculiarities of each project's encoding are expressed as points of dissent from the TEI itself: in other words, from a known quantity. In areas where the WWP does not disagree with the TEI, it uses the standard terms and structural concepts. The second project, coming later, is able to use the standard TEI markup present in the existing project data unmodified; from the WWP's ODD file they can also see explicitly where this encoding dissented from the TEI consensus, and assess how these points of difference accord with their own methods (and modify the WWP data if necessary). They can also bring in data from other TEI projects on the same basis: because all of the dissent has been organized into a set of vectors all pointing in towards the same hub (instead of being expressed combinatorially), the challenge of harmonizing the data and discerning the points of essential agreement and disagreement is significantly diminished (though of course not eliminated).

What is taking place here is a form of negotiation that manages in a subtle and bizarre way to be bi-directional even though it takes place across the passage of time. Each project, through its TEI customization, places its data in an explicit relation to the standard, with any reservations or qualifications clearly expressed: thereby deliberately putting the data "up for collaboration". What is distinctive about the TEI customization mechanism is that it formalizes dissent in this way, in relation to the standard, and allows its vector to be traversed in two directions. The same path that leads away from the unmodified TEI standard towards the individual application (from generality to specificity) can also be followed back to the center again. This traversal can be effected both by human beings and by computer processes. Information concerning what has been changed and why can be expressed in human-readable form and may serve as a valuable support in understanding the methodological choices that underlie a project's encoding practice.

Similarly, the ODD customization file can serve as the basis for automated analysis of difference and similarity of encoding methods. By analyzing a set of customization files, for example, we could identify all projects from a large set that use the same set of TEI modules or remove the same set of elements. We could generate a list representing the greatest common set of values for a given vocabulary (such as poetic genres) across a group of projects and also identify the values that are unique to each project, or identify the range of new elements created by each project, together with their closest TEI equivalents.

Interoperability and Collaboration

Taken as a whole, the customizable approach taken by the TEI permits the standard to function (both socially and technically) as an agreement at many levels—on the intention to treat data as a sharable and preservable resource, on the value of shared data standards, on the descriptive utility of this particular approach to modeling humanities texts, and on the impossibility of creating a single descriptive model that will satisfy all needs.

This last point may seem like a significant or even fatal concession, and it may be appropriate to consider here whether collaboration, in the absence of a single descriptive model, is really possible at all. Would it not enhance the collaborative effect in the examples given earlier, for instance, if instead of the elaborate articulations of the customization mechanism we had a simple and uniform standard for exchanging data in an unambiguous manner? Is this not in fact the mission with which the TEI was initially begun? Martin Mueller put the case for the value of uniformity in a presentation at the 2008 TEI conference:

> To compare is to look for significant difference, but without comparability difference cannot be identified in the first place….consistent encodings across many texts in a heterogeneous document space have a greater scholarly pay-off than finely grained encodings in closely defined but not necessarily compatible environments. (Mueller 2008)

From this perspective, in which the goal of interoperability is given primacy, the heterogeneity of different encoding approaches looks simply like grit in the works.

The challenge here is to understand the relationship—and the difference—between interoperation and collaboration. Interoperation is a functional property of data and tools: a measure of how seamlessly a new data set will perform within a system not explicitly designed for it. If I have written a stylesheet that expects a certain kind of TEI data, and your data flows through it without producing error messages or an unprocessable residuum, then your data is interoperable with my stylesheet.

Or is it? What if our measure of seamlessness here is not the unimpeded working of the tool, but the meaningfulness of the output? Take as an example a stylesheet that is designed to generate an index of first lines in poetry. It looks for the first "verse line" child of each "poem", extracts it, and sorts the resulting list. Imagine now that my collection of poems includes some with epigraphs or introductory notes following the poem title. If the schema for poetry does not include separate markup elements for these features and I cannot alter it, I may be compelled to represent them as part of the poem itself, using the element designated for verse lines. The stylesheet, none the wiser, obediently identifies these and extracts them as "first lines" without complaint, and the resulting index contains a mix of first lines and things that are not first lines but cannot be distinguished from them. From the standpoint of interoperability the error is hard to detect. The data itself is structurally indistinguishable from what the system anticipates; the problem is simply that the data is lying.

This problem is not a belated discovery, but a deep tension that has inhabited the design of the TEI from the start. Let us now recall, from earlier on, Poughkeepsie Principle 1: "The guidelines are intended to provide a standard format for data interchange in humanities research." This seems

like an unambiguous statement about the primacy of interchange and its dependency on concepts of standardization. But in the "Design Principles for Text Encoding Guidelines" which comment and expand on these principles, the editors of the TEI Guidelines note concerning this point:

> For interchange, it must be possible to translate from any existing scheme for text encoding into the TEI scheme without loss of information. All distinctions present in the original encoding must be preserved. (TEI 1988)

Meaningful interchange and interoperability thus require us to attend not only to the structural uniformity of our data, but to its semantic uniformity as well[5]: in other words, the distinctions being preserved in one encoding must be mapped onto the distinctions that are relevant in another encoding. The brief admonition that "all distinctions present in the original encoding must be preserved" is in fact a statement in support of the distinctiveness of that "original encoding" and the legitimacy of its distinctions: the role of the interchange format is to permit translation, not to efface difference. As Sperberg-McQueen's admonition to Olsen insists, uniformity comes only at the cost of eliminating what is "absolutely essential for dealing rationally with some texts." Fifteen years after that exchange, with digital editing now an important application of the TEI, we might expand this point to read "…and for dealing articulately with some editorial problems." The uniformity that cannot be assumed in our primary sources is equally rare in our scholarly methods. To collaborate effectively under these circumstances is thus a matter not of enforcing an artificial uniformity through which vital distinctions are elided, but rather of supporting the real and accurate exchange of the data in which we have a strong stake. The fundamental question, in other words, is whether we really want to hear what our collaborators really have to say.

This is not an argument for diversity for its own sake. In a strong collaborative ecology, the corollary of respecting each others' differences is respecting the commonalities that draw us into joint work in the first place. Structurally speaking, the TEI customization mechanism is as well suited to creating schemas that reflect a well-honed, tightly constrained expression of those commonalities as it is to expressing individual divergences from the consensus. But while a well-constructed schema can represent and support a consensus of this kind, it cannot create it. A standard like the TEI, in other words, is not the appropriate mechanism for *producing* a successful collaborative agreement except in cases where the quality of the data (its semantic nuance or its methodological rigor) is of no consequence. It is, rather, an extraordinarily effective mechanism for *representing* such an agreement once it has been made by the collaborating parties.

This point complicates what is commonly assumed about data standards in the digital humanities: that they support collaboration unproblematically, and that the better we standardize, the stronger our collaborations will be. Representational systems like the TEI operate in a domain where our descriptions, in their complexity and their consequentiality, reach towards the condition of human language and thought. This complexity is a strength and a measure of the intellectual potential of these representations, not a weakness. The challenge—as with human language—is to achieve mutual intelligibility while still being able to say what is worth saying. Collaboration, too, walks

---

[5] I am indebted to my colleague Syd Bauman for sustained discussions of this point that have helped me understand its significance.

this precarious line between egoism and altruism, between private insight and public communication, between local nuance and common ground. In this respect, the TEI is not simply a tool for collaboration but a methodological reflection of its structure at the deepest level.

Works Cited

Allen, C. 2007. Guidelines for Supporting ISO Code Sets. Available at: http://ns.hr-xml.org/2_5/HR-XML-2_5/CPO/GuidelinesForISOUtilities.html [accessed: 20 May 2010].

British Standards Institute. 2010. What are the differences between consensus and commissioned standards? Available at: http://www.bsigroup.com/en/Standards-and-Publications/About-standards/Differences-between-Consensus-and-Commissioned-standards/ [accessed: 20 May 2010].

Burnard, L., Bauman, S., eds. 2007. *Guidelines for Electronic Text Encoding and Interchange*. TEI Consortium. Available at: http://www.tei-c.org/release/doc/tei-p5-doc/en/html/index-toc.html [accessed: 20 May 2010].

ISO. 2004. Information technology — Codes for the representation of human sexes. Available at: http://standards.iso.org/ittf/PubliclyAvailableStandards/c036266_ISO_IEC_5218_2004(E_F).zip. See also http://en.wikipedia.org/wiki/ISO_5218 [accessed: 20 May 2010].

Mueller, M. 2008. TEI-Analytics and the MONK project. TEI annual conference, London, November 6–8, 2008. Available at: http://www.cch.kcl.ac.uk/cocoon/tei2008/programme/abstracts/abstract-169.html [accessed: 20 May 2010].

Nunberg, G. 2009. Google's book search: a disaster for scholars. *The Chronicle of Higher Education.* [Online] August 31. Available at: http://chronicle.com/article/Googles-Book-Search-A/48245/ [accessed: 20 May 2010].

Olsen, M. 1997. Humanist Discussion Group, Vol. 10, No. 895. Available at: http://www.digitalhumanities.org/humanist/Archives/Virginia/v10/0515.html [accessed: 20 May 2010].

Sperberg-McQueen, C. M. 1995. Preface to TEI Bare. [Online] Available at: http://www.tei-c.org/Vault/Bare/ [accessed: 20 May 2010].

TEI. 1988. Design principles for text encoding guidelines." TEI ED P1. Available at: http://cmsmcq.com/1990/edp1.html [accessed: 20 May 2010].

Trimbur, J. 1989. Consensus and difference in collaborative learning. *College English* 51.6 (October 1989), 602-616.

Extreme Markup Languages 2002

Montréal, Québec
August 6-9, 2002

Drawing inferences on the basis of markup

C. M. Sperberg-McQueen
World Wide Web Consortium

David Dubin
University of Illinois at Urbana/Champaign

Claus Huitfeldt
Avdeling for kultur, språk og informasjonsteknologi

Allen Renear
University of Illinois at Urbana/Champaign

Abstract

Various authors have sketched out proposals for identifying the meaning, or guiding the automated interpretation, of markup, sometimes with the goal of using the information expressed by markup to guide the extraction of information from documents and using it to populate reasoning engines. We describe one approach to the problems of building a system to perform such a task.

Drawing inferences on the basis of markup

Table of Contents

1 Introduction 1
2 The problem 2
    2.1 An example 2
    2.2 A Logical Approach 3
    2.3 Kinds of sentences 3
    2.4 Two approaches: inference and direct instantiation 6
        2.4.1 Inference-driven approach 6
        2.4.2 Direct-instantiation approach 6
3 Document representation 9
4 Propagation sentences 11
5 Application sentences and further inferences 11
6 Skeleton sentences and mapping rules 15
7 Some challenges 17
Footnotes 18
Bibliography 19
The Authors 20

EXTREME MARKUP LANGUAGES 2002

Drawing inferences on the basis of markup

C. M. Sperberg-McQueen, David Dubin, Claus Huitfeldt, and Allen Renear

§ 1 Introduction

Text encoding has traditionally promised to make explicit our understanding of (or: theories about) a document ([Coombs et al. 1987] [Coombs et al. 1987], [Sperberg-McQueen 1991]). Well-designed SGML/XML markup languages such as Docbook [Walsh/Muellner 1999] or the TEI [ACH/ACL/ALLC 1994] appear to be fairly successful at this. However, a close look reveals that there are limits to the degree of explicitness that can be achieved with current encoding languages. The problem may be traced to the fact that although SGML/XML-based markup languages provide explicit rules for syntactic well-formedness and validity, they provide nothing analogous for semantic correctness. As a result they are reasonably well suited for generating data structures, but are not, at least not without further development, as effective at expressing interpretations [Renear 2001]. And many developers of programming systems have wished they had a mechanism for specifying, more exactly than XML 1.0 DTDs allow, exactly what application data structures or what tables and columns in a SQL database are to be built from XML data when it is received.

Our immediate area of concern is the task of providing a clear, explicit account of the meaning and interpretation of markup.

Scores of products and projects which use XML and SGML assume implicitly that markup is meaningful, and use its meaning to govern the processing of the data. A number of authors have described systems for exploiting information about the meaning or interpretation of markup; among those most relevant to the work described here are [Simons 1997], [Simons 1999], [Welty/Ide 1997], [Ramalho et al. 1999], [Sperberg-McQueen et al. 2001a], [Sperberg-McQueen et al. 2001b], and [Thompson 2001].

There are a variety of reasons to be interested in this problem. Better understanding of the meaning properly assigned to markup seems certain to make it easier to write better, smarter tools for creating, managing, and exploiting markup. [Ramalho et al. 1999] make a strong case that it will enable substantially improved validation and quality assurance, enabling automated systems to detect a large number of errors in the use of markup or in the data which cannot be detected by purely syntactic or datatype-oriented methods. [Welty/Ide 1997] argue that a more formal approach to markup semantics will dramatically improve information retrieval. Even if the kind of logical inference they have in mind proves too time-consuming to perform at query time, it may be helpful in reducing inessential variation in markup practice within a collection, or in masking the variation in markup within heterogeneous collections [Schatz et al. 1996]. For non-documentary uses of XML, the meaning of markup may be seen in its mapping to target data structures — or rather the meaning can be seen in the range of target application data structures the markup may legitimately be mapped to; this position has been put succinctly by Henry Thompson in discussions of earlier work on this topic, and underlies the work reported in [Thompson 2001]. On a purely practical level, experience with systems of the kind we describe here can be expected to provide useful information about how to make the documentation of markup applications clearer and more useful.

Our main motivation for this work, however, is not so much its manifold practical applications as its intrinsic interest.

In earlier work [Sperberg-McQueen et al. 2001a], we proposed to identify the meaning of markup in a document as the set of inferences licensed by it[1] and outlined a 'straw man' proposal for defining the proper interpretation of markup in a given markup language and formulating certain simple rules for mechanically generating the proper inferences from documents marked up in that language. Implementation of the straw man proposal demonstrated that the straw man proposal is too simple

to explain real markup languages; its rules lead to a number of false inferences. In [Sperberg-McQueen et al. 2001a], we sketched in general terms how a better account of meaning and interpretation in markup could be constructed; this paper reports on work toward the concrete realization of one part of that 'framework' model and will outline some of the problems encountered in specifying the inferences licensed by commonly used DTDs.

§ 2 The problem

The framework outlined in [Sperberg-McQueen et al. 2001a] includes:

- some representation of the document, probably in a form which can be used with some existing inference system. In our current work, we use a Prolog interpreter as our inference engine.
- a set of *skeleton sentences*. These are sentence schemata or patterns which can be used to form sentences which express the meaning of each construct in the markup language. A skeleton sentence is like a natural-language sentence, or a Prolog predicate, or an expression in some other formal system, in which certain words or items have been replaced by blanks. When the blanks are filled in properly, a normal sentence in the language is the result. (Some readers will be reminded of the game marketed a few years ago under the name Mad Libs.) In the systems we describe here, the blanks are typically to be filled in with information from the document itself, and each blank is associated with a *deictic expression* showing how to fill it in.[2]
- some language, possibly a rather small one, in which to write the *deictic expressions* which can be associated with the blanks in the skeleton sentences; in common languages like Docbook, HTML, and TEI, it is easy to foresee the need for expressions meaning "the contents of this element", "the value of this attribute", "the nearest ancestor of type *bibl*", "the value of the *xml:lang* attribute on the nearest ancestor element which has a value for it", etc.
- some categorization of predicates according to the rules governing inferences from them. For example, some properties of elements are inherited by descendant elements, and others are not — a satisfactory account of the meaning of markup should capture this general distinction and others like it.
- some generic routines for generating statements about the document by extracting suitable information from the document and using it to fill the blanks in the skeleton sentences, thus forming full sentences in the target language.
- (optionally) rules allowing further inferences. These are often not closely tied to specific markup of specific document instances, but may express more general rules about properties expressed by markup (e.g., "if something is an author, and not identified as a corporate author, then it is a person", or "if something is a person, then it is human").

Several of these components will be discussed separately below.

2.1 *An example*

To illustrate the basic ideas, let us consider as an example the sample purchase order used in the Primer for W3C XML Schema 1.0 [Fallside 2001]. If the meaning of markup is to be found in the set of inferences we can draw from it, what inferences can be drawn from the sample purchase order, and how?

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
```

```
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

The sample document above shows an order for a lawnmower and a baby monitor; the items are to be shipped to one Alice Smith of Mill Valley, California, and billed to one Robert Smith of Old Town, Pennsylvania.

In the following sections, we will illustrate the discussion with examples drawn from this document.

2.2 *A Logical Approach*

Our general approach to the problem of representing the meaning of markup is the construction of a formal logical system. This system will serve both a theoretical objective: illuminating *how* document markup licenses inferences; and one practical one: supporting the mechanical calculation of the inferences.

As with most efforts to formalize a domain area we do not begin from scratch, but rather accept the expressive devices and deduction rules of an existing system of predicate logic, and supplement those with (i) constants distinctive to our domain of interest; and (ii) a set of 'axioms' or 'premises' which reflect not fundamental truths about logic but rather fundamental facts about markup about which we wish to reason.

In translating crucial propositions from another notation into a more purely logical notation, we are following the practice of most work on programming-language specification and semantics; cf. [Guttag/Horning 1993], which explicitly makes it a goal of the Larch system to provide a basis (i.e. an axiomatic basis) for reasoning about programs. The notion of the meaning of some sentence as the set of inferences to be drawn from it is also borrowed from this field; it came to our attention from [Turski/Maibaum 1987].

2.3 *Kinds of sentences*

We provisionally identify several kinds of sentences in our system. In this section we list these kinds and give intuitive characterizations of each. We also give examples of what such sentence would mean, using a stylized English sentences that corresponds naturally and unambiguously to the expressions of predicate logic. Then the following sections we explore Prolog representations that support inferencing. Rigorous formal characterizations of these sets of sentences are underway and will be reported on in later publications.

image sentences
: These directly translate the marked up document into the notation of the logical system. In practice, they constitute a more or less literal and

mechanical translation from a given markup notation (e.g., XML) into a set of sentences. These sentences ascribe certain properties — such as having a certain generic identifier or a certain attribute name and value — to certain objects which map 1:1 to the elements, attributes, and so on of the input document. Strictly speaking the objects so described are not XML elements but the result of mapping XML elements (or element information items) into our logical system. We will call these objects *images* of the document instance; the information items to which the images correspond we will call their *pre-images*. For our purposes, we regard the translation into images sentences as sufficient if by working from the logical form alone we can construct an XML document with the same infoset [Cowan/Tobin 2001] as the original document.

Examples of image sentences:

- *a* has the generic identifier "*authorname*".
- *a* has the content "Alan Turing".
- *a* has the value "de" for the *lang* attribute.

The graph structure of the document instance must also be represented if the image sentences are to capture the basic information set of the input document. The following are thus also examples of image sentences:

- The first child of *a* is *b*.
- *b* has an immediately following sibling; this immediately following sibling is *c*.

property rules

These associate specific properties — such as being a quotation, being in the German language, or being a name — with the generic identifiers and attribute names of a particular markup vocabulary. Where appropriate these rules will also specify the conditions for inheritance of the property.

Examples of property rules:

- If *x* has the generic identifier "*authorname*" then *x* is a *name-of-an-author*.
- If *x* has the value "de" for a *lang* attribute, then: *x* is in the German language, and any descendent *y* of *x* is in the German language unless there exists some element *z* which is both a descendent of *x* and either an ancestor of *y* or identical to *y* and *z* has a *lang* attribute with a value other than "de".

propagation sentences

These sentences result from applying a property rule to an image sentence:

Image sentence:

- *a* has the generic identifier "*authorname*".

Property rule:

- If *a* has the generic identifier "*authorname*" then *a* is a *name-of-an-author*.

Resulting propagation sentence:

- *a* is a *name-of-an-author*.

In more complicated cases the generation of all implied propagation sentences also requires applying the inheritance conditions indicated by the property rule:

Image sentence:

- *a* has the value "de" for a *lang* attribute.

Image sentence:

- *a* is the immediate parent of *b*.

We note in passing that *b* does not have an attribute-value specification for the *lang* attribute, and there is thus no image sentence giving that attribute any value for *b*.

Property rule:

- If *x* has the value "de" for a *lang* attribute, then: *x* is in the German language, and any descendent *y* of *x* is in the German language unless there exists some element *z* which is both a descendent of *x* and either an ancestor of *y* or identical to *y* and *z* has a *lang* attribute with a value other than "de".

Resulting propagation sentences:

- *a* is in the German language; *b* is in the German language.

One may reasonably wonder whether the variables in the antecedent and consequent of property rules really should be ranging over the same set of objects, with the result that one and the same thing both, e.g., has a lang attribute and is in German. Currently we believe that this is a reasonable assumption and one which simplifies our system — allowing the hierarchical relations to continue to apply with being mapped to corresponding higher level relations. However, we are prepared to reevaluate this assumption as we accumulate more practical experience translating documents and calculating licensed inferences.

mapping rules

These rules associate objects and properties in the propagation sentences with objects and properties in the application domain. Like the property rules, mapping rules express generalizations rather than specific claims about document instances.

Examples of mapping rules:

- If *x* is an article and *y* is a name-of-an-author and *x* is the parent of *y*, and *z* is denoted by *y*, then *z* authored *x*.
- If *x* is a meeting-name and *y* is an attendee-name, and *w* is denoted by *x* and *z* is denoted by *y*, then *y* attended *x*.

Note that in some cases the consequent of a mapping rule attributes a property to a document component, but in other cases no property is being attributed to the document component (at least directly), but only to the object that the component denotes.

application sentences

These sentences are typically (but not necessarily) about objects and properties external to the document, such as authors, customers, prices.

They result from applying a mapping rule to one or more propagation sentences.

Examples of application sentences are:

- *a* is the author of (the article) *b*.
- *c* is a meeting.
- *a* attended *c*.

fundamental axioms — Axioms of the sort common to systems supporting commonsense reasoning. Examples:

- If *x* occurs before *y*, then *y* does not occur before *x*.
- If *x* occurs at the same time as *y* then *y* occurs at the same time as *x*.
- If *x* is a part of *y* and *y* is a part of *z*, then *x* is a part of *z*.
- if *p* is necessary and (if *p* then *q*) is necessary, then *q* is necessary.

Further discussion of the fundamental axioms is outside the scope of this paper.

world knowledge — These are sentences about the world or some part of it. They may be particular fundamental facts (water is a material substance, five is a number) particular contingent facts (the United States signed the Berne treaty), or they may be rules — in practical systems, business rules will be of particular importance here. Strictly speaking, these sentences, like the fundamental axioms, are also outside the scope of this paper, but they are a necessary part of the target system: without them, it is unlikely that many useful applications will be built.[3]

Some applications may not need to have all of these kinds of sentences present in the logical system. In some cases, the main value will lie in the application sentences and the theorems which will follow from combining them with the fundamental axioms and world knowledge. The image and propagation sentences will be present only in order to help in generating all the appropriate application sentences. In some cases, the image sentences will serve no purpose at all except the generation of the propagation sentences.

2.4 *Two approaches: inference and direct instantiation*

2.4.1 Inference-driven approach

We are exploring two approaches to the task of generating useful sets of inferences. The *inference-driven approach*, illustrated in figure1, uses all the different kinds of sentences outlined above. We will represent the description of the markup vocabulary and input document directly in the reasoning system (in the form of property rules, mapping rules, and image sentences) and use normal inference techniques to generate all the other sets of sentences. As shown in the figure, image sentences are derived from an XML document instance; property rules and mapping rules are derived from a formal tag set definition. From the image sentences and the property rules, an inference process derives the propagation sentences. From the propagation sentences and the mapping rules, an inference process derives the application sentences. From the application sentences, fundamental axioms, and world knowledge, an inference process derives further sentences.

2.4.2 Direct-instantiation approach

In the other approach, application sentences are generated directly by instantiating (filling in the blanks in) the sentence schemata or *skeleton sentences* described in the framework proposal of [Sperberg-McQueen et al. 2001a]. A system overview is given in figure 2. In this approach, XSLT

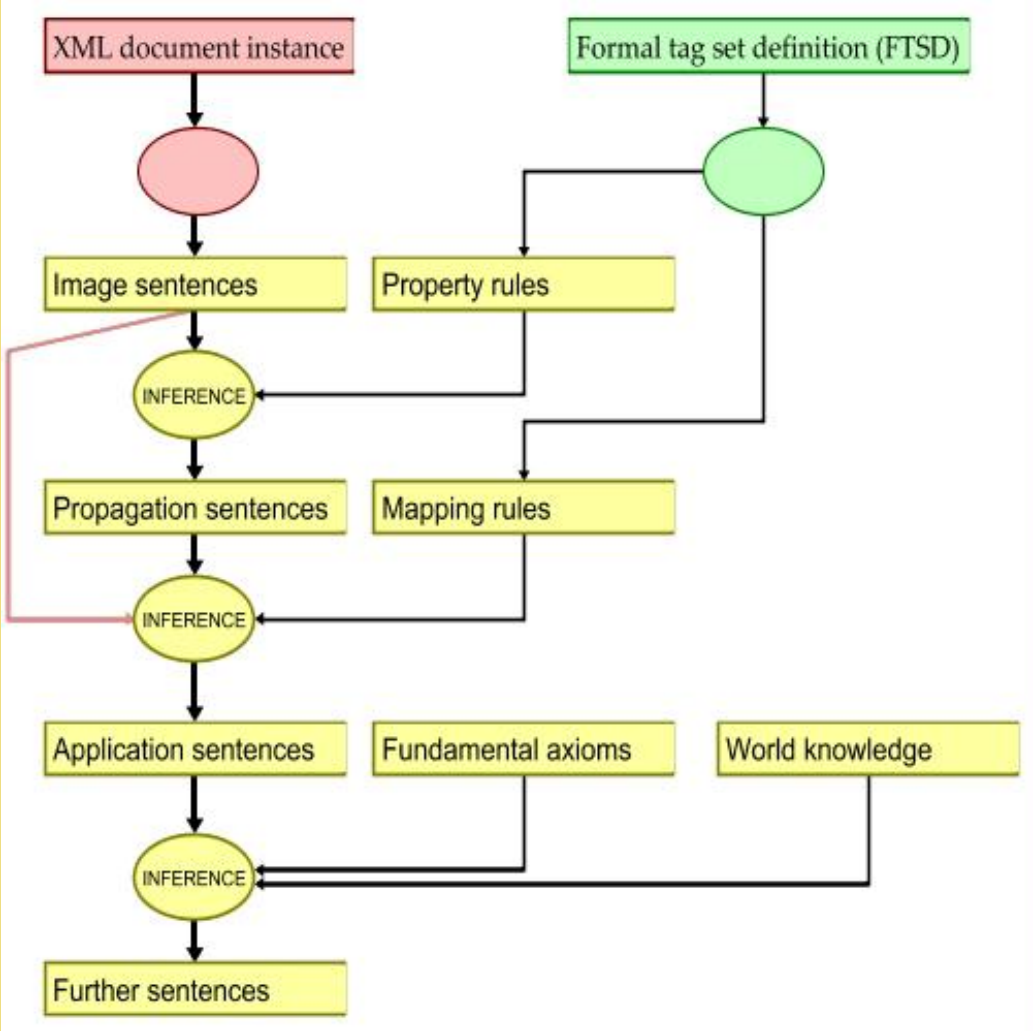


Figure 1 Data flow in the inference-driven approach

transformations are used to map directly from the XML document instance into application sentences.[4] The XSLT stylesheets are specific to particular document types, and they directly embody the property and mapping rules for that document type. This is not, however, the most compact or intuitive form in which to represent the property and mapping rules; in order to make the mapping and property rules accessible for human inspection and for reuse, however, they are formulated in declarative form in the formal tag set definition, and a second-level transformation is applied to generate the appropriate first-level transformation sheets which take a document instance as input and produce application sentences as output.

If it is desirable for auditing or other purposes, an instantiation-based system can also be used to generate image sentences, propagation sentences, and explicit property and mapping rules. An augmented data flow for such a system is shown in figure 3.

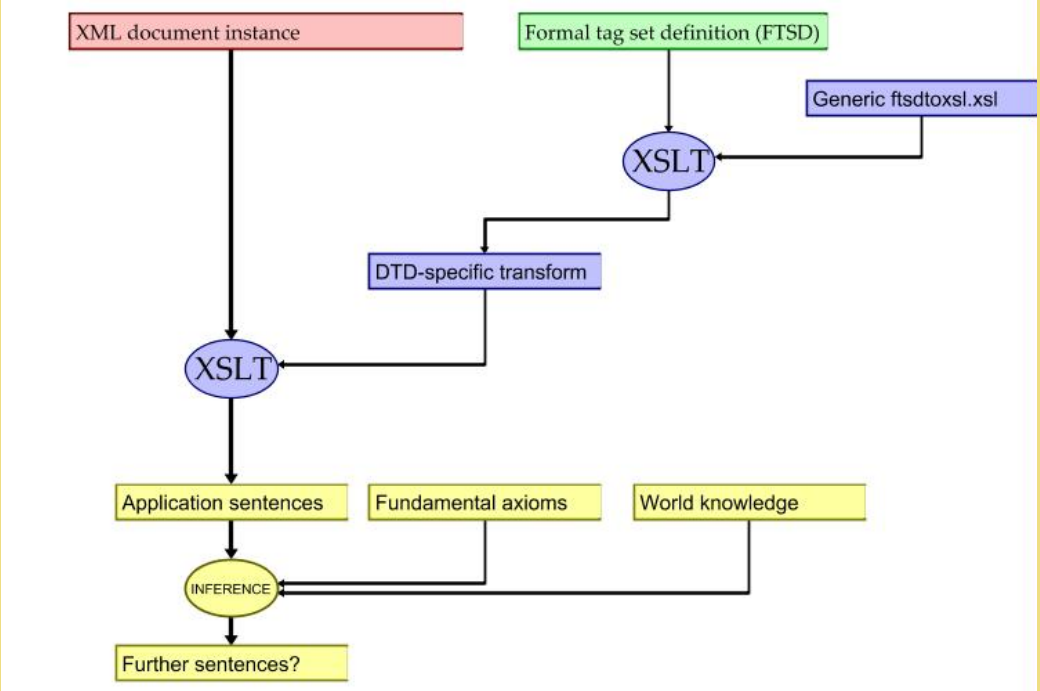


Figure 2 Data flow in the direct-instantiation approach

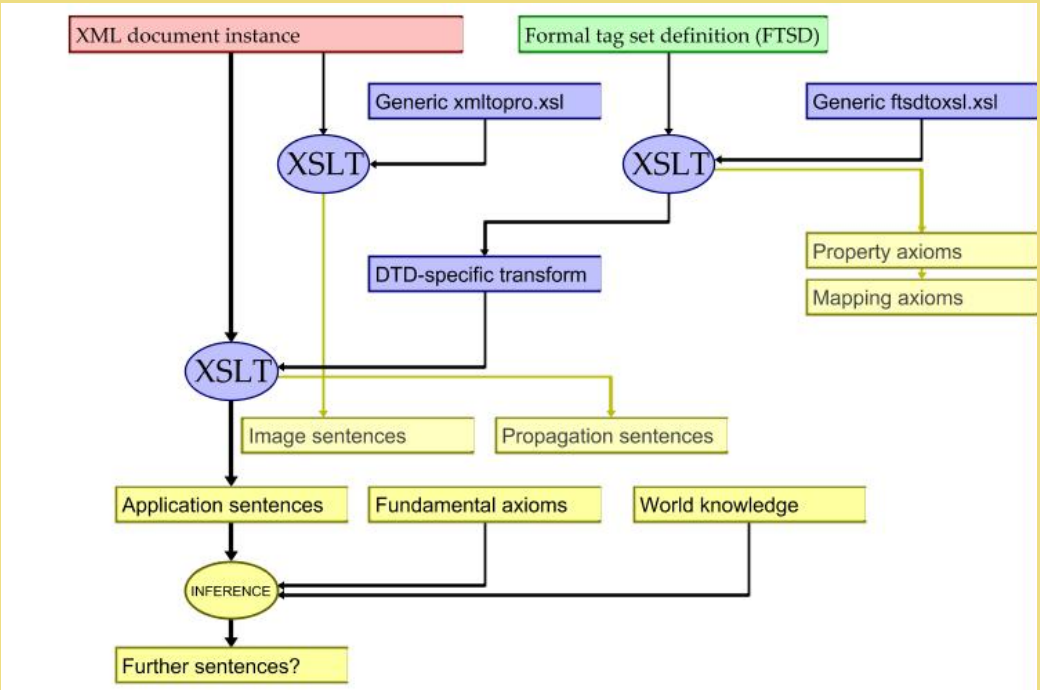


Figure 3 Data flow in the direct-instantiation approach, augmented

§ 3 Document representation

The *image sentences* described above provide a representation of the input document with sufficient detail to enable the original document to be recreated — or, more precisely, a document sufficiently similar to the original to have the same basic information set [Cowan/Tobin 2001]).

In this section, we describe our Prolog representation of image sentences. In earlier work [Sperberg-McQueen et al. 2001a], we described a different Prolog representation, but it is inefficient and exposes somewhat more of the internal implementation data structures than is desirable. The representation outlined here is based on several simple principles:

- The lower level implementation choices are encapsulated as thoroughly as possible. References to data structures are hidden behind Prolog variables, thus insulating the higher level predicates from changes we may later make at the lower levels.
- Identifiable entities, such as nodes, documents, or paragraphs are treated as objects with identities. They are referenced by unique identifiers, rather than by address or location.
- The W3C Document Object Model (DOM) includes methods for node navigation and attribute retrieval that can simplify our definitions. We emulate these methods in Prolog, though ours is not a full DOM implementation.

Less crucial from some points of view, but worth mentioning, is the fact that the Prolog implementation we use (SWI Prolog) provides an SGML/XML parser, which we call and whose results we use to create our representation. To allow portability to other Prolog implementations, none of the parser-specific predicates are essential to the basic system architecture, nor are they intimately coupled with the portable code. All functions performed by the parser could be handled by XSLT transformations that would write the image predicates as output. We have also defined XML serialization routines which operate on the predicates described below, so that the reasoning system can read and write XML directly.

We store facts about the XML tree in image sentences using predicates like the following:

```
node(node6).
gi(node6, quote).
parent(node6, node4).
first_child(node6, node7).
nsib(node6, node10).
attv(node6, lang, de).
content(node9, 'Die Welt ist alles, was der Fall ist.' ).
```

These might be paraphrased roughly as follows:

- **node(node6).** The object called *node6* is a node in the tree.
- **gi(node6, quote).** The generic identifier (sometimes called the *element type*) of *node6* is *quote*.
- **parent(node6, node4).** The parent of *node6* is *node4*.
- **first_child(node6, node7).** The first child of *node6* is *node7*.
- **nsib(node6, node10).** The following sibling of *node6* is *node10*.
- **attv(node6, lang, de).** *node6* has an attribute called *lang*, which has the value **de**.
- **content(node9, 'Die Welt ist alles, was der Fall ist.').** The content of *node9* is the string “Die Welt ist alles, was der Fall ist.”

The identifiers for nodes are generated automatically and carry no significance. The first few nodes of the purchase order example might look something like this:

```
node(node1).
gi(node1,purchaseOrder).
first_child(node1, node2).

attv(node1, orderDate, '1999-10-20').
node(node2).
gi(node2,shipTo).
parent(node2, node1).
first_child(node2, node3).
nsib(node2, node11).

attv(node2, country, 'US').
node(node3).
gi(node3,name).
parent(node3, node2).
nsib(node3, node5).

content(node4,'Alice Smith').
parent(node4, node3).
node(node5).
gi(node5,street).
parent(node5, node2).
nsib(node5, node7).

content(node6,'123 Maple Street').
parent(node6, node5).
node(node7).
gi(node7,city).
parent(node7, node2).
nsib(node7, node9).

content(node8,'Mill Valley').
parent(node8, node7).
node(node9).
gi(node9,state).
parent(node9, node2).
nsib(node9, node10).
```

From Prolog atoms of this kind, it is possible to derive other convenient predicates, such as:

```
children(node6, [node7, node8]).
attl(node11, [purpose='title', lang='la']).
psib(node10, node6).
child(node4, node6).
ancestor(node6, node1).
nearest_anc(node6, p, node4).
nearest_anc(node9, lang, 'de', node6).
```

These might be paraphrased as follows:

- **children(node6, [node7, node8]).** The object called *node6* has as children *node7* and *node8* in that order.
- **attl(node11, [purpose='title', lang='la']).** Node *node11* has attribute-value specifications for attributes *purpose* and *title*, with the values **title** and **la** respectively.
- **psib(node10, node6).** The previous sibling of *node10* is *node6*.
- **child(node4, node6).** *node6* is a child of *node4*. This is the inverse of the *parent* relation.
- **ancestor(node6, node1).** *node1* is an ancestor of *node6*.
- **nearest_anc(node6, p, node4).** The nearest ancestor of *node6* which has the generic identifier *p* is *node4*.
- **nearest_anc(node9, lang, 'de', node6).** The nearest ancestor of *node9* which has the value **de** for the attribute *lang* is *node6*.

§ 4 Propagation sentences

The sample purchase order has no attributes with inherited values, or with complex rules for calculating default values; the propagation-sentence layer of our system, which is intended to handle such cases, therefore has nothing much to do and can be passed over without comment.

§ 5 Application sentences and further inferences

Expressed in English, the markup of the example purchase order allows us to infer propositions in the application domain like the following:

- There exists an object (in this case an abstract object) of the type purchase order (which for convenience we will refer to by the name P-123).
- Purchase order P-123 was placed on 20 October 1999.
- There is someone named Alice Smith now able to receive mail at 123 Maple Street, Mill Valley, California. (For brevity's sake, we will refer to her as S-45 and to her address as A-45.)
- There is someone named Robert Smith now able to receive mail at 8 Oak Avenue, Old Town, Pennsylvania. (For brevity we will refer to him as S-46 and regard him as a customer.)
- The items on P-123 should be shipped to person S-45 at address A-45.
- Customer S-46 is to be billed for the items on purchase order P-123 at the prices indicated.
- Purchase order P-123 includes one (1) item of part number 872-AA.
- The item denoted by part number 872-AA is a lawnmower.
- Customer S-46 is to be billed $148.95 for the item denoted by part number 872-AA.
- Purchase order *P-123* includes one (1) item with part number 926-AA.
- The item denoted by part number 926-AA is a baby monitor.
- Etc.

In the typology given above, these propositions would all be expressed by *application sentences*.

We express application sentences in Prolog using a simple object-oriented language which defines objects and classes of objects, properties of objects and their values, and relations among objects, using the following predicates:

- **object(O)**: *O* has been instantiated as an object.
- **obj_class(O,C)**: Object *O* is of class *C*.
- **models(O,L)**: The real-world object modeled by *O* is represented at the syntax level by the XML elements which are the pre-images of the nodes in *L*.
- **class(C)**: *C* is a class of objects.
- **subclass(Sub,Super)**: Class *Sub* is a subclass of class *Super*. Subclasses can take the same properties and participate in the same relations as their Superclasses.
- **property_of(C,P,T)**: Objects of class *C* have property *P*, which is of type *T*.
- **opv(O,P,V)** Object *O* has value *V* on property *P*.
- **relation(R,L)** Relation *R* can hold among objects of classes listed in ordered list *L*.
- **relation_applies(R,L)** Relation *R* applies to the objects in the ordered list *L*.

Expressed in Prolog using these predicates, the application sentences listed above would take something like the form outlined below.

The vocabulary makes use of the following classes, properties, and relations; we assume here the simple types of XML Schema, but the specifics of the simple type system are not relevant to our argument.[5] The purchase order has a date property, and relations with ship-to and bill-to addresses, comments, and items.

```
class(purchase_order).
property_of(purchase_order,date,'xsd:date').
relation(shipto,[purchase_order,address]).
relation(billto,[purchase_order,address]).
relation(has_comment,[purchase_order,comment]).
relation(has_item,[purchase_order,item]).
```

The actual purchase order in hand allows us to infer the existence of an instance of class *purchase_order*; we can assert its existence by giving it an arbitrary identifier and stating the values of its properties, and asserting its occurrence in members of various relations:

```
object(p123).
obj_class(p123,purchase_order).
models(p123,[node1]).
opv(p123,date,'1999-10-20').
relation_applies(shipto,[p123,a45]).
relation_applies(billto,[p123,a46]).
relation_applies(has_comment,[p123,c926]).
relation_applies(has_item,[p123,p123_i01]).
relation_applies(has_item,[p123,p123_i02]).
```

We call out persons as a special class, because (let us say) we know that they are important for the application area; a more purely mechanical translation from the schema might not define a separate class for persons.

```
class(person).
property_of(person,name,'xsd:string').
```

The two addresses each allow us to infer (for application purposes, at least) the existence of a person at that address:

```
object(s45).
object(s46).
obj_class(s45,person).
obj_class(s46,person).
opv(s45,name,"Alice Smith").
opv(s46,name,"Robert Smith").
```

Addresses have a number of simple properties, mostly string-valued. To illustrate the subclass predicates in our system, we have followed the XML Schema primer's international purchase order example in defining separate types for US and UK addresses. Because we have defined *person* as a separate class of objects, we need to use *relation*, not *property_of*, to describe the link between address elements and their first child (the *name* element).

```
class(address).
class(us_address).
class(uk_address).
subclass(us_address,address).
subclass(uk_address,address).
relation(is_at_address,[address,person]).
property_of(address,street,'xsd:string').
property_of(address,city,'xsd:string').
property_of(us_address,state,'xsd:string').
property_of(us_address,zip,'xsd:string').
```

```
property_of(uk_address,postcode,'xsd:string').
property_of(uk_address,exportcode,'xsd:positiveInteger').
```

The two addresses in our sample offer no particular difficulties:

```
object(a45).
obj_class(a45,us_address).
opv(a45,street,"123 Maple Street").
opv(a45,city,"Mill Valley").
opv(a45,state,"CA").
opv(a45,zip,90952).
...
relation_applies(is_at_address,s45,a45).
relation_applies(is_at_address,s46,a46).
```

Items on the purchase order have product name and part number, price, and ship date.

```
class(item).
property_of(item,productname,'xsd:string').
property_of(item,quantity,'xsd:positiveInteger').
property_of(item,us_price,'xsd:decimal').
relation(has_comment,[item,comment]).
property_of(item,shipdate,'xsd:date').
property_of(item,partnum,'xsd:string').
```

The simple structure of the example makes it easier to fill in the required information:

```
object(p123_i01).
obj_class(p123_i01,item).
opv(p123_i01,productname,"Lawnmower").
opv(p123_i01,quantity,1).
opv(p123_i01,us_price,148.95).
relation_applies(has_comment,[p123_i01,c926]).
opv(p123_i01,partnum,"872-AA").
...
```

Both purchase orders and items can have arbitrary numbers of comments; to model this one-to-many relation, we make comments a class of objects by themselves.[6]

```
class(comment).
property_of(comment,contents,'xsd:string').
```

The two comment elements in the document are represented straightforwardly:

```
object(c926).
obj_class(c926,comment).
models(c926,[node39]).
opv(c926,contents,"Hurry, my lawn is going wild.").

object(c927).
obj_class(c927,comment).
models(c927,[node55]).
opv(c926,contents,"Confirm this is electric").
```

Several observations may be worth making. First, some of these inferences are redundant and provide no new information. No e-commerce site will rely on incoming purchase orders for the knowledge that part 872-AA is a lawn mower or that one of its prices is $148.95. But the redundancy is intentional and may be useful: if the internal system catalog shows 872-AA as a power drill, there is a contradiction which should be resolved before the purchase order is fulfilled. Similarly, most production systems should know from the data provided by the U.S. Postal Service that the zip code

given in the ship-to address is not valid for Pennsylvania. One step in resolving the problem is to notice the contradiction; the redundant information can help make that happen.

Some other inferences are also possible, if we apply some knowledge of the real world; in our system as currently structured, these are not application sentences but appear in the data flow diagrams as "Further sentences".

- There is a street in Mill Valley called Maple Street.

  The Maple Street in Mill Valley has a house 123 on it.

  And so on. Lots of inferences might be drawn from the addresses alone, but we will ignore most of them here.
- (Depending on how the purchase order was generated, we may be more or less likely to infer:) The item denoted by part number 872-AA was quoted to customer S-46 as having a price of $148.95.
- Because purchase orders are supposed to be placed only in good faith, we might infer that customer S-46 is willing to pay for the items on purchase order P-123 at the prices indicated.
- Because purchase orders are sometimes filed by the person who will receive the goods, it is possible that the act of placing this order was performed by Alice Smith.

  Because purchase orders are sometimes filed by the person who will pay for the goods, e.g., because they are giving someone a gift, it is possible that it is Robert Smith who placed the order.
- From the comment "Hurry, my lawn is going wild!" a human or a computer system with a good grasp of English syntax and pragmatics, may well infer that it was Alice Smith, not Robert Smith, who placed the order. This may be an important inference, but since it relies on our understanding of English, rather than on our understanding of the purchase-order markup, it is beyond the scope of systems like those we describe in this paper.
- Because there is no point in shipping goods to people who cannot use them, and since people can use goods only if the people are alive, we may infer that Alice Smith was alive when this order was placed.[7]
- A similar argument may lead us to infer that Robert Smith was probably alive when the order was placed.

There are some inferences we may be tempted to draw, and which may in fact be true in the common case, which are probably not, strictly speaking, licensed by the purchase order or its markup. We mark these tempting but not necessarily valid inferences with a star.

* Person S-45 is desirous of receiving items 872-AA and 926-AA.

* Person S-46 has expressed a willingness to pay for the items on purchase order P-123.

The first inference, though not strictly justified by the markup, may be true often enough that we may use S-45's address for mailings about lawn-care products.

The second inference, although it goes beyond the meaning of the markup as documented by its creator, may nonetheless be licensed by the knowledge at our disposal. If our business rules require some indication of willingness to pay the bill whenever the ship-to and bill-to addresses are not the same, then we may indeed be able to draw this inference — if customer S-46 had not expressed a willingness to pay, then the purchase order would not exist in this form at this point in the processing path. That is, the inferences we can draw from the markup itself may interact with general rules already present in our logical system (the *world knowledge* mentioned above) and generate further inferences. Such inferences are beyond the scope of the system we are describing, though clearly

for some purposes the prospect of drawing such further inferences will be the main reason for wishing to generate application sentences in a logical notation.

§ 6 Skeleton sentences and mapping rules

One immediate problem we face is the development of a notation (specifically, an SGML/XML DTD) for expressing what [Sperberg-McQueen et al. 2001a] call "sentence skeletons", or "skeleton sentences". These are sentences, or rather fragments of sentences, either in English or some other natural language or in some formal notation, for expressing the meaning of constructs in a markup language. They are called skeleton sentences, rather than full sentences, because they have blanks at various key locations; a system for automatic interpretation of marked up documents will generate actual sentences by filling in the blanks in the skeleton sentences with appropriate values from the documents themselves.

Some of the skeleton sentences for our sample inferences in English might look like this, if we used parenthetical expressions to show how the blanks are to be filled in:

- Purchase order _____ (p.o. ID) was placed on _____ (date).
- There is someone named _____ (name) now able to receive mail at _____ (address).
- There is a street in _____ (city) called _____ (street).
- The items on _____ (p.o. ID) should be shipped to person _____ (ship-to-name) at address A-45.
- Customer _____ (bill-to-name) is willing to pay for the items on purchase order _____ (p.o. ID) at the prices indicated.
- Purchase order _____ (p.o. ID) includes _____ (quantity) item of part number _____ (part number).
- The item denoted by part number _____ (part number) is a _____ (productName).
- Customer _____ (bill-to-name) is willing to pay _____ (price) for the item denoted by part number _____ (part number).
- The item denoted by part number _____ (part number) was quoted to customer _____ (bill-to-ID) as having a price of _____ (price).

The skeleton sentences for our Prolog facts would look like this, if we use comments and XPath expressions to say how to fill in the blanks. First, the purchase order; the XPath expressions are to be interpreted as if the *purchaseOrder* element were the current node:

```
object(X),                    /* X is an arbitrary ID */
obj_class(X,purchase_order),
models(X,Y),                  /* Y is generate-id(.) */
opv(X,date,Z).                /* Z is string(./@date) */
```

Next, the person and address items; here, the *shipTo* or *billTo* element provides the current node for the XPath expressions:

```
object(P),
object(A),
obj_class(P,person),
obj_class(A,address),
opv(P,name,N),                /* N is string(./name) */
relation_applies(is_at_address,P,A),
opv(A,street,S),               /* S is string(./street) */
opv(A,city,C),                 /* C is string(./city) */
opv(A,state,T),                /* T is string(./state) */
opv(A,zip,Z).                  /* Z is number(./zip) */
```

Note that if the current node has `country="US"`, then we should not assert **obj_class(A,address)**, but instead **obj_class(A,usaddress)**, and similarly for `country="UK"` and **obj_class(A,ukaddress)**. Similar skeleton sentences can be constructed along similar lines.

In existing markup systems, as in the imaginary vocabulary being used in the purchase order example, the appropriate values for variables are often to be taken from whatever occurs in the document at a specific location. In the TEI DTD, for example, the current page number for the default pagination is given by the value of the '*n*' attribute on the most recent '*pb*' element, while the identifier for the language of any natural-language material is given by the value of the '*lang*' attribute on the smallest enclosing element which actually has a value for the '*lang*' attribute; the language itself is described by whatever '*language*' element in the TEI header has that identifier as the value of its '*id*' attribute.

In the skeleton sentence, we need to label each blank with some expression which describes unambiguously how to derive the appropriate value to be used in the sentences constructed from this skeleton. The parenthetical notes and comments used in the examples above illustrate the point; in real systems we need a more formal way to provide the information. Since these expressions typically "point" to other nearby markup structures, we refer to them as "deictic expressions"; they express notions like "the contents of this element" or "the value of the 'lang' attribute on the nearest ancestor which has such a value" or "the value of the '*type*' attribute on the nearest ancestor of type '*div*'".

Several theoretical and practical problems arise in using skeleton sentences to say what the markup in some commonly used DTDs actually means, in a way that allows software to generate the correct inferences from the markup and to exploit the information.

First, what formalism should be used to write the skeleton sentences? We can easily adopt some existing formalism, e.g., that of Prolog or whatever inference engine we choose to use; can we devise a formalism that will not commit us to a particular inference engine?

There appears to be a significant difference among (a) skeleton sentences which serve to formulate in some formal notation the specific facts expressed by the markup in the document (the *mapping rules* described above), (b) sentences or skeleton sentences which express invariant rules about specific properties captured by the markup, e.g., "The value of the '*lang*' attribute is inherited; the value of the '*n*' attribute is not inherited" (the *property rules*) and (c) sentences or skeleton sentences which express invariant rules about textual and other constructs, e.g., "The author of a letter is physically located at the place given in the place-date line, on the date given in the place-date line, unless the letter is falsified or forged in some way" (which were described above as *world knowledge*). Sentences in group (b) serve to capture useful generalizations about the way markup constructs in a given DTD behave; sentences in group (c) are important for certain kinds of inferences, but appear to have relatively little to do with the markup itself. What is the best way to reflect these differences in function among the sentences and skeleton sentences of a markup system?

One experimental syntax for skeleton sentences uses Prolog notation for the structure of the sentences, a small XML vocabulary for the framework and for filling the blanks, and XPath as the language for the deictic expressions. Some of the skeleton sentences above would look like this in this syntax:

```
<var name="A" val="concat('o-',generate-id(.))"/>
<var name="P" val="concat('o-',generate-id(./name))"/>
<rule>object(<val var="P"/>).</rule>
<rule>object(<val var="A"/>).</rule>
<rule>obj_class(<val var="A"/>,address).</rule>
<rule>obj_class(<val var="P"/>,person).</rule>
<rule>relation_applies(is_at_address,
                       [<val var="A"/>,<val var="P"/>]).</rule>
<rule>opv(<val var="P"/>, name,  <de xv="string(./name)"/>).</rule>
<rule>opv(<val var="A"/>, street,<de xv="string(./street)"/>).</rule>
<rule>opv(<val var="A"/>, city,  <de xv="string(./city)"/>).</rule>
```

Each skeleton sentence is tagged as a *rule* element, the contents of which follow Prolog notation. Deictic expressions within the rule are marked by *de* elements, which carry an attribute whose value is an XPath expression. Frequently used expressions can be assigned to variables, for compactness and clarity. When the skeleton sentences are applied to the document instance in order to produce sentences in the notation of the logical system, a current element is selected and the XPath expressions are evaluated in that context. Their values are written out as part of a concrete sentence which has the same overall form as the skeleton sentence, but has no blanks.

The experimental syntax allows the creator of the skeleton sentences to specify the node from whose context the XPath expressions should be evaluated, by embedding the skeleton sentences within larger structures, which themselves identify element types by means of XSLT match patterns. A larger fragment of the documentation for the fictional purchase order language looks like the following example; the rules defined do not generate the Prolog shown above but a different set of Prolog application sentences. Each set of rules is embedded in an *elemtype* element which describes the element type to which they apply.

```
<elemtype match="purchaseOrder">
 <doc>
  <para>This XML element represents a purchase order.</para>
 </doc>
 <rule distributed="false" lang="Prolog">
  purchase-order(<de xv="generate-id(.)"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  dated(<de xv="generate-id(.)"/>, <de xv="@orderDate"/>).
 </rule>
</elemtype>

<elemtype match="shipTo">
 <doc>
  <para>The person and address to whom to ship.</para>
 </doc>
 <rule distributed="false" lang="Prolog">
  person(<de xv="generate-id(.)"/>, <de xv="name"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  address(a-<de xv="generate-id(.)"/>,
  <de xv="street"/>,
  <de xv="city"/>,
  <de xv="state"/>,
  <de xv="zip"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  person-address(<de xv="generate-id(.)"/>,
    a-<de xv="generate-id(.)"/>).
 </rule>
 <rule distributed="false" lang="Prolog">
  ship-to(<de xv="generate-id(..)"/>,
  <de xv="generate-id(.)"/>,
  a-<de xv="generate-id(.)"/>).
 </rule>
</elemtype>
```

§ 7 Some challenges

In our attempts to formulate skeleton sentences for existing real-world vocabularies like HTML, TEI, and Docbook, some questions have arisen which have proven difficult to answer.

In skeleton sentences like “[this element] is in English”, what do the deictic expressions like “this element” actually refer to? In some cases the inferences appear to relate to the linguistic components of the text itself (“This document is written in English”), and in some cases to the text's formal properties (“Augustine's *Confessions* is divided into 13 books”). In some cases, the markup appears to license inferences about some object or entity in the real world (“Henry Laurens was in Charleston

on 18 August 1775"), but sometimes the entities referred to are not at all in the real world ("Harry Potter missed the Hogwarts Express on 1 September"). In still other cases, the inferences appear to apply to the electronic encoding of the text itself ("This metadata was last revised on 20 July 1998") or to some other witness to the same text ("The recipient's copy of this letter is preserved in [some particular archival collection, with some particular call number]"). Attempting to disentangle these lands the would-be formulator of skeleton sentences promptly in a thicket of ontological questions which have not yet received adequate attention.

The ontological questions become even more thorny in connection with markup systems like that of the Text Encoding Initiative, which are intended for use by a wide variety of projects which are expected to have widely different views about the ontological commitments to be associated with the TEI markup. Do statements about Augustine's *Confessions*, for example, relate to some abstract text distinct from each physical copy of the text, or is the phrase "Augustine's *Confessions*" merely a convenient shorthand for "all the physical documents which witness Augustine's *Confessions*"? It would appear essential to decide this question in order to formulate skeleton sentences for markup languages like the TEI, but the TEI itself is intentionally coy about the issue, in order to ensure that textual Platonists and textual constructivists can both use TEI markup. It is a challenge to build a similar ambiguity or vagueness into the set of skeleton sentences which document the prescribed interpretation of TEI markup.

Notes

1. In this we follow a proposal made by Turski and Maibaum in their discussion of programming-language semantics [Turski/Maibaum 1987]; they put the proposal thus (p. 4):

   "Two points deserve special attention: we expect programs to be capable of expressing a meaning and we want to be able to compare meanings. Unless we are very careful, we may very soon be forced to consider an endless chain of questions: what is the meaning of ...? what is the meaning of the meaning of ...? etc. Without going into a philosophical discussion of issues certainly transgressing any reasonable interpretation of the title of this book, we shall accept that *the meaning of A is the set of sentences S true because of A*. The set *S* may also be called the set of consequences of *A*. Calling sentences of *S* consequences of *A* underscores the fact that there is an underlying logic which allows one to deduce that a sentence is a consequence of *A*.

   "Usually the *A* itself is a set of sentences, thus we are saying in fact that the meaning of a set of sentences is the set of consequences that are deducible from it."

2. The term *deictic expression* comes from traditional grammar, where it is used to denote pointing expressions like "this one over here" or "that one over there" — from the Greek word *deixis*, for pointing.

3. The history of work in artificial intelligence includes many examples of knowledge bases which capture the kind of information we believe will go here, and attempt to show how to build useful applications using it. Recent relevant work includes [Fikes/McGuinness 2001].

4. Obviously, XQuery functions can also be used.

5. The simple types assigned are more or less those assigned in the XML Schema primer; we have not attempted to model the USState type here, which is a restriction of *xsd:string* with an enumerated set of values. We have retained the type *xsd:positiveInteger* for zipcode; we are strongly tempted to change it to *xsd:string* for the sake of verisimilitude, because

leading zeroes are not omissible in zip codes, but we have decided to follow the schema primer here.

6. We could allow many-valued properties, but prefer a more strictly relational approach.

7. Actually, if the order was placed by Robert Smith, it is not entirely safe to infer that Alice Smith was alive; it is plausible, however, that Robert Smith thought she was alive. Unless, of course, we are reasoning about events in a detective story, in which case it may only be the case that Robert Smith wished to give the *impression* that he thought Alice Smith was alive.

Bibliography

[ACH/ACL/ALLC 1994] Association for Computers and the Humanities, Association for Computational Linguistics, and Association for Literary and Linguistic Computing. 1994. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 1994.

[Coombs et al. 1987] Coombs, J. H., Renear, A. H., and DeRose, S. J. 1987. Markup systems and the future of scholarly text processing. *Communications of the Association for Computing Machinery 30*, 11 (1987), 933–947.

[Cowan/Tobin 2001] Cowan, John, and Richard Tobin, ed. 2001. "XML Information Set." W3C Recommendation 24 October 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/xml-infoset/

[Fallside 2001] Fallside, David C. 2001. XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/xmlschema-0/

[Fikes/McGuinness 2001] Fikes, Richard, and Deborah L. McGuinness. 2001. "An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL (March 2001)." W3C Note 18 December 2001. [Cambridge, Sophia-Antipolis, Tokyo]: World Wide Web Consortium. http://www.w3.org/TR/daml+oil-axioms

[Guttag/Horning 1993] Guttag, John V., and James J. Horning, with S. J. Garland et al. 1993. *Larch: languages and tools for formal specification*. New York: Springer-Verlag. Texts and monographs in computer science 981.

[Hughes/Cresswell 1968] Hughes, G. E., and M. J. Cresswell. 1968. *An introduction to modal logic*. London: Methuen.

[ISO 1986] International Organization for Standardization (ISO). 1986. *ISO 8879-1986 (E). Information processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*. International Organization for Standardization, Geneva, 1986.

[Ramalho et al. 1999] Ramalho, José Carlos, Jorge Gustavo Rocha, José João Almeida, and Pedro Henriques. 1999. "SGML documents: Where does quality go?" *Markup Languages: Theory & Practice* 1.1 (1999): 75-90.

[Renear 2001] Renear, A. 2001. Raising the bar: Text encoding from a logical point of view. CLIP 2001: Computers, Literature, Philology, Gerhard-Mercator University, Duisburg, Germany, December 2001.

[Rowe 1988] Rowe, N. C. 1988. *Artificial Intelligence through Prolog*. Prentice Hall, Englewood Cliffs, NJ.

[Schatz et al. 1996] Schatz, B., Mischo, W. H., Cole, T. W., Hardin, J. B., Bishop, A. P., and Chen, H. 1996. Federating diverse collections of scientific literature. *Computer 29* (May 1996), 28–36.

[Simons 1997] Simons, Gary F. 1997. "Conceptual Modeling versus Visual Modeling: A Technological Key to Building Consensus." *CHum* 30.4: 303-319.

[Simons 1999] Simons, Gary F. 1999. "Using Architectural Forms to Map TEI Data into an Object-Oriented Database." *CHum* 33.1-2: 85-101. Originally delivered in 1997 at the TEI 10 conference in Providence, R.I.

[Sperberg-McQueen 1991] Sperberg-McQueen, C. M. 1991. "Text in the Electronic Age: Textual Study and Text Encoding, with Examples from Medieval Texts." Literary and Linguistic Computing, 6:1, 34-46.

[Sperberg-McQueen et al. 2001a] Sperberg-McQueen, C. M., Claus Huitfeldt, and Allen Renear. 2001. "Meaning and interpretation of markup." *Markup Languages: Theory & Practice* 2.3 (2001): 215-234. http://www.w3.org/People/cmsmcq/2000/mim.html

[Sperberg-McQueen et al. 2001b] Sperberg-McQueen, C. M., Claus Huitfeldt, and Allen Renear. 2001. "Practical extraction of meaning from markup." Paper given at ACH/ALLC 2001, New York, June 2001. (Slides at http://www.w3.org/People/cmsmcq/2001/achallc2001/achallc2001.slides.html)

[Swick/Thompson 1999] Swick, Ralph R., and Henry S. Thompson, ed. 1999. *The Cambridge Communiqué*. W3C NOTE 7 October 1999. http://www.w3.org/TR/schema-arch

[Thompson 2001] Thompson, Henry S. 2001. "Normal Form Conventions for XML Representations of Structured Data". Talk at XML 2001, Orlando, December 2001. http://www.ltg.ed.ac.uk/~ht/normalForms.html

[Turski/Maibaum 1987] Turski, Wladyslaw M., and Thomas S. E. Maibaum. 1987. *The specification of computer programs*. Wokingham: Addison-Wesley.

[Vorthmann/Robie 2001] Vorthmann, Scott, and Jonathan Robie. 2001. "Beyond schemas: Schema adjuncts and the outside world". *Markup Languages: Theory & Practice* 2.3: 281-294.

[W3C 2000] W3C (World Wide Web Consortium). 2000. "XHTML 1.0: The Extensible HyperText Markup Language. A Reformulation of HTML 4 in XML 1.0." W3C Recommendation 26 January 2000 [Cambridge, Sophia-Antipolis, Tokyo]: W3C. http://www.w3.org/TR/xhtml1/

[Walsh/Muellner 1999] Walsh, N., and Muellner, L. 1999. *DocBook: The Definitive Guide*. O'Reilly and Associates, Inc., Sebastopol, CA.

[Welty/Ide 1997] Welty, Christopher, and Nancy Ide. 1997. "Using the Right Tools: Enhancing Retrieval from Marked-up Documents." *CHum*y 33 (1999): 59-84. Originally delivered in 1997 at the TEI 10 conference in Providence, R.I.

The Authors

C. M. Sperberg-McQueen
World Wide Web Consortium, MIT Laboratory for Computer Science

C. M. Sperberg-McQueen is the Architecture Domain Lead for the World Wide Web Consortium and a visiting researcher at the University of Bergen.

David Dubin
University of Illinois at Urbana/Champaign, Graduate School of Library and Information Science

David Dubin is a research scientist at UIUC's Information Systems Research Laboratory, working mainly with the Electronic Publishing Research Group. His research interests are in the areas of information retrieval, text and document processing, and classification. He is a member of the Association for Computing Machinery, the American Society for Information Science, and the Classification Society of North America.

Claus Huitfeldt
Avdeling for kultur, språk og informasjonsteknologi, Bergen University Research Foundation

Claus Huitfeldt is an associate professor of philosophy at the University of Bergen and the director of the the University of Bergen Research Foundation's department for research in culture, society, and technology (AKSIS) and of the Humanities Information Technology Research Programme. From 1990 to 1999 he was the director of the Wittgenstein Archives at the University of Bergen, which created an electronic edition of Wittgenstein's posthumous papers.

Allen Renear
University of Illinois at Urbana/Champaign, Graduate School of Library and Information Science

Allen Renear is an Associate Professor in the Graduate School of Library and Information Science. His principal research focus is on how digital documents function as knowledge representation systems.

As a student at Bowdoin College and Brown University, he specialized in epistemic logic and the philosophy of science; after several years teaching philosophy, he joined Brown's Computing and Information Services in 1984, working first as a systems analyst and project leader, and then as a strategic planner. During this time he consulted on or managed many humanities computing projects and became involved in a variety of text encoding and computing activities — including X3V1.TG8, the Text Encoding Initiative (TEI), and the Association for Computers and the Humanities (ACH). In 1988 Renear helped design the Brown Women Writers Project (WWP), serving at various points as WWP Co-Director, Acting Director, and Director, and in 1993 he became founding Director of the Scholarly Technology Group. He is currently the chair of the Open eBook Publication Structure Working Group.

Extreme Markup Languages 2002
Montréal, Québec, August 6-9, 2002
This paper was formatted from XML source via XSL
Mulberry Technologies, Inc., August 2002

Alois Pichler

Transcriptions, Texts and Interpretation

This paper does not deal with Wittgenstein's philosophy, nor does it speak about his Nachlass. Rather, it discusses one aspect of making his Nachlass accessible to machine processing: computer aided transcription, as it is done at the Wittgenstein Archives at the University of Bergen (WAB). This discussion involves questions about text representation and text in general.

The terms "transcribe" and "transcription" are often understood as the process of making - by writing - an exact representation of a document, text, etc. Consequently it may be said that the aim of transcription is to represent the original as correctly as possible. However, I would argue that transcription - in particular transcription as it is done at WAB - is a much more complex process than "to represent the original manuscripts as correctly as possible".

The following discussion will therefore firstly focus on the questions "what is transcription?" and "what is the aim of transcription?". Secondly I will consider in more detail the question of transcription and interpretation. We must recognize that transcription work involves a range of different interpretational activities, which need to be distinguished in order to understand transcription work properly and avoid problems and inconsistencies in this work.

Both parts of my paper serve to illustrate the following general point: Transcribing is not copying, but - as text-editorial work in general - rather selecting and interpreting. Any edition of Wittgenstein is in a strong sense a result of interpretation. Our only option is to formalize interpretation, and to make it explicit.

I.

The aim of transcription has often been defined as "to represent the original manuscripts as correctly as possible". This needs a clarification, therefore let us ask some questions:

What does "as correctly *as possible*" mean? What is the criterion of *correctness*? And what does "to *represent*" mean? To represent in which medium?

I think, the essential question is not about a true representation, but: Whom do we want to serve with our transcriptions? Philosophers? Grammarians? Or graphologists? What is "correct" will depend on the answer to this question. And what we are actually going to represent, and how, is determined by our research interests (philosophical, grammatical, philological, graphological ... in-

terests), and not by a text, which exists independently and which we are going to depict.

In our transcription work at WAB we do not for example distinguish between features of handwriting such as convex and concave "r"s, or the position of the dot over the "i". We do not record the endings of a line, and we represent only a few grammatical elements of the text. All this would be possible, but you might say: It is of no relevance to distinguish such features. However, the fact is, we have decided, what is relevant for us and what not. We have made certain decisions about what a transcription should contain, and these are in answer to our idea of whose interests we serve.

Even so, our decisions are not final, or at least, ought not to be. We could, for example, insert all line endings; our reason for not doing this at present is not that it might not be of importance to someone, but rather, that it does not belong to our current interests. This again also has to do with practical questions such as financing, time schedules etc. As you can imagine, making a record of all the line endings is an extremely time consuming task.

The sign of a good transcription system is that it has extensibility. This means that it should be possible to revise and adjust the system to serve new interests as and when they arise. Thus we choose to transcribe certain things at an early stage knowing that we can include other things later if necessary. What we choose to include initially is also determined by what is easier to distinguish while emersed in the text. The insertion of codes for wordclasses, for example, would be a simple task more easily performed afterwards, whereas the distinction between different uses of parentheses is best done during the initial transcription process.

With these considerations in mind I conclude: Our aim in transcription is not to represent as correctly as possible the originals, but rather to prepare from the original text another text so as to serve as accurately as possible certain interests in the text. We do not want to produce a photograph of the original - this is the function of a facsimile. "As correctly as possible" can only mean: "as correctly as possible in relation (in answer) to certain research interests".

What these interests are, with regard to the work done at WAB, I will present in the second part of this paper.

II.

To transcribe a text according to specific interests will require that those interests be clearly served by different conventions, e.g. different codes. The ap-

plication of these conventions presupposes in turn a variety of different interpretational decisions. For example, to identify the string *DIE ABHANDLUNG* as an inscription which bears a meaning requires a different interpretational standpoint and knowledge than is required when encoding it as an intertextual reference to Wittgenstein's "Abhandlung", the Tractatus, or as a variant to *MEIN BUCH*.

What are the different interests - on a macro level - we at WAB focus on when transcribing Wittgenstein's manuscripts?

(a) A fundamental aspect of transcription is that of graphic transcription.

This means that we transcribe "a"s as "a"s, "b"s as "b"s, "c"s as "c"s etc., sections as sections, deleted text as deleted, inserted text as inserted, etc. But already here the selective element becomes obvious, because we do not record that the single handwritten letters look different, that the lines which strike out text can vary significantly etc. A totally faithful graphic transcription is not possible, neither is it desirable.[1]

Speaking of graphic transcription we must make a point which concerns the role of perception. When transcribing, one first tries to grasp the written word as a Gestalt, and not as a sequence of single letters. Very often - in particular when transcribing Wittgenstein's secret code passages - we cannot see what the single letters are until the whole word has been grasped. But this kind of interpretation in the reading of words and single letters is quite different from the interpretation involved in the encoding of a title as a title.

(b) The next transcription activity I want to distinguish is syntactic transcription.

Syntactic transcription has the particular aim of providing for text processing which produces a syntactically well-formed text. In order to meat this requirement of well-formedness we often have to rearrange the text. In the case of an insertion outside regular lines which adds text to the text in line, for example, we will have to include the inserted text in the line. However, doing so, we do not forget about the principle of graphic transcription and, in this case, encode the text in addition as inserted outside regular line (e.g. in the upper margin of the page).

There is a fundamental difference between graphic and syntactic transcription. A graphic transcription of

I ~~AM~~ FEEL FINE

1 I should make it clear that a facsimile does not fall under the notion of "transcription", as it is understood here.

where *AM* is deleted and *FEEL* inserted above would mean the encoding of the deleted text as deleted and the inserted text as inserted, and might look as follows:

I <del/AM> <ins/FEEL> FINE

But to encode the same text syntactically leads to something quite different: Since we probably all understand the inserted word *FEEL* as replacing the deleted word *AM*, we should - according to the requirement of well-formedness - embed the elements in a substitution code which then suggests a certain relation between them, namely the relation of substitution (which might be represented as follows):

I [substlAMl FEEL] FINE

What WAB does is both, graphic and syntactic transcription:

I [substl<del/AM>l <ins/FEEL>] FINE

Another example of syntactic transcription is the following:

I <del/<npc/IF>> LIKE COWS

The deleted word *IF* does not form a part of the syntactic context, therefore it is encoded by "<npc/.../npc>" (= not part of context), a syntactic code. This is in contrast to cases such as

I LIKE ~~BIG~~ COWS

which can be transcribed - purely graphically - as

I LIKE <del/BIG> COWS

The inclusion of *BIG* does not conflict with the requirement of well-formedness, since *BIG* can form a part of the syntactic context and has no replacing function.

However, it is often left to the transcriber to decide whether a certain interlinear string is an addition to the text or rather a substitution.

An analysis of WAB's transcription shows that codes which fall under the "syntactic transcription"-category constitute a large part of the codes used in the transcriptions, which in return means that this type of transcriber's interpretation is highly present in the transcriptions. WAB's transcriptions are therefore much more than an ad literatim transcription: they *do* provide for accurate diplomatic printouts, but they also allow for the possibility of printouts of well-formed texts.

(c) A third type of encoding is normalization of orthography.

If we aim at being able to produce both what we call diplomatic and normalized printouts from the very same transcription, the transcription file has to provide the basis for both; this means in the case of orthographical errors that both the authentic and the normalized versions will be accessible.

(d) A fourth activity includes the application of codes for documentation of the source.

These codes contain information concerning material matters such as size of the original, writing medium, different hands, as well as information about the history of the original, its origin and dates, and references to catalogues. This presupposes knowledge of the source.

(e) A fifth group concerns documentation of the use of the code system in the transcription work and transcribers' explanations and comments on the text, the transcription process and the use of the code system.

Since the code system is a reflection of work in progress which is updated time and again, the specification of the system used will contain important information for any further work with the transcriptions. It might e.g. be that the transcriber has difficulty applying certain codes (following up certain interests) in a certain manuscript. These matters need proper documentation and explanation; they might in themselves lead to further changes in the code system.

This group would also contain codes indicating uncertainty regarding the correctness of the encoding itself, codes for not clearly legible passages, codes for text which cannot be deciphered at all etc. Some text passages might be very difficult or impossible to transcribe: here the transcriber will make a comment which refers the user back to the original.

Sometimes text phenomena need an explanation which the transcriber - since he has worked intimately with the text - might be able to give, and which the user might appreciate. In the case of WAB this does not imply philosophical commentaries on the text, but rather information about such things as particular orthographic habits or the author's use of markers.

(f) Another type of interpretation is again involved in what we call disambiguation: WAB's code system provides possibilities for distinguishing between different functions of the same graph. Therefore the transcriber is required to distinguish whatever should and can be distinguished within practical limits. Parentheses in Wittgenstein's Nachlass, for example, can have quite different functions (besides the conventional use: e.g. suggesting a deletion, indicating a possible substitution etc.); hence parentheses with different functions should be disambiguated and encoded differently.

(g) A final type of encoding serves the retrieval and analysis of various

kinds, such as indexation. These codes concern among other things the registration of compositional features and intertextuality.

What I mean by intertextuality is codes which record internal and external references, names of persons, references to published works, relations to other manuscripts etc., as made by either the author or the transcriber. Such codes allow for hyperlinks which guide you to variants in other manuscripts etc.

Compositional registration implies distinguishing different types of text within the manuscript, e.g. where something functions as a preface to label it as such, and similarly for the author's own miscellany, editorial instructions, titles, content tables etc.

The classification of interpretation types as presented here is by no means exhaustive in relation to text encoding in general. The classification does not for example include a set of codes for grammatical encoding or for subject indexing. Neither are the codes of WAB's transcription standard exhaustive in their particular areas: it would for example be easy to distinguish further graphically between different kinds of deletions.

In order to provide for consistent and smooth transcription work, it is wise to keep these categories as much apart from each other as possible, which means that the single types must be extractable without hereby interfering with other types. From this it follows e.g. that it is necessary to encode variants which at the same time are insertions, both as variants and as insertions - since the graphic level shall be kept apart from the syntactic level.

The conclusion from these considerations about transcription work at WAB is that transcription work is essentially selective and interpretational in nature, moreover, that any text editing work is interpretational work. Editing Wittgenstein's Nachlass in book form presents the same types of interpretational problems as are encountered in preparing an electronic edition. The difference between a book edition and an electronic edition lies, however, in an electronic edition's potential to be able to make the types of interpretation - and their differences - explicit and extractable, to give the user the possibility to choose between the different levels of interpretation, and to realize them in different ways: e.g.: to choose an ad literatim printout rather than a normalized one, but at the same time have text, which was originally underlined, printed in italics. With regard to these demands a machine-readable version has considerable advantages over a book edition.

Alois Pichler
The Wittgenstein Archives
University of Bergen

Harald Hårfagresgt. 31
N-5007 Bergen
Norway

Commentary and Opinion

Beyond the "descriptive vs. procedural" distinction

Wendell Piez
Mulberry Technologies, Inc.
17 West Jefferson St.
Suite 207
Rockville MD 20850
EMAIL wapiez@mulberrytech.com

There has come to be a consensus that the "procedural vs. declarative" distinction is useful, if only as a rough guide, in the design of markup languages. To understand how and why this is the case, we need to ask questions that are usually left unasked when this principle is proposed, such as "is it the model (the schema) that we consider to be descriptive, or the tagged document?" or, more deeply, "why do we validate our markup anyway?"

A number of our fundamental assumptions are not always true. Sometimes a schema might be more than a "go/no-go gauge", becoming a diagnostic and investigatory instrument. Sometimes marked-up documents look backward (as representations of something preexisting), not just forward to processing. Sometimes semantic opacity is a feature, not a bug. In order to understand the power of markup languages, it is helpful to keep in mind that they are both technologies and a species of rhetoric. New characterizations and categories of markup languages may help focus our design efforts.

Why are we still talking about the "descriptive/procedural" distinction? The conception continues to be a focus, because it continues to have explanatory power. Yet at the same time, it clearly demands quite a bit of refinement when we look at the increasingly broad spectrum of different markup languages, and markup language applications, that are now proliferating. Descriptive? Of what? Separation of format from content? What's content, and what's format? Apart from its format, to what do we refer to determine what "content" is; how do we specify it, and how do we go about designing tags for it? What kind of thing are we trying to model, anyway?

The traditional arguments around the "descriptive/procedural" distinction[1] have detailed a number of advantages to descriptive markup languages (also

1 Two "canonical" references I consulted ([Goldfarb 1990] and [Sperberg-McQueen 1994]) make the distinction between "descriptive" and "procedural" approaches. To label the different design strategies "declarative" and "procedural", while observing a distinction with a history in computer science, is evidently problematic in this context, mainly since those terms are so relative. In this paper, as I am deliberately reflecting on the distinction as traditionally rationalized, I'll use the terms "descriptive" and

Wendell Piez, "Beyond the "descriptive vs. procedural" distinction"
Markup Languages: Theory & Practice 3.2 (2001): 141–172

loosely identified as "generic" languages) over their procedural cousins: scalability, reusability of data, and so forth. While these advantages are demonstrably real, nonetheless the evolution of XML technologies, especially in such applications of XML as XSLFO, SVG, SMIL, or even XSLT,[2] shows that the opposite approach to designing a markup language also is playing an important role. Sometimes, it is clear, a procedural language is exactly what we want.

At the Extreme 2000 conference on markup technologies in Montreal, Allen Renear picked up the task of scrutinizing the opposition, proposing an alternative framework for which he introduced terms from linguistics and speech-act theory. I'll return to Renear's argument; but in order to get at these issues at a deeper level, I start by pointing out one begged question, and potential ambiguity, generally at issue when we assert these categories. When we describe a "markup language" (or a "tag set") as descriptive or prescriptive, are we talking about model or instance? That is, are we talking about the proposed, implied or asserted semantics of an abstract model for a document type (classically, as formalized by a DTD); or are we making generalizations about the tags in use, that is the (implied or effective) semantics of element and attribute types as instantiated in documents? It matters which one of these we are describing, not simply because they may be different (in the ideal case they should perhaps not be), but because the very fact that the two things (DTD and document) might possibly end up "meaning" something different in practice, raises questions about the relation between model and instance. In the real world (not to put too fine a point on it), sometimes users "mean" tags in ways not intended by designers, and this fact bears directly on the problem because it indicates how a model's "description" of a document *type* may not be exactly what a document's own tags "describe" (or may be purported to describe, depending on who you talk to).

Now any designer will seek, and will probably assume, that the semantics of model and instance should be the same, or at least not at cross-purposes. When they diverge, we call it "tag abuse", thus begging the question by simply handing authority for correctness to the designer's "intent", whether stated or implied. In fact, since we generally design models first and write instances later, it is a design goal, necessarily implicit and always assumed, that the model be complete and well-fitted enough to the problem domain, that its semantics[3] can be effectively

---

"procedural" when referring to the traditional dichotomy, and occasionally loose synonyms such as "generic" or "presentational" when it serves my purposes (one of which is, of course, to clarify what we might mean when we use these terms).

Another examination of this issue, tracing out several possible "axes" along which distinctions may be made (and thereby anticipating the arguments of Allen Renear [Renear 2000] and myself), is Mavis Cournane's. See [Cournane 1997].

2 Note that in this context, XSLT is procedural as a *markup* language (the tag set is closely bound to a set of processing requirements), while being declarative as a transformation or *processing* language: it is procedurally bound to a declarative application. This in itself is an indication of how relative these terms are.

3 Note here I specifically mean the *implied* semantics of the model, not any behavioral or operational

reflected in instances without strain. But it may also be that to engineer a system in which this ideal may be realized (or *nearly* realized), we had better come to an understanding of how the model and the instance relate to each other not just in theory as an objective, but also in practice, where things always seem to have at least the potential of falling short, and where nothing is so certain as the human capacity to introduce uncertainty through creative adaptation. I will suggest that model and instance need not always relate to each other in the same way; and in fact that the way requirements dictate they must relate to each other in any given application of a markup language, has a direct impact on the suitability of different strategies available to the designer. Since these strategies are commonly framed by distinguishing descriptive vs. prescriptive, declarative vs. procedural, or any of several other oppositions down to "separation of presentation [or format] from content", it is ultimately this distinction that we are illuminating.

To ask how model and instance relate to each other is to ask, in a very general way, about the process and role of what we usually call *validation*, that is the process by which model is applied to instance. (It is not the only such process; but the nature of validation — and usually, its purpose — is such that it can be taken to stand in for others.) So the first thing we need to consider is what validation is and why we do it.

Why it matters what "validation" is

What is "validation"? As soon as asked, it turns out that this is very much a live question. XML and XML-based technologies have lately been serving as an incubator for all kinds of new approaches to validation. Some seek merely to recast inherited notions of validation into new forms (presumably more tractable), some seek to enhance them with capabilities of alternative validation regimens, and some may go in entirely new directions. To say nothing of non-XML approaches (and I hope we see plenty of innovation on this side as well, insofar as there are certainly significant features of texts and interesting problems to which XML does not easily lend itself), in XML we have well-formedness checking, DTD validation, XML Schema, RELAX, TREX, XML-Data Reduced, Schematron, Examplotron, etc. etc.

It is not my concern here to consider these in any detail, or even to distinguish between them, except to point out the interesting (and significant) fact that they do not all take the same thing as their object of examination. Basically, when we validate, we take an instance (an "XML document") and a model (the

semantics that may be actuated in code. As Robin Cover points out, between DTD and the markup constructs as implied by the syntax, SGML/XML is a fairly weak format for specifying the latter [Cover 1998]. We are always free, however, through names, relationships, or explicit documentation, to assert informal "human" semantics: to say, that is, what we think we mean.

"schema" or "specification"), and compare these for purposes of saying whether the instance conforms to the model, or in what ways it fails to conform. But some of these approaches work on an XML document as a text entity (a sequence of alphanumeric characters, some of which constitute data, some of which constitute markup, as per the XML Recommendation [XML 2000]); while others operate on some kind of more complex structure, typically a document object or "infoset" held in memory. An important aspect of this is how the formalization in XML of well-formedness gives us a new platform on which to build and standardize validation techniques. A definition of "well-formed" (as distinct from "valid") brings with it the capability of doing what could be called a "plain parse", rendering a sequence of characters into an abstract information set without otherwise concerning ourselves with the higher-order semantics of elements and attributes. This is important because we may not know or care about such higher-order semantics every time we process. And when we do, testing the conformity of an XML document to any particular semantic profile (however represented) becomes, properly, just one more kind of processing, albeit of a distinctive kind (or to a specific end). Thus validation, considered in light of its purposes and often its methods, is actually closer to querying, say, or to transformation, than it is to parsing as such.

The proliferating approaches to validation also demonstrate that (among other things) any XML document — whether considered as a stream of characters or as an abstract information set — potentially exhibits a range of different features or characteristics which we might be interested in testing:

- Constraints on structure of elements and attributes by type ("a *head* is permitted inside a *chapter*, but a *chapter* is not permitted inside a *head*").
- Conformity of data elements (element content or attribute value) to specific lexical or other requirements: data type integrity; "authority control"
- Referential integrity of links and pointers
- etc. etc.

Any or all of these might be considered to be properly within the realm of validation; and more to the point, the list is as open-ended as we wish it to be.

Validation and workflow: strict validation

The intention or purpose of validation is to subject a document or data set to a test, to determine whether it conforms to a given set of external criteria. Validation may thus be distinguished from processing in general, which may not bother to conduct any such tests (or which may use other tests). It is precisely because the range of features and characteristics in which we are interested, and which we need to be able to constrain, is so open-ended, that testing becomes a useful thing to do in practice. (If it weren't, our tools could all be built so as not to need tests.) Our need to test is simply explained and understood (so much so that it

rarely needs to be explicated): if there exists a point in a process where it is less expensive to discover and correct problems than it is to save the work of testing and fix at later points, it is profitable to introduce a test. The ideal workflow, that is, is one in which we make any correction or adjustment to the materials being processed at the point where it is easiest and least expensive, making allowances for the expense of running tests. This assumes, of course, a workflow that is sufficiently defined to make this possible.

We validate, that is, because we want to know *in advance* of something whether our data set conforms to a set of specified requirements. Notice a key concept we have introduced here: such an operation only makes sense, and only becomes necessary, in an articulated workflow. Validation, that is, is a type of "quality assurance" applied at a particular stage in processing. We need downstream processing to be predictable, and wish to engineer away, to whatever extent we can, any possibility of having to decide how to resolve or render any given anomaly (however interesting it may be) at a later stage of processing. Rather, we want to invest energy now in assuring that our data already conforms to a set of clearly-understood criteria.

In fact, this is nothing more than the application of a simple rule of industrial engineering, here applied to information systems. In effect, we are designing a process (even if a simple one) — an assembly line. Validation provides us with what is called a "go/no-go" gauge.

This is not merely an analogy. If we look at the beginnings of mass production technologies, we find a significant transition occurs in the nineteenth century with the development of the"American System of Manufacture".[4] What distinguished this approach to mass production from previous efforts is that the ancient principle of division of labor was joined with a new one: making the component parts of the product to be interchangeable. Division of labor, of course, has been practiced for many centuries and in a range of societies worldwide. (Nor is it limited to human culture, being found also in the natural world.) But by itself, division of labor is not sufficient to win the economies of scale that result from modern manufacturing methods. As long as parts were not interchangeable, production of any manufactured item had to be done on a piece-by-piece basis, each piece being unique. Only when the *individual components* of a manufactured item were submitted to quality control mechanisms, such as jigs, gauges, and quality checkpoints, could higher-order economies be realized.[5]

A "go/no-go" gauge is a device used precisely to provide such a check. The utility, and ubiquity, of such a device is instantly recognizable to anyone working

4 As it was dubbed at the Crystal Palace Exhibition in London in 1851, where the arms manufacturer Colt demonstrated interchangeable parts. Needless to say, there is nothing inherently "American" about the principle (an idea that had been around, in Old and New Worlds, for many decades) or its application.

5 See [Hounshell 1984].

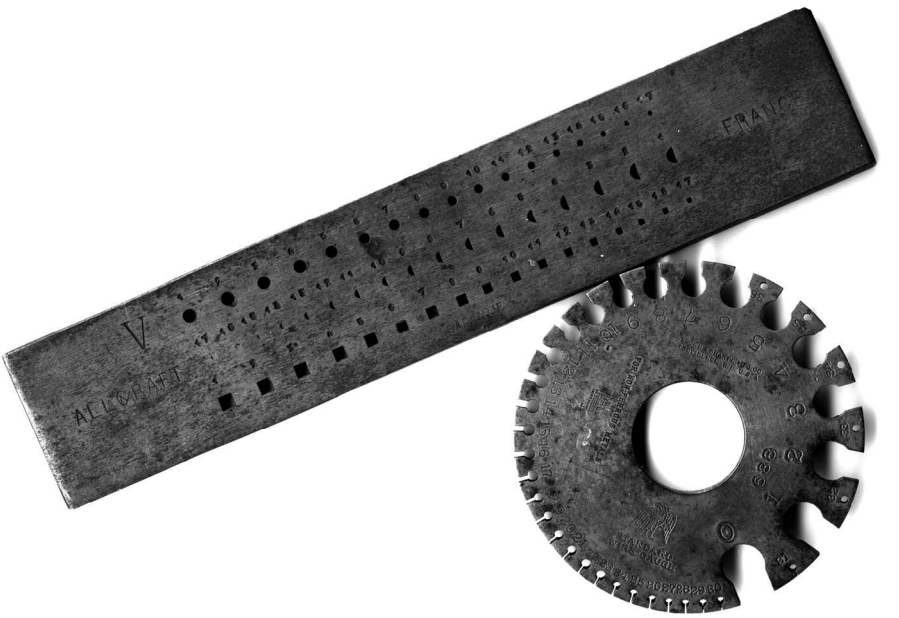


Figure 1 | A go/no-go gauge, with accompanying jig

The gear-shaped device is a wire gauge. Either a length of wire is a specified thickness, or it is not. Pictured with the wire gauge is a draw plate, used for drawing out wire of different gauges (thus serving as a jig). The draw plate must necessarily conform to the gauge in its measurements (and be checked from time to time to be sure it has not worn).

A DTD or formal schema functions as a gauge when we use it to perform strict validation, as a jig when we use it to configure, for example, a structured editing tool.

Thanks to B. Tommie Usdin for lending these examples of the tool-maker's art.

with a complex workflow — especially a process which already involves a complex division of labor or differentiation of roles.[6]

Although they are no longer physical objects, we test our information sets against abstract specifications for the same reason that in a factory, the machine tools are set up to mill parts to exact specifications, and are frequently tested (the tools themselves, that is) to reassure conformity. In fact, the markup industry's leaders have unerringly, if not always deliberately, been proponents of open standards for markup technologies for the exact reason (among others) that it is standards-based interchangeability, when applied to information objects, that provides us with the coveted advantages for our data of vendor- and application-independence, of modular architectures and layered systems, commodity tool

6 A significant detail about gauges used in machine tooling is that they may be crafted, and must be maintained, by hand. In effect, the craft shifts from the creator of each individual product, to the industrial engineer who develops a product *line*.

markets, and long-term data stability. (Not that any of these things become easy to achieve even on a standards basis: but at least with standard ways of judging correctness, there is some hope for them.)

Whenever a validation technology is applied this way, I think it appropriate to call it "strict": I want to convey that it proceeds by posing a binary choice: thumbs-up or thumbs-down. Note that this does not indicate anything about what, precisely, is being validated (structures, data types, referential integrity etc.), or even how extensively, but rather the manner of and rationale for validation. The expectation is that if a document instance fails to validate, there is something wrong with it, and it will be diverted away from the main workflow in order to be "fixed".

Strict validation is very usefully decoupled completely from specific applications. (The measurements of the parts of the gun may be tested apart from the gun itself.) The effect can be to loosen the bindings between stages of the process or layers of the system, allowing agents to work more independently. (Gun parts can be manufactured in one place and assembled in another.) "Can we process this in our system?" An electronic document, like any other manufactured component, must satisfy strict constraints in order to assure predictability downstream; but if we can validate *apart from* the eventual application, the producer of the document on one side, need not have any knowledge or interest in the operation to be run on the other. This decoupling creates opportunities for reuse: the familiar hub-and-spoke architecture of markup-based publishing systems — with a generic format in the center and different formats for production or interchange on the outside — becomes practical. In many cases, validation is therefore useful (as has not escaped notice) for specifying contracts, as the mechanism for a gateway (to an authenticated "safety zone"), or as a "seal of approval".[7]

While challenging to engineer and document, markup-based information systems that routinely subject their data sets to such rigorous specification and testing — and especially when built to standard specifications, enabling them to take advantage of commodity tools — have again and again proven to be both scalable, and more flexible over time, than single-layered systems handling media only in presentational (or application-specific) formats.[8] The principles underlying this

7 This was especially the case in systems like those for which markup applications were first developed: a formatter, for example, running replacement macros over a marked-up text, has to work with a narrow range of structured inputs; but by its very nature (it must use available resources to process what's there and not expend resources on exception handling), it is not coded to analyze abstractly whether a given data set's markup conforms to the expected pattern. The advantages of decoupling are here, that limited processing power can be applied strategically to one job at a time. Decoupling provides similar advantages when processing is distributed across organizations or along a workflow or supply chain.

8 Contrasting approaches would be dedicated word processors, which are generally only suited for end-to-end processing by a single person and not for complex editing systems with demanding layouts; or virtually any publishing system for print or the web, which (except experimentally) have only served to automate the very tail end of production.

are exactly those that allow a factory to become more efficient and productive than individual craft workers, once basic problems of workflow, parts specification, machining and conformance testing are dealt with.[9]

Validation regimens are useful (and sometimes necessary) because they stretch processing along a time frame, making it possible to encapsulate tasks, divide labor into roles, and systematize and routinize processing. In an automated system in which a document may take many forms in its passage — from authored drafts to editorial cuts to assembly to formatting and presentation for many media, through a range of various post-publication transformations including indexing and aggregation, only after many changes to end in the morgue or archive, or perhaps never ending at all but persisting as part of the cultural currency, like Shakespeare's plays or Lincoln's Gettysburg Address — dependable processing could simply not happen without validation. Albeit informally and manually, it happens in paper-only information systems all the time. Appropriate validation is exactly the practice that makes it possible not to be applying human intelligence repeatedly to mindless processing tasks, or to resolving (inefficient) decision-making tangles. Large complex systems learn this the hard way, even when they have lots of cycles to burn. Validation allows human intelligence to work better because it only has to concern itself with one set of standards at a time, not with all standards, for all conceivable uses or needs, at every point in the process.

Finally, by supporting interchangeability, external means of validation provide a foundation for an entire economy or even (at the grandiose extreme) for "information ecologies", because they introduce network effects among applications. So we see that XML applications, unlike older applications based on proprietary formats, work not to compete with each other, but rather to complement one another, since each can work in different ways to support a common data set. Accordingly, the usefulness and value of the entire information system (and thus of each application within it) goes up exponentially with the addition of each new application. This is exactly what happened when, for example, gauges of wire or threads of screws were standardized; and it is what is happening today with data encoding technologies.

This is the compelling and overarching benefit to standards-based validation, and it has been provided as a rationale for the deployment of DTDs (document models) in markup systems since their inception.[10] But it is not the only conceivable way of applying, or reason to apply, validation techniques to encoded data.

---

9 It may be that this can even serve as an indicator of those kinds of processes that are receptive to automation. For example, in the case of elementary education, can we define "workflow, parts specification, machining and conformance testing" sufficiently to automate it? Do we want to?

10 See, for example, [Goldfarb 1990]. SGML DTDs provide much more than a model against which an instance could be validated: by indicating tag omissibility, SGML DTDs (along with their associated SGML declarations) also give critical information about how lexical information in an instance (or the lack

Validation as discovery: loose validation

The usefulness of a validation regimen in framing a clearly-defined workflow makes an extremely compelling case for it. But a gauge that can be used to judge a piece of work as pass or fail, can often be used as easily as a measuring device. The same techniques (parsing or querying an instance, comparing the instance to a model) can be used in a more flexible kind of application. Preceding the operation of judging, is the operation of observing. What can we see about this data? Where does it fail to conform to a given pattern? Validation is essentially *analytic*: data may or may not satisfy given constraints; but our exception-handling may be permissive. In contrast to the use case described above, I call this kind of validation "loose". Note that it is the means of application that is loose or strict, not the routine in itself (whether it be referring a document to a DTD, an XML Schema or what have you[11]) — although typically, it may be expected that the type of processing in a loose routine may be less comprehensive, but possibly more narrowly focused, than a strict routine, and so, accordingly, that some tools will be better suited for the work than others. Such suitability stems not from any fundamental differences in technology or methods, but rather from the relative adaptability of different tool sets to the different requirements we seek to address with them.

In other words, we are not using a validation mechanism — a DTD, a Schema, a specialized processor — as a simple gauge. It may be more like a caliper or a scale, a measuring or reporting instrument.

It is possible to envision, in sophisticated systems, looser routines combined with stricter tests. Validation routines may even be connected in series or staged from looser to stricter. But in its purest form, we might expect "loose validation" to be most useful in an altogether different setting.

In a paper delivered to the 1998 *Markup Technologies* conference, David Birnbaum sketched out such a scenario [Birnbaum 1998]. Birnbaum describes an application in which an historical edition of a dictionary is being encoded in SGML, posing a dilemma for the encoder when the dictionary violates its own structural conventions. Does the encoder intervene editorially, changing the text

---

thereof) is to be interpreted. This, however, can be taken to be secondary, as such lexical optimizations are only possible given a deterministic element structure. By disallowing tag minimization, XML reduces the role of the DTD almost to its core, to provide a gauge or pattern for testing the element structure of an instance against a set of external constraints.

11 In XML terms, "validation" is necessarily strict, and with respect to a single given DTD (named in the DOCTYPE declaration). In XML terms, "loose validation" is a contradiction in terms, and it might be better if one were to speak of querying, structural pattern-matching, etc.

Likewise, just as in XML terms "valid" is itself a binary condition, it may be useful to consider "strict" an absolute in this respect, so that one would not say, for example, that one querying or type-checking regimen is "stricter" than another. Rather, *strict* would by definition mean "either acceptable or not"; whereas *loose* would be *any* routine in which a question may be raised whether the document should be rejected or "corrected", or some alternative course taken.

Figure 2 | Brown and Sharpe 599-100 0-1.2" Digital Micrometer

This model is available with an RS-232 port. High-resolution image provided, with permission, by Brown and Sharpe, Inc., of North Kingstown, RI (`http://www.brownandsharpe.com`).

to fit the normal model? This would be unacceptable for a project given to representing the record, not changing it. Does he relax the constraints of the DTD? Then he loses the capability of modeling properly the majority of the dictionary entries, which are structurally conventional. Does he model the exceptions in a parallel structure? This is possibly a workable compromise, but is less than ideal inasmuch as it is precisely that the anomalous entries are structurally exceptional, that the encoder wishes to trace. "We are not conditioned to think of syntactically invalid SGML as a natural or desirable state, or as a practical or appropriate way of representing syntactically contradictory source data", remarks Birnbaum. He concludes that markup-based systems could be far more amenable to the special requirements of scholars working on legacy texts, if they had some capability to handle structurally invalid markup, at least in some kind of transitional mode.[12]

How should we call this approach to markup? The primary goal of markup in such an application is apparently to describe a pre-existing object. In the extreme case, the objectives of future processing (or, more narrowly, of certain kinds of future processing) might be postponed; at any rate, the purpose of the

12 Birnbaum also explores the issue in an earlier paper, arriving eventually at a moderate position: "I do not advocate, of course, that we prepare and publish invalid SGML, or that SGML processing software be enhanced to react affirmatively not only to valid SGML events, but also to SGML errors. But I would suggest that when we perform document analysis on existing texts, we recognize that some oddities may at least logically (although perhaps not practically) be represented not as document structure, but as violations of document structure." See [Birnbaum 1997].

markup is to identify and trace those features of the text as object, that are interesting and important to the encoder. We might like to call this kind of markup "descriptive" — but since that term has already been appropriated for an entire species of markup applications that do not take such a radical position, I propose the terms *mimetic* and *exploratory* to distinguish it.[13]

Now, it should be admitted that in its pure form, exploratory descriptive tagging would be somewhat paradoxical, since the effort is clearly given to tracing textual features precisely so that patterns, as well as anomalies, can be recognized and exploited — in principle, recognized and exploited by automated processors (or we would be using a pencil to do the work). Even if the primary goal is to describe something pre-existing — if need be, to develop a language capable of such description — it remains the case that the goal of *this* activity may be to make automated processing over this description possible. Or is the latter goal the subordinate one; do we want to automate our processing only in the interests of a more exact and complete description? An answer to this question is very rarely stated explicitly. The potential stress between these objectives is something we will come back to.

Still, the idea of approaching a text and doing a direct, ground-up development of a set of markup conventions, without any great concern either for processing or for standards, has its appeal.[14] So, for example, it is not difficult to imagine how a scholar might go about creating a marked-up version of a literary anthology — only to change the markup and adapt it frequently, so frequently that it becomes impractical to track innovations in markup with a formal model. Different poems would have different features marked up. There might be stylesheets and processors that work on the material, but no explicit model that constrains the entire thing. The markup would be more in the way of a running commentary and apparatus, than it would be a single system bound to processing in a particular way.

---

13 "Mimetic" in that it aims to "imitate" its source, and "exploratory" in that its design is adaptable. The term "exploratory" was suggested to me by Geoffrey Rockwell, of McMaster University, who attributes it to John Bradley (of King's College, London): "One of the things that struck me about COCOA and XML is that in certain situations you don't know what the final hierarchy will be. In the early stages of markup of something for study ... you need something flexible and simple like the COCOA tags. At the end you should know enough to reencode descriptively. . . . I think John Bradley has called it exploratory coding. The problem with COCOA is that it doesn't let you make the transition from exploratory to descriptive easily. Ideally one wants something where you can, once you are sure something is fixed, replace it with a robust scheme" [Rockwell 2001]. The encoding syntax COCOA is a very flexible, non-hierarchical (stream-based), event- or milestone-driven markup scheme, interpretable by several early open-format text analysis packages.

14 Nor is there any reason why this couldn't be done with XML ... such a project would reverse the usual order (design and DTD development first, then mark up the texts), and concentrate on transcribing an analysis of text *in the process of analyzing it*, then working over the markup to recognize patterns and locate points of interest. Any models would only emerge later. Having introduced the notion of well-formedness, XML should be very well suited for this.

In contrast to more familiar kinds of markup, it is worth noting two particular aspects of exploratory, mimetic tagging. First, in this kind of work, the tagging comes first, the modeling later — if there is a model at all, it is subordinate to the tagging in the instance: it merely describes it, never dictates to it, and is not deployed as a way of introducing constraints, except provisionally. Second, in this type of tagging, there is no question as to what the markup describes (instance or model): it is always the instance. The model does not exist *a priori*, but rather only as a (second-order) description.

Interestingly, such a strategy seems to have been part of the original intention, at least among some of its developers, for TEI tagging:

> A balance must be struck between the convenience of following simple rules and the complexity of handling real texts. This is particularly the case when the rules being defined relate to texts which already exist: the designer may have only the haziest of notions as to an ancient text's original purpose or meaning and hence find it very difficult to specify consistent rules about its structure. On the other hand, where a new text is being prepared to an exact specification, for example for entry into a textual database of some kind, the more precisely stated the rules, the better they can be enforced. Even in the case where an existing text is being marked up, it may be beneficial to define a restrictive set of rules relating to one particular view or hypothesis about the text — if only as a means of testing the usefulness of that view or hypothesis. [Sperberg-McQueen 1994]

Note that in this view, validation is a means not only of testing a text, but also of testing the model that (provisionally) purports to describe that text.

After everything, exploratory markup will be difficult to justify for most applications, especially over the long term. Since it does not rely on or stress methods of strict validation, it does not share in the virtues of scalability. Likewise, it is difficult to envision how it could be conducted except by practitioners who are expert both in markup technologies, and in the specialized subject matter they are treating. As an instrument of analysis and representation of a literary text, however, this kind of technology would have great potential.[15] And it is not only the literary scholar who might be interested in this avenue of approach, using document markup in a new way. It could prove to be a useful methodology in psychology, sociology, economics — any study with a complex and manifold data set — and a source of hitherto-unthought-of ontologies and modeling techniques.

15 A fine example of a project of this kind is Willard McCarty's *Analytic Onomasticon* to Ovid's *Metamorphoses* [McCarty 1999]. The design of the (non-XML) markup is unique and especially suited to the indexing and tracing of interconnections that McCarty has developed for this poem. In the end, the markup will validate to its own kind of model (its own set of gauges). But this is a case where exploratory markup has grown directly into something more "procedural" (or at least application-bound).

Mapping the territory

Apparently there are two kinds of descriptive markup: the classical form (what I will identify as "generic" markup) which works descriptively but which is aimed at future processing, and what may be called an "exploratory" approach to markup. In practice, the difference between these is primarily that exploratory markup will not rely especially on strict validation, in particular when the requirements of a strict validation regimen may interfere with the markup designer's capabilities to introduce new terms to refine or extend an accounting, treatment or handling of the text. A more conventional generic language, however, validates strictly, thereby allowing more-or-less dependable bindings to downstream processing. As we turn back to the classic "descriptive vs. procedural" dichotomy, it may be helpful to keep this possibility in mind.

Descriptive markup and validation

Whatever the explanation, it is evident that "descriptive languages" work (meaning, this time, generic languages but not their exploratory cousins). It is possible, and at times highly practical, to have a formally-defined document type that provides considerable advantages for processing — because it admits of strict validation — and yet, that works by describing an abstract model rather than by committing a data set to one or another kind of processing format. In other words, although there is an inherent stress between, on the one hand, requirements for, or intentions or biases towards the kind of consistency enforced by strict validation (a consistency that lends a data set to future processing), and on the other, to the backwards-looking interests and tendencies of text description — although these purposes are sometimes at odds, nonetheless they are not so mutually incompatible that a workable compromise, taking advantage of the capabilities of either, is not possible between them.[16]

Generic markup languages occupy exactly this middle ground between being bound to a certain kind of processing (the "procedural" side), and very loose languages (maybe they are merely markup conventions or practices), that have great freedom to trace their subjects, but that may be hard to deploy or scale up in production — the truly "exploratory" descriptive languages.

Consider Figure 3. In this diagram the exact placement of one or another language might be disputed. At this point, the placement really matters only

16 Again, I do not think this is accidental. Ever since Gutenberg, the automatability of print has been regarded as one of its most important features. Print applications in particular — everything from newspapers to academic journals to catalogs of every kind — have always been at the forefront of automated production systems precisely because the codex has been a successful technology, answering to people's wishes for granular access to information. As technologies of production have evolved, so has the codex form itself — with its headers and subheads, footnotes, indexes etc. — into elaborations that require formal consistency to function. Thus, not just the technical, but the conceptual groundwork for markup-based systems was laid by the evolution of print media.

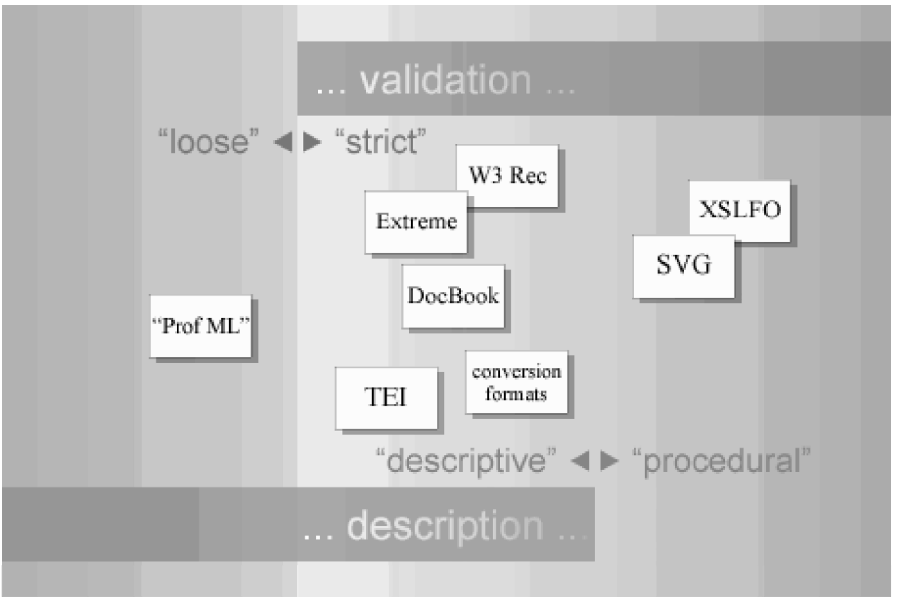


Figure 3 | Markup Languages mapped with respect to validation and description

Strict validation is only possible with a specified set of constraints, so it is at odds with any application of markup that must describe the data with "higher fidelity" than those constraints allow. Yet an in-between zone exists, where formal models provide for strict validation, but are "descriptive" (and so, application-neutral) enough to support a range of different kinds of processing.

along the horizontal axis. (Languages are also distributed vertically, both for legibility and in anticipation of my argument to come.) On the left is a fictional language, "Prof ML", which (we can stipulate) is a set of markup conventions that could be used in an exploratory way. Procedural languages such as XSLFO and SVG are far to the right, indicating not only that their binding to processing is strong (they are expected to be processed one way at least, if not others[17]), but also that if we wish to validate them apart from processing, DTD or even XML Schema validation may not, by themselves, be sufficient. (Both XSLFO and SVG imply, in effect, through their constraints on attribute values, notions of "data types" that are stipulated over and above the constraints on element structure. Whenever an attribute value is expected to resolve, for example, as CSS, XPath or SVG path syntax — all of these amounting to distinct syntaxes apart from the grammar of the document as XML instance — we will need more than a DTD to validate.)

17 A procedural language could in fact target more than one application. XSLFO, in fact, verges on this by targeting on-screen display, print, and audio output.

This diagram also dramatizes how, when strict validation regimens are introduced, there is also necessarily a shift in emphasis for design. On the left side, models are probably informal and implicit in the documents (since if we are not validating, any model must be provisional); whereas as we move to the right, models will become formal and explicit (in the form, say, of a DTD or XML Schema); so a generic descriptive language that validates, ends up describing not the text "as in itself it really is", but a theory about, a model of, the text.[18] To set out to describe "the text itself" runs the risk, at least, that in the long term validation will fail on us, as the model fails to "flex" to the ever-open possibilities for new description.

There will always be a tension, in some ways irreconcilable, between the impulse to fit and form a text, or a markup language, to the peculiar circumstances and opportunities of the moment, and the attraction, and profit, of submitting ourselves to a regimen good for all time. How to position our design between these poles, is what we are determining when we try to "tune", as it were, the level of abstraction of a markup language: we are determining to what degree and in what respects it will be flexible, in what respects specific.[19] But regardless of whether the underlying rationale is a fiction or not (the notion that there is one regimen of tagging that is good for all time — for some more narrowly scoped tasks, it may not be a fiction at all), there is a kind of genius in exactly that rough level of validation achieved by SGML DTDs (of which the XML DTD is, for these purposes, a more refined form). Enough structure is there to support workflow-based go/no-go tests; yet the models are semantically opaque enough[20] to work generically. This allows SGML- or XML-based systems to occupy a middle zone, validating up to a useful point, but also having enough flexibility to work, albeit fairly roughly (only one hierarchy, etc.), "descriptively" — at least when the tag set is well designed. That it is not truly exploratory is something that has occasionally been pointed out as one of SGML's weaknesses.[21] But any number of successful medium- and large-scale systems are demonstration enough that a middle ground is possible — and a rewarding place to build.

---

18 The Greek word at the root of "theory" has a sense of *seeing, beholding*, with an implication that there is some object there to be seen. Once we have a DTD, we actually have such an object. Of course, it can be argued whether a reader or interpreter *ever* encodes anything but a theory of a text; nevertheless, it should be evident how the necessity of modeling in a certain way, would influence the direction of what (and how) the text is theorized to be.

19 Here my argument has been anticipated by Liam Quin. See [Quin 1996].

20 Robin Cover ([Cover 1998] and [Cover 2001]) assesses SGML DTDs as lacking in semantic transparency, therefore inadequate for many modeling functions. But (as I will argue further below) the DTD's semantic opacity in this sense, is actually of benefit for certain kinds of systems.

21 See, for example, Ian Lanchashire's comments in [Lancashire 1995]. At that (relatively early) time, Lancashire's critiques addressed perceived shortcomings in both SGML and TEI, without always being clear which is which. But many or most of his arguments would have been neutralized if TEI tagging could be something closer to exploratory (which, given the role of the DTD in SGML systems, it could not have been).

Adding another dimension

When Allen Renear examined these questions [Renear 2000], he came up with an analysis of the problem with several points of contact with mine. The gist of Renear's argument can also be presented as a diagram.

Note that Renear was not concerned to examine the role of validation in these systems, so his horizontal axis maps only roughly to mine, distinguishing only between different "domains" which a markup language might address. But I think it is not unfair to relate a discrimination between logical and renditional domains, to a distinction between the kinds of constraints each domain may be expected to introduce, and the conditions of their introduction — even apart from the semantics those constraints imply. Whereas a renditional domain must, in the end, "validate" in its application (either the stuff formats properly, or it does not) — and whereas it is likely that in order to do so, some markup semantics may need to be observed that are outside the scope of DTD-based structural validation (so that a DTD-based validation regimen would need to be supplemented to be complete) — the "logical" domain, on the other hand (especially as it concerns what Renear describes as "content objects") might well be defined in such a way that a DTD is sufficient to describe it.[22]

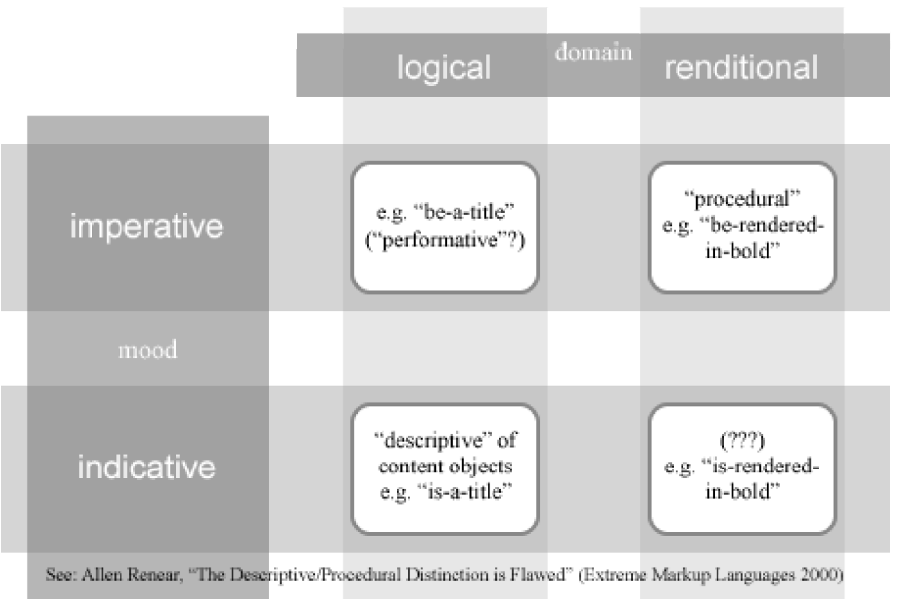


Figure 4 | Renear's Map

Allen Renear's speech-act linguistic analysis of markup languages. See [Renear 2000].

22 Recollecting Robin Cover's argument about the semantic capabilities (or rather, the lack thereof) of SGML/XML ([Cover 1998]; see also [Cover 2001], it may be that we have here a case of the tail wagging

What my analysis adds is the suggestion that to bind a tag set to a particular kind of processing (whether it be in the "renditional" domain or not) implies both strict validation, and a range of other considerations and constraints such as data typing or referential integrity between elements (which may require more fully-featured validation mechanisms than DTDs alone); whereas to work in the "logical" domain puts us in a relatively free in-between zone, where validation provides us the benefits of predictability, control, and a model-centered design, but where the semantics of the markup itself does not rise to the level of specifying behaviors (without the kinds of mapping or augmentation that are provided by stylesheets) — thereby leaving it to be "clean", "logical" and "generic".

But Renear's strongest contribution is in adding a dimension we have not really attended to. By discriminating on a second axis (I have made it vertical) between "indicative" and "imperative" (or "performative"), Renear isolates a very useful axis that had gone pretty much unnoticed. (I believe his basic proposition, that the descriptive/procedural distinction has served to mask this dimension, to be essentially correct.) In my diagram we might notice, for example, that notwithstanding the apparent advantages of generic markup, it is still evident that there is a clear difference even between (say) the W3C Rec document type (the DTD by which W3C drafts and recommendations are marked up), and (say) TEI markup. In a sense both may be considered to be descriptive: but it still seems significant that one presumes to describe something that already exists (TEI documents usually purport to be faithful representations of texts already extant in print or manuscript), whereas another (W3C Rec) describes something that never exists apart from its tagging (or in products derivative of that tagging), to be created and then maintained in that form.

While Renear himself is not altogether satisfied with the categories he proposes,[23] it is evident that either or both "imperative" and "performative" can provide the necessary distinction from the opposite term, "indicative".

To reduce this to its essence, it appears markup can look "into the text", or "out to the application" (this would seem to be a very loose way of characterizing our old friend, the descriptive/procedural distinction); but it can also look forward in time, to eventual processing, or it can serve, irrespective of application,

---

the dog: if the semantic expressiveness of DTDs were richer, the "logical" domain could be accordingly more fully-featured. Models would be more directly tied to processing semantics — and we would not have had the same chance to learn the capabilities and occasional advantages of the looser coupling between model and application that the logical domain implies.

23 Although imperative and performative moods are supposed to be distinct in the scheme Renear proposes, in his treatment he is not quite able to clarify why the mood of a "renditional imperative" and a "logical performative" (a bit of markup that makes something a title, say, by so labeling it) should be considered to be different. I submit that the difference is one of agency. An imperative is spoken by one agent, to be performed by another, whereas a performative is something that is done in the speaking of it. But, when applied to markup languages, this in turn raises other questions: is such agency a property of the language itself, or is it determined by the design of the architecture in which it functions? In linguistic terms, the "pragmatics" of the situation are entirely different.

to represent some state that pre-exists, for example in a document already extant. While it might be tempting to call the latter kind of markup "descriptive", this requirement is in fact orthogonal to the requirement for application binding we have been examining so far. Renear's major contribution, by identifying a kind of markup in the logical domain, but the imperative or performative mood, is to show that descriptive markup (in the traditional sense of the term) can in fact look either back, or forward. In fact, many or most of the current initiatives in XML languages are of exactly this forward-looking type. The markup serves descriptively, but only to describe the text's content with respect to a logical model, designed to be amenable to some particular kind (or some range) of processing. This is quite a different thing from using markup to describe some extant artifact in the world. A confusion over the stresses between the two views is at the heart of many design problems and infelicities.

We can adopt this point of view in developing our map of markup languages: one way to name this new axis is between "prospective" and "retrospective" markup languages. A retrospective markup language is one that seeks to represent something already existing; whereas a prospective markup language is one that seeks to identify a document's constituent parts as a preliminary to further processing. Prospective markup, that is, may be "procedural" in the sense that SVG or XSLFO is. Alternatively, it may seek to claim all the advantages of generic markup (scalability, strict validation, content re-use etc. etc.) without having to be bound to describe anything apart from itself.

In my map (Figure 5), this could be distinguished by a vertical axis, "prospective" corresponding to Renear's imperative/performative mood, "retrospective" corresponding to Renear's indicative; but it is interesting to see that when we begin to place actual markup languages into this conceptual space, that there are blank spots. In particular, there are two positions left empty in a possible grid of six (we can conceive of Renear's arrangement with a new domain to the left, "exploratory/mimetic", next to logical and renditional to the right). For one, it seems unlikely that we would have an application of markup that is both prospective (Renear's imperative), but exploratory, having no use for validation or the kind of binding to (even implicit) semantics that validation implies: if we are creating a new format for a new application, what does validation lose us? It could be that markup instances that are purely *ad hoc* files for momentary processing, would fall into this category.[24]

Equally unlikely would be a conjunction between retrospective and procedural (or application-specific). This would correspond to Renear's category of "indicative renditional", which he also remarks would seem to make little sense.

24 I actually think there is an important role to be played by such little languages, exploring not artifacts or texts, so much as processing opportunities.

Evidently, procedural and retrospective markup serve requirements that are in conflict. We can either describe the world as we find it (with retrospective markup) or we can dictate in what way we need our data to be handled (with procedural markup). The fact that traditionally, generic markup systems (or at any rate, those that had retrospective designs) have sought to mediate this exact conflict, does not make it any easier to do so. The more we need our application to serve retrospectively, the less we can expect to find thorough, detailed and strict validation regimens of much help.[25]

That is, although we can distinguish a vertical axis that indicates a markup application's orientation in time (forward- or backward-looking), it is clear that this axis is not completely orthogonal to the spectrum of loose-to-strict validation that I began by tracing. It is likely that a prospective application will find strict validation both useful, and not particularly burdensome. To the extent that an application is retrospective (such as might be the case with a markup language written to support conversion of a legacy data set, or a scholarly project in textual editing), however, it may prefer any testing to be loose. In graphing it out, therefore, this axis appears on a diagonal.

Generic markup as a form of rhetoric

Prospective, procedural languages clearly have a place: it would be hard to argue against the utility of standard XML vocabularies such as XSLFO and SVG.[26] At the other extreme, retrospective, exploratory applications of markup would appear to be very fruitful as approaches to certain intellectual problems (although until it became practical to develop markup applications without DTDs, this kind of application of technology was severely hampered by a lack of a standard toolset), particularly problems that have directly to do with questions of how we represent non-digital phenomena by digital, processable means. But what is most interesting here is the broad grey zone between these extremes, a zone occupied by applications of markup that have a need for strict validation as an instrument in workflow and processing architectures, but that are not exclusively bound to any particular type of processing or application, as would be implied by a procedural language. This is the zone of "generic", loosely called "descriptive" languages such as TEI, W3C Rec ML, or for that matter, the language used to mark up this paper.

25 Nevertheless, applications like this are conceivable, and have even been executed in part. For example, if an attribute syntax were to be adopted on top of a generic markup like TEI, especially if the attributes worked to prescribe formatting (embedding, as it were, a style mapping into the generic instance), it might achieve something like this.

26 In fact, as for example in the "XSL Formatting Objects Considered Harmful" argument [Lie 1999], when these languages come in for criticism it is precisely because they have certain kinds of utility (though perhaps not others).

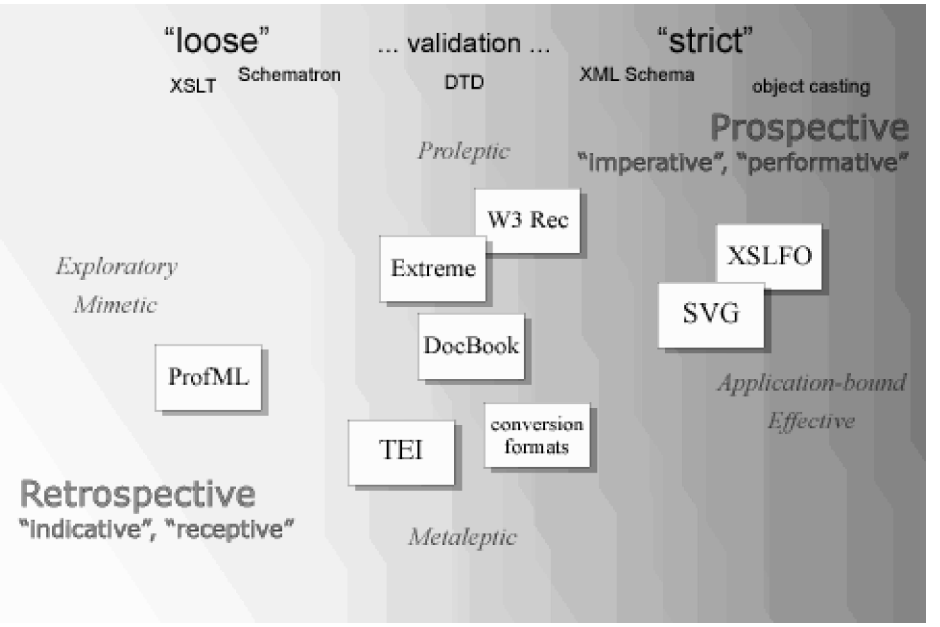


Figure 5 | Markup languages mapped on two axes

The horizontal axis represents the level of validation appropriate or called for, and thus the specificity of machine (behavioral) semantics. Requirements for a tag set to be prospective (provide for future use) or retrospective (describe a given artifact physically or "logically") align along the diagonal lower-left to upper-right.

Most discussions of "semantic" in the context of automated text (or "knowledge") processing end up having to distinguish between two meanings. There is the realm of human semantics, largely if not principally *representative*, our "meanings" when we express ourselves in language or by any other means. Then there are machine semantics, the sorts of behavior, events, products or controlled processes that can be expressed through a machine — and which are the normal objective of workflow-oriented systems. If you like, you can consider this a spectrum between *word* (on the "human" side) and *act* (the machine behavior). (Tim Berners-Lee, for example, in his discussion of the "semantic web", has openly affirmed that he is concerned only with the second kind.)

The world of "content" (text) that is encoded generically is a fascinating one, in which these two competing notions of the semantic discover themselves head-to-head. In this world, markup simultaneously links people and processes in different roles, and serves as a conduit or channel for "meanings" that have the interesting property of skipping or passing through stages in a process (a "supply chain") that can go directly from creator and producer, to audience or consumer. That is, markup provides a kind of framework or packaging by which words

(written texts or representative codes) can be passed without consideration of what they "say". As a kind of interchangeable part, as long as the package or framework is correct, the meaning or substance of the "text itself" (what we call the "content") can be more or less completely opaque to participants along the chain. The framing or wrapping provides the text with sufficient information (about its nature, about its internal structures and relations) that it can be passed and processed without constant rediscovery and reinvention. This wrapping or packaging takes the form of markup; and of course, relative to the processes, the markup is meaningful — yet its meaning is local and provisional. Accordingly, these can be staged systems in which interpretation happens in an articulated way. For example, authors decide some things, bibliographers some things, catalogers some things, layout designers others. In such a system, a degree of "semantic opacity" is a *feature* (cf. [Cover 1998]), allowing us to provide appropriate processing based on some kind of "intention" as a tag set presents that, but always leaving it up to us to decide finally what that means. The masking or withholding of any details or specifications for processing, that is, behind "element type names" or "generic identifiers", is part of what makes such a system work: by not insisting on a binding to any one thing, the mechanism of element typing is free to support an open-ended range of things. Note that exactly insofar as machine semantics is devalued (or rather, postponed or layered) in such a system, the expression of human semantics becomes very important: generic markup languages become worse than useless if their tag names are cryptic or if they are not well documented. But when a markup language is designed well, it can be used to frame and drive a process in which different participants can provide their added value, each without having to get involved in exchanges of no direct concern to him- or herself.

In these respects, a generic language is able to be, or to pretend to be, exactly, *descriptive* or representational — meaning that, pragmatically, it has some kind of implied human semantics, without being bound to any, or any particular, processing (machine) semantics. That is, we take the tags to "mean" something — but what they actually mean, in the event a file is ever processed, may be different from (albeit in some way implied by) the meanings of the tags. In other words, there is a *slippage* between what a descriptive tag set purports to mean, and what it actually "means" (*does*) in the event.[27] This slippage is the

27 Robin Cover argues [Cover 1998] that this makes it important to provide XML with a means of strong semantic specification, which in and of itself it does not have (since XML syntax, nor DTD-based content modeling, are incapable of providing it). In Cover's terms, this is XML's lack of "semantic transparency". And for procedural applications of the syntax, this is certainly a critical issue. It can be addressed in several different ways, for example by providing some kind of formal ontology; by merely presenting a notation for some other data model; or by passing the problem into a syntax carried in attributes, such as CSS, XPath or SVG path syntax. Yet for descriptive or generic applications, XML's semantic opacity is actually be a feature of the technology. It's where things can get slippery between layers.

source of the power of descriptive languages, their famous "indirection": meaning nothing directly, they can be taken to mean a great range of things if we only bind their evident and ostensible meanings (that in practice do nothing but structure and disambiguate between types) to behaviors. To tag a data element as a `title`, say, may mean nothing more than "whatever you do with titles, do it with this thing".

So generic markup involves us in a strange paradox. It foregoes the capability of controlling behavioral "machine" semantics directly, but wins, in return, a greater pliability and adaptability in its applications for human expression. This kind of middle-ground markup would be systematic enough to be receptive to automation, but would not necessarily be automated "out of the box". Another way of describing this kind of markup application, as opposed to more strongly typed and validated kinds, is that this is the kind of system in which a stylesheet writer has something significant and important to do. Stylesheets are a natural way to get from an abstract model, into an application. But they might require, as stylesheet writers know, some addition of information, interpretation and restructuring, as well as mere mapping. Stylesheets are also where a great deal of creative work can come into play.

If this variety of markup language is not really a set of instructions, but a complex representation (on which a later process may be expected to act), the proper discipline for regarding it would seem therefore to be, not formal languages (that have the virtue of being readily bound to processing), but something closer to linguistics and rhetoric.[28] This is the realm where we experience slippages — whether inadvertent, or "intentional" — between actual and potential meanings.[29]

In effect, markup languages are far more than languages for automated processing: they are a complex type of rhetoric working in several directions at once, often in hidden ways. Inasmuch as markup systems then may begin to resemble other textual systems (such as literary canons or conventional genres), it is reasonable to turn to rhetorical or literary critical theory for explanations, or at least

28 Lately, Michael Sperberg-McQueen, Claus Huitfeld and Allen Renear have sought to formalize markup languages' (including generic markup languages') handling of meaning by saying markup "licenses certain inferences" about a text. (See [Sperberg-McQueen 2000].) In the notion of *inference* — and the evasion of the issue of how an inference can be constrained or defined (since isn't an inference precisely that kind of communication that can't be constrained or defined?) — they effectively elide this transition between formal information theory, and rhetoric (which is enamored of formalisms, but resists being comprehended by them). To "license an inference" is, in effect, to say something without saying it. Is this logic, or rhetoric?

29 To examine this in the context of Renear's categories: one difference between imperative and indicative, or between a "performative" and an indicative (the axis Renear describes as "mood: whether markup describes something, or requests processing" [Renear 2000]), is that an indicative refers back to the past (or disinterestedly to the present or future). It is the projection or implication of some reality apart from the markup (the separation of format from content!), whether this is a feature of some kind as documented, a perception, or an imaginative projection, which competes with processing objectives, that opens up the important area of slippage.

higher-level characterizations of them. I am not going to begin to plumb the depths of this subject here. Given both the complexities of real-world workflows, and the fact that many of the agents are human beings only as mediated through their machine proxies, it is difficult to say who is saying what to whom through (and in) a markup language or markup language application. Then too, the ways in which messages and meanings trace through an electronic text system, is going to be highly, sensitively dependent on the unique particulars of media, technology and culture at work in a particular case. One thing that does need to be observed here, however, is that in markup, we have not just a linguistic universe (or set of interlocking linguistic universes) but also a kind of "rhetoric about rhetoric". That is, markup languages don't simply describe "the world" — they describe other *texts* (that describe the world).

As it happens, critical theory has had occasion to consider such complex types of figuration, representation or meaning. I am going to draw on the work of scholars who have studied intertextual referentiality[30] (where this type of phenomenon is especially pronounced), to distinguish between the tropes *metalepsis* and *prolepsis*. These are distinguished from the usual run of rhetorical figures such as metaphor, metonymy and so forth, because unlike others (which are occasions of figurative representation), these are tropes about tropes. It is not "something in the world" that is represented in a metalepsis (or its less common complement, prolepsis), but rather some other act of figuration.[31]

Proleptic markup

Of *prolepsis* and *metalepsis*, the first is possibly simpler to grasp quickly: Prolepsis is a rhetorical trope or gesture[32] in which an expression or figure of speech takes its meaning from something that is to appear later. Dramatic irony (where a character in a play, for example, says something that carries an extra meaning to an audience that knows or guesses what is to happen in the drama),

30 In particular, on the work of the poet and literary scholar John Hollander [Hollander 1981] and his colleague, the critic Harold Bloom [Bloom 1982].

31 Hollander (and with him, Bloom) claims that this type of thinking is to engage not just in the usual kind of "synchronic", but a "diachronic" rhetoric. That is, ordinary treatments of rhetoric pay attention to the use of figurative language as if all the signifiers were related outside of time. (This would seem to be a Platonistic view of text, with all meanings always available *sub specie aeternitatis*.) But it is possible, not only to consider how language or signification interacts as a kind of "random access" system, but also to think of how meanings work over time and across it, how they shift and change in relation directly to one another, how they recapitulate or anticipate. This kind of thinking is extremely helpful as soon as we start looking at layered systems and complex, dynamic information interchange — but it involves us, in effect, in representing the flow of information, the stages of its passage.

32 The extremely useful word "trope" may call for some explanation. From the Greek for "turn", it is a traditional word to designate a figure of speech or signification (whether spoken, written, or by some other means), or any occasion when something is expressed by saying something somewhat different. Metaphor is a trope, though its cousin simile (a comparison using "like" or "as"), even when poetical, is only a trope in a loose sense. Other tropes include metonymy, synecdoche, irony, etc. etc.

or literary or dramatic foreshadowing, is prolepsis; but so is any "casting forward" or anticipation, such as an argument one might make in a conference paper in anticipation of counter-arguments. Consequently, the full meaning of a prolepsis is impossible to know without taking account of its relation to the future. Whether what is forecast does, in fact, come to pass in the way forecast, opens prolepsis up to capabilities for irony. On the other hand, sometimes saying something, makes it so: so prolepsis often has the capacity for a kind of poetic "fiat" or self-fulfilling prophecy.

Any prospective tagging might be called "proleptic" because the meaning of the tagging is intimately connected with our expectations for processing it. Even when such markup is generic, we call something a `head` or a `section` because we intend to treat it as a head or a section in processing. This is Renear's "performative" markup: the section becomes a section through the act of naming it so.

But it might also be that the term *proleptic* would be useful to distinguish exactly that type of prospective (performative) markup that works generically, such as DocBook, the W3C "XML Rec" markup, or even certain kinds of XHTML (probably "XHTML Strict"), as opposed to prospective markup that is merely, in effect, an application binding, such as SVG or certain other kinds of XHTML (such as a DHTML application, heavily laden with script and tuned to a particular browser). Admittedly, this too may be a spectrum rather than a simple either/or classification; also, it should be noticed how a markup language may actually "grow into" an application binding — or conversely, how an application binding or API may grow around a markup language.[33] Nevertheless, there will be occasions when, although we have expectations for processing our data, they may not be specific or limited expectations. In other words, we want a method (a generic language) that affords us that slippage between specification and processing. The word "proleptic" seems to allow for this: as a trope, the meaning of a prolepsis has to be seen as conditioned by the possibility, at least, that things don't quite turn out as expected. Especially when marking up new texts (or composing texts in a new language), this is a very powerful way to approach the design and practice of markup: an artful combination of specification and slippage is what enables most of the promises of generic markup to be realized.[34] When we design, we may want to know in detail (or at least in principle) the application requirements of a markup language; we may want to be prospective if not actually procedural. Nonetheless, we always want to keep our eyes also on the bigger picture, since a careful restraint devoted to modeling our information "logically" (that is, in some sense, descriptively, if only to be descriptive of an

33 So, for example, CSS has grown up as an API (in effect, albeit "declarative") around HTML, therefore pulling HTML/CSS further into the procedural than plain "generic" HTML on its own. As an API to a display engine, of course, CSS is useful to more than HTML.

34 Again, see [Quin 1996].

abstract model) rather than in the language actually of an application, pays off in the long run in data independence, reuse, longevity, and so on.

In this kind of endeavor, validation routines are going to be very useful. We will build our workflows around them. More interestingly, possibly, our means of specifying validation, such as DTDs, will be useful as specifications for tools, many of which can be automatically fitted to the task. This is an application of a gauge (the DTD), which is used to check conformity to an external measure, as a jig — a device or tool fitting, that allows us to make the component to measure the first time. In a sufficiently evolved production system, we may never even have documents that are "invalid" in the XML sense, and we may have needs and uses for all kinds of validation besides simple structural element type checking. But we may do all of this without any particular or specific expectations for processing.

Metaleptic markup

So proleptic markup is that type of generic markup that looks forward. What of generic markup that looks at what is past? That is, that tries seriously to register, in some disinterested and objective way, features and organizations of information already out there? In some ways it would seem unnecessary to have to submit a descriptive markup convention to strict validation, with all that implies (we remember Birnbaum's argument [Birnbaum 1998]). Nevertheless, whatever processing we expect to do over data sets, on however large a scale, will demand some kind of validation at some point, and there are many reasons, both intellectual and practical,[35] to try to design a generic language that also tries to capture some "truth" (or at least theory) about the world. Having formalized our theories in abstract models, we can then test them by running the very same validation routines that we apply to encodings that have been specifically designed for processing, not for representation. In the end, validation is not only a testing instrument in a workflow: it is an investigatory instrument in its own right. DTDs are representations of texts. So we look backward, in an interestingly formalistic way. But we also get the benefits of looking forward.

This type of markup tries to be retrospective (and in this presumes to *describe* the data set), but nevertheless relies on, and benefits from, strong or "strict" validation regimens. Such tag sets would include TEI,[36] or any tag set developed for data conversion or retrospective document conversion which seeks

35 Practical reasons: converting large amounts of data from a legacy format. A well-designed model that looks to how that data is formatted, can preserve information through conversion to an open format like XML, and ease the conversion process. Intellectual reasons: develop a theory about a (body of) text; formalize that theory; demonstrate its utility.

36 In most applications. TEI can also be used in a proleptic way, for example when it is used to drive a web site of original documentary materials (a task for which a TEI subset is actually fairly well suited). Notice that it is not a markup language (a tag set) that is *per se* proleptic or metaleptic, but an application of it. Some tag sets can be used in all kinds of ways: HTML certainly has been.

at once to be both descriptive and generic. Markup systems like this are evidently descriptive after a fashion; but it is also clear that their prospective applications, be those presentational, analytical or what have you, are a big part of their conception.

In contrast to prolepsis, *metalepsis* is the rhetorical trope in which the meaning of an expression is in direct reference to what has already happened in the past.[37] Of course, this is in some sense true of all rhetoric, since all rhetoric is situated, in some way, in a moment with a history (and inasmuch as this is the case, all rhetoric is metalepsis, successful or failed); but in a narrower sense, metalepsis is what occurs when reference is made to another*figure* that has already appeared (some event of meaning or figuration that has already taken place), but in such a way that the meaning of the earlier figure is itself changed by the appearance of the metalepsis.[38]

What then would be a metaleptic markup language? Keep in mind, to begin with, that document markup as rhetoric is necessarily complex; there are various levels of expression here. A tag set describes a data set, or it describes a theory about the data set; when it looks back, what does it see? Might it not sometimes have reference to one or more earlier systems of description (earlier theories?), including implicit traditions? A consideration of markup systems actually in use (I've mentioned academic projects including TEI, as well as transition or conversion formats being used in industry), suggests that such a reference is not, in fact, uncommon.[39] In general, metaleptic tagging will act retrospectively, and may even

37 John Hollander, considering metalepsis as a "diachronic figure", describes it as related to allusion but entailing a deliberate relation between before and after. "We deal with diachronic trope all the time, and yet we have no name for it as a class. An echo of the kind we have been considering may occur in a figure in a poem, and it may echo the language of a figure in a previous one. But the echoing itself makes a figure, and the interpretive or revisionary power which raises the echo even louder than the original voice is that of a trope of diachrony" [Hollander 1981]. As a variety of allusion with "interpretive and revisionary power", metalepsis is not any ordinary act of signification or representation: it is a representation with reference to another (previous) representation. Once he has alerted us to this possibility, Hollander is able to show that gestures of transumption (the Latin "transumption", with its morphological variant the verb "transume", has long been a variant of the technical Greek "metalepsis") are in fact not uncommon in literary language. "Save for dramatic irony, with its audience's — or reader's — proleptic sense of an outcome of which the dramatic speaker is unaware, and which engenders an interpretation more powerful than the raw intended meaning of the speaker himself, only transumption seems to involve a temporal sequence" [Hollander 1981]. His fascinating book *The Figure of Echo* contains a thorough examination of the dimensions and history of this category in critical theory. Nor is this conception, concludes Hollander, of application limited to poetic language. "Not only particularly preexistent metaphors, but formal structures — and M.H. Abrams and, more recently, Paul Fry, have shown us authoritatively the intricate turnings of the transumption of a previously public form in the history of the ode — are recreated metaleptically. So are genres" [Hollander 1981]. And so, I submit, are markup languages.

38 The Christian New Testament is metaleptic with respect to the Old Testament. Virgil is metaleptic with respect to Homer, Dante with respect to Virgil. Strong poetry is almost inevitably metaleptic, since poets, it seems, cannot help but echo and try again their predecessors, but in such a way that they commandeer the older works and set them to later purposes. Blake and Shelley succeeded at this so thoroughly with Milton, that we cannot even read Milton any more (if we do at all) without meeting up, in some way, with Shelley's Romantic heresy. This also happens in musical traditions: Brahms is metaleptic (or attempts to be) with respect to Beethoven, and so forth.

39 I suppose any extension of a standard or off-the-shelf markup language might be metaleptic in a simple

pretend, and attempt, to be thoroughly descriptive and retrospective in its relation to already captured information (figures already spoken); but it relies on strict validation. This betrays its true nature: its design and application is really done for purposes of future uses of the data (new meanings), not merely to "describe" in the more limited senses of that term. It is retrospective tagging for prospective purposes: thus, it works by saying something about the past (or about the presumed past), but in order to create new meaning out of it.

Typically, it does this by positing a model of the text and then asserting, implicitly or explicitly, that this model is sufficient for all practical (if not conceivable) descriptions or applications of the text. And in well-designed, mature systems (by which I mean ones which have clarified the way they actually work and are not confusing either their rationales or their design with those of other markup applications), metaleptic languages do in fact function very nicely as generally-accurate descriptions — though it should be added, that when they succeed in this way, it is typically because they determine *not* to try and describe *everything*.

Just like any other future-bound processing, this kind of markup will be able to take advantage of strict, go/no-go validation. Because this kind of tagging often originates as a description of a given artifact (a known text), it is easy to identify it with true descriptive markup. But as I've said, that is a very rare thing (unheard of in commercial or industrial applications): the goal of describing a pre-existing object must generally yield, at some point, to the more practical need to constrain and process the information set. Hence most markup languages that go by the name "descriptive" are only so up to a point — they are in fact metaleptic.[40] A metaleptic markup language (or rather its designers and advocates, perhaps its users) may be entirely innocent of any perceptions of stress between extant documents, and abstract models — fundamentally, the stress over which many struggles over validation will take place. Absent any consciousness of such a stress, a metaleptic design may take, or propose, its model or theory of the text as a kind of reality, thus claiming the title "descriptive". But we can know it for what it is when we see it being validated strictly, and when we also hear, in addition to its claim that it works by description, that it expects all the benefits downstream of validation, in the data set's readiness for further processing (be that publication of electronic or print editions, providing database access, or what have you).

---

way. But more common, and more complex, are cases where the references are merely implicit, if sometimes obvious.

40 In fact, the process of formal document analysis as it is practiced in the markup industry, can involve a complex interplay between an actual descriptive exercise, as a way of driving work and exploring the problem domain, while ultimately keeping focus on requirements for future processing.

So far so good; the dark side of metalepsis is, possibly, when it denies its own complex and layered nature. An act of transumption (a synonym for "metalepsis") changes, transfigures, that which it transumes (in this case, "describes").[41] To pretend otherwise, it would seem — to pretend, for example, that our representations are in all respects (or even all important respects) identical to what they represent — would only have the effect of setting ourselves up for disappointment. In the worst case, we may completely fail to determine our actual needs and fit our design to them: stuck on the horns of this dilemma, we may end up with neither an adequate representation of our source text (however we define that), nor data that is well suited for automated processing.

More commonly, rather than being purely descriptive/exploratory, or purely proleptic, applications adopt a metaleptic design strategy because they need to meet requirements on two sides, past and future. In time, if they are lucky, they grow into a consciousness of their ambiguous status; but the actual rationales, expectations, and design of these projects are often complex and intermixed. Sometimes project participants themselves have not exactly clarified what their main interest is; often they are working with several conflicting rationales or requirements.

Yet in general, the emergence of this type of markup is of great importance because it has led us more quickly and readily to understand the efficiencies, power and scalability of layered markup systems: just like proleptic markup (which is generic without being retrospective), metaleptic tagging is very much at home in such a system of at least two tiers, possibly because it itself has two faces, looking in and looking out.[42] And when they are well designed (which not coincidentally, often means *intentionally* designed) and appropriately deployed, such a markup language can be fascinating in its own way, quite differently from either of the other two forms of markup that are prevalent (leaving aside exploratory markup as more rare than it should be, we also see generic proleptic markup, and procedural applications). It has its own kind of art. It does not try merely to transcribe, as purely descriptive, exploratory tagging would (though as scholars know, "merely transcribe" is an impossibility and an oxymoron), nor merely to function in future systems, like prospective markup (spectacular though

41 In a metaleptic markup language, there is a missing term standing between the language itself, and the text, the presumed "content" that completes (and is completed by) the markup: that term is the theory of the text, the model, that the language formalizes. (Here I am concurring with Paul Caton, [Caton 2000].) It is the movement from one term to the next (here, from text to theory, theory to model, model to application) that makes for the rhetorical complexity of such a language, sometimes most complex when it aspires to be most "transparent" — and that may help make applications of these languages suitable for particularly interesting processing, as being particularly "slippery".

42 This is of course the famous separation of format from content. Two tiers would be the repository and presentation layers (think of a TEI text and its HTML rendition); this also maps over to the model/view/controller paradigm, with "descriptive" or "generic" instance as model, rendition version (say, HTML) as view, and stylesheet or script as controller.

that might be). It aspires to both, by seeking to balance between them. An effective markup language will work by establishing a self-contained, internally consistent and clear set of categories perfectly sufficient for handling the data set to which it will be applied, within the range of applications for which it is due. But this ideal is impossible for a truly descriptive language to achieve, since the world is not a closed, finite set of phenomena that is liable to such treatment.[43] Metaleptic markup gives us the next best thing: it invents its own imagined world, proposing earnestly or ironically that this serves both sides, both accounting for external reality as it is, and creating it as it needs to be.

TEI, incidentally, has occasionally been represented as a true retrospective tag set, yet is torn about the issue. It aspires to provide certain functionalities along with transcription, such as eased production costs for print or online editions, or eased repurposing across different applications, that can only be guaranteed through strict validation. Up until recently (when XML has made processing without a DTD more practical), validation has been a particularly all-or-nothing proposition. New (and newly accessible) tools and approaches supporting "loose" validation may now seem more of an option than they have hitherto (especially to strapped academic programs). Nonetheless, TEI cannot help but continue to be powerfully metaleptic: pretending to be simple, naive, retrospective (and accordingly, extensible!), *and* simultaneously stressing validation as a means of smoothing transitions of its texts to new media and new applications — a *prospective* gesture — it ends up "falling into" metalepsis despite itself.[44]

Finally, it may be worth observing how architectures supporting metaleptic languages or applications will differ from those for proleptic languages. For one thing, we will be able to tell the difference when we look at a project's regard for, and use for, validation mechanisms. For a metaleptic system, validation will need to be strict to the extent that future processing is anticipated. On the other hand, since there is at least a presumed interest in the prior or "original" nature of the textual content's own structures, features or organizations — however these are conceived — it may be at times that the best solution to a misfit between document and formal model is to change and adapt the model (and accordingly, the system) rather than forcing the document. As in pure exploratory applications, markup is designed first, formalized after, whereas in a proleptic system, the model or schema will come prior to the markup. This difference in emphasis may

43 This observation has often been made informally, in a variety of ways. "Selection is easier than synthesis, but the world is not finite", says Brian Reid [Reid 1998].

44 This tension can be seen to play out exactly in the role validation is expected to play in TEI projects. On the one hand, the tag set is provided with an apparatus to support extensibility. This is the promise of descriptive markup: that no text should have to be forced to fit. On the other, validation is considered indispensable, not only for usual quality-assurance reasons, but also because in it there is an assurance (for example) that the rigors of the `teiHeader` are observed, or that off-the-shelf (or nearly off-the-shelf) stylesheets be able to be used — or that interchangeability be achieved (a prospective requirement, perturbed by local extensions). It is, after all, the Text Encoding Initiative for Information *Interchange*.

make for different toolsets, to an extent. Also, we can expect of metaleptic systems, in particular, that the natural stresses between requirements for description (sometimes in the guise of backwards-compatibility) and for interchange, will be at their greatest: balance will only be achievable if we keep a realistic view of what we intend to achieve and how we intend to do it. But when metaleptic systems are well designed, the rewards, both in our mastery of complex bodies of information, and in our understanding of them, will be great as well.

Conclusions

- **"Descriptive" may not be the best word.** It means too many things. Even the procedural languages are descriptive: they describe a binding, API, or object model. The differences are in the closeness of the binding and the extent to which an abstract syntax allows us to validate without binding, hence letting us design a language at a higher level of abstraction (and get capabilities of reuse and refitting thereby). *Generic* is a somewhat more useful term: these are languages that can be strictly validated, but that are only loosely bound to processing. At the far end, markup that isn't validated at all, if it is retrospective, may be said to be descriptive, insofar as it describes some external object (and is therefore directly representational). But historically, no standards have existed to support markup systems of this kind. XML may help stimulate more of this work.
- **Watch out for clashing requirements.** Prospective ("performative") markup can be generic, and generic markup may seek to be either prospective (proleptic) or retrospective (metaleptic), or both together. But the more we try to "describe", the more difficult we will find it to validate (in the broadest senses of that term). We should be careful to distinguish the requirements presumably served by our design strategies. Academic projects with a commitment and interest in description of something external (say, a literary or manuscript text) may have a particularly difficult time with this — for example, when an exploratory design clashes with a requirement for interchange. The "descriptive vs. procedural" distinction can, if we are not careful, muddy the waters here even further.
- **New approaches to design: bottom-up.** Loose validation with Draconian error-handling at the syntactic level (e.g. XML well-formedness) — even if it involves no "validation" at all in the formal XML sense — should open up new possibilities for design strategies and methods, as well as for new applications of markup, including exploratory modes of markup such as I have described. Up to this point the design process for a document model has usually been driven by a top-down analysis, and centered on DTDs. As long as DTDs provide a useful means for testing for the kinds of interchange and

downstream processing that have been prominent requirements, this will continue to be appropriate. But if and as we design systems and markup languages with other aims — such as, for example, an exploratory application rather than a "performative" or direct application of markup to processing — other techniques and approaches may prove useful. What if designs were centered not on DTD validation, but on stylesheets and query sets that provided meta-information (including validation checks) along with or in place of their more usual kinds of transformations? What kind of markup applications would be well served by such an approach?

- **New complications include maintenance and oversight.** Already approaches to XML validation are proliferating. Which of the various approaches now being tried, both strict and loose, come to be prevalent (and which approaches in which environments and domains), is an issue I can't address. But nothing is either/or here: just because we use DTDs or XML Schema to validate one set of features to requirements, does not mean we can't use other means (stylesheets or query sets, for example) for others.

  If and as we do this, however, we should be careful to keep clear what we are doing where, and why. It could easily become a problem if the same set of constraints on a document set, or type, comes to be validated through more than one tool: this would introduce new problems of parallel maintenance. (It would be like having two rulers to measure things, but not being sure they measured the same "inch".) Yet different kinds of validation, and of tools to do it with, might well be very usefully done at different stages of a document lifecycle. (Such routines have been commonplace for years in any case.) When systems become complex and validation routines overlap, it might be helpful to have a "validation validation" regimen to appeal to. This is what, for example, testing suites for tools provide — just as standardization has been managed, again, even since the very first years of interchangeable parts.

Received 23 July 2001
Revised 5 October 2001
Accepted 8 October 2001

References

[Birnbaum 1997] Birnbaum, David J. 1997. "In Defense of Invalid SGML". At `http://clover.slavic.pitt.edu/~djb/achallc97.html`

[Birnbaum 1998] Birnbaum, David J. 1998. "The Problem of Anomalous Data". Markup Technologies '98.

[Bloom 1982] Bloom, Harold. 1982. *The Breaking of the Vessels*. The Wellek Library lectures at the University of California, Davis. Frank Lentricchia, Series Ed. Chicago: University of Chicago Press.

[Caton 2000] Caton, Paul. 2000. "Markup's Current Imbalance". Extreme Markup Languages 2000.

[Cournane 1997] Cournane, Mavis. December 23, 1997. *The Application of SGML/TEI to the Processing of Complex Multilingual Historical*

Texts. Doctoral Dissertation, University College, Cork. Cork, Ireland.

[Cover 1998] Cover, Robin. 1998. "XML and Semantic Transparency". At http://www.xml.coverpages.org/xmlAndSemantics.html.

[Cover 2001] Cover, Robin. 2001. "Conceptual Modeling and Markup Languages". At http://xml.coverpages.org/conceptualModeling.html.

[Sperberg-McQueen 1994] Sperberg-McQueen, C.M., and Lou Burnard, eds. 1994. "A Gentle Introduction to SGML". In *Guidelines for Electronic Text Encoding and Interchange*. Repr. 1997. Chicago, Oxford: Text Encoding Initiative. pp. 13-36. Available online at http://www.uic.edu/orgs/tei/sgml/teip3sg/

[Hollander 1981] Hollander, John. 1981. *The Figure of Echo*. Berkeley, CA: University of California Press.

[Hounshell 1984] Hounshell, David. 1984, 1985. *From the American System to Mass Production, 1800-1932*. Baltimore: The Johns Hopkins University Press.

[Lancashire 1995] Lanchashire, Ian. 1995. "Early Books, RET Encoding Guidelines, and the Trouble with SGML". At http://www.ucalgary.ca/~scriptor/papers/lanc.html

[Lie 1999] Lie, Håkon W. 1999. "Formatting Objects considered harmful". At http://www.myopera.com/people/howcome/1999/foch.html.

[McCarty 1999] McCarty, Willard. August 29, 1999. An "Analytical Onomasticon to the *Metamorphoses* of Ovid". On-line sampler. At http://ilex.cc.kcl.ac.uk/wlm/onomasticon-sampler/.

[Quin 1996] Quin, Liam. November 1996. "Suggestive Markup: Explicit Relationships in Descriptive and Prescriptive DTDs". SGML'96. Graphic Communications Association.

[Reid 1998] Reid, Brian. 1998. Keynote address to Markup Technologies '98.

[Renear 2000] Renear, Allen. 2000. *The Descriptive/Procedural Distinction is Flawed*. Extreme Markup Languages 2000. Reprinted in*Markup Languages: Theory and Practice*, 2, no. 4 (Fall, 2000).

[Rockwell 2001] Rockwell, Geoffrey. February 20, 2001. Private e-mail to the author.

[Goldfarb 1990] Goldfarb, Charles F. 1990. *The SGML Handbook*. Oxford: Clarendon Press. Annex A. Adapted from Charles F. Goldfarb, "A Generalized Approach to Document Markup", in *SIGPLAN Notices*, June 1981.

[Sperberg-McQueen 2000] Sperberg-McQueen, C.M., Claus Huitfeldt, and Allen Renear. "Meaning and Interpretation of Markup". Extreme Markup Languages 2000.

[XML 2000] Bray, Tim, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, eds. 6 October 2000. "Extensible Markup Language (XML) 1.0 (Second Edition)". W3C Recommendation . At http://www.w3.org/TR/2000/REC-xml-20001006