Extreme Markup Languages 2006®

Montréal, Québec
August 7-11, 2006

A natural-language approach to modeling
Why is some XML so difficult to write?

Yves Marcoux
GRDS - EBSI

Abstract

Writing valid XML can be viewed as a collaborative process in which (roughly speaking) the modeler supplies the structure (markup), and the author the contents. When an information management chain includes document creation by a human, failure to mobilize and properly support the author in his task may result in errors or loss of valuable information. In this paper, we first argue that the usual pragmatical approaches to specifying the semantics of XML models do not allow authoring environments to easily provide sufficient *semantic* support to authors, whereas *syntactic* support is profusely available. Then, we sketch a semantic framework (provisionally called *intertextual semantics*), which we think could allow modelers to specify the semantics of their models in a form that can be turned into semantic support to authors in authoring environments. We discuss the pros and cons of the proposed framework, as well as avenues for further work.

A Conference of IDEAlliance
Extreme Markup
2006 Languages®
Montréal, August 7–11, 2006

A natural-language approach to modeling

Why is some XML so difficult to write?

Table of Contents

A Conference of IDEAlliance
Extreme Markup 2006 Languages®
Montréal, August 7–11, 2006

A natural-language approach to modeling
Why is some XML so difficult to write?

Yves Marcoux

§ Introduction and overview

Writing valid XML is a collaborative process

It is amazing how differently the same piece of information can be represented in digital objects. Even restricting ourselves to the realm of structured documents, the fact that a person is a male might be represented in any of the following ways:

1. `<person sex="M">John...`
2. `<male.person>John...`
3. `<sex><male /></sex>`
4. `<applicant-is-female>FALSE</applicant-is-female>`
5. `<gender gender="&#x2642;" />`
6. `<property propNum="3" target="p007" value="1" />`
7. `<note>Oh, and, BTW, it's a boy!</note>`

And those are just a few of the possibilities we can imagine. By the way, the character at Unicode code point U+2642 is "♂", known as `&male;` in the `ISOpub` entity set.

If an insurance company agent is jotting notes on paper while talking on the phone to a policy applicant, he might take note of the fact that the applicant is a male in any way that best fits his state of mind at the moment. If, by any chance, he is jotting notes in well-formed XML on a computer, then he might choose on-the-fly any of the above possibilities, or some other one. But if he is entering the fact in valid XML, then the choice of an appropriate representation has been made beforehand for him: by the *modeler*.

Seen from this angle, writing valid XML is a collaborative process, in which (very roughly speaking) the modeler supplies the structure (markup), and the author the contents; much like a filled-out form is the result of collaborative work between the form designer and the person who fills it out. As any collaborative process, writing valid XML has to be supported by adequate communication, so that the "intentions of the modeler" (for example, what should be entered in such and such an element or attribute, and how) are understood and taken into account by the author. Failure or weaknesses in that communication result in any or all of Maler & El Andaloussi's *tag abuse syndrome* [Maler & El Andaloussi 1996], semantically poor documents, misinterpretations, errors, and inaccuracies.

Modeler-author communication

So how are the "intentions of the modeler" communicated to the author, and how can that communication be facilitated? This is the main object of our reflection. More specifically, we want to investigate how that communication is affected, not only by the characteristics of the model itself, but also by peripheral artefacts, such as model documentation and editor/authoring environment configuration. Through this investigation, we hope to derive guidelines—and eventually, tools—to help modelers develop models (schemas, DTDs, etc.) *and* peripheral artefacts that, together, facilitate the communication to authors of the modeler's intentions.

Obviously, we are mainly interested by situations in which a human author is involved. Some XML is "authorless", in the sense that it is produced automatically, without any human intervention (sensor data, for example). We do not address those situations directly; however, we believe that the kind of reflection conducted here may still be relevant, because it suggests a semantic framework that might turn out to be helpful in making a model comprehensible not just authors, but to anybody involved in processing and interpreting conforming XML instances.

Just as a structured-document model has two parts, syntax and semantics, so have the intentions of the modeler. As we shall try to demonstrate, in actual authoring scenarios, the authoring environment is much more efficient at conveying to the author the *syntactic* intentions of the modeler than her *semantic* intentions. We claim that this situation is *for lack of a suitable semantic description of the model*. Our goal is to suggest, and briefly sketch, a semantic framework, which we provisionally call *intertextual semantics*, in which the semantics of a model can be described in a way that is amenable to communication to the author at authoring time.

Related work

Standard modeling methodologies (for example, [Travis & Waldt 1995] [Maler & El Andaloussi 1996] [Glushko & McGrath 2005]) do not in general include a full-fledged formal semantic framework. The semantic aspects of modeling are rather treated in pragmatical terms through discussion of *human-readable documentation* and *application development*. We call those two approaches the *pragmatical* semantic frameworks of structured documents, and review them in Section "Pragmatical semantics for structured-document models". Semantic properties and usability of models and of conforming documents, though recognized as crucial, are side-effects of those pragmatical considerations.

Some formal semantic approaches have been proposed in the past to provide semantics to structured documents. Renear, Dubin, and Sperberg-McQueen [Renear et al. 2002] provide a historical background and a description of a specific project: BECHAMEL. Judging from their paper, the general premise is that natural-language descriptions are insufficient and must be complemented by a separate formal apparatus:

> What is needed is a mechanism that would allow the markup language designer to rigorously and formally specify semantic relationships; these specifications could then be read by processing applications which would configure themselves accordingly, without case by case human intervention.
>
> [Renear et al. 2002], p. 122.

While we share that point of view, the "processing applications" we have in mind are very specific: document authoring. That is why we do not follow the same track as [Renear et al. 2002]. In a way, we could describe our approach as trying to operationalize natural-language descriptions is such a way that they can support document creation. We do not replace natural-language, we frame it with mechanisms that make it supportive of the document creation process.

The semantic approach that seems closest to ours has been introduced by Wrightson [Wrightson 2001] [Wrightson 2005] [1]. Wrightson used *situation semantics* [Devlin 1991] to analyse, among other things, human legibility of XML *well-formed* documents. Although we are interested here only in valid documents, there seems to be a lot in common between Wrightson's preoccupations and ours. For example, we also derive explanations for some of the legibility phenomena analyzed by Wrightson.

Wrightson's approach seems to be more *explanatory*, whereas ours aims to be more *prescriptive*, in the sense of being deployable right at document-authoring time, and of suggesting modeling practices more directly. We concentrate on the *writing* of *valid* XML documents, hence on the communication between modeler and author at document-creation time, direct human legibility of documents being only a possible side-effect. A thorough and systematic comparison between Wrightson's approach and ours would be interesting, but it is not the purpose of this paper. Our goal is to sketch a first draft of a semantic framework and argue that it has the potential of being helpful in the context of modeling methodologies. A thorough development of the framework, together with a precise comparison with existing frameworks, would be natural future work.

Another approach with which we have a lot in common is that of Sperberg-McQueen, Huitfeldt, and Renear, presented at Extreme 2000 [Sperberg-McQueen et al. 2000]. The authors develop a framework for structured-document semantics based on *sentence skeletons* and *deictic expressions* (a generic concept of which XPath relative expressions are a specific case). Although similar concepts can be found in our framework (or in possible extensions), there are two important differences between our approaches: First, although the possibility of using natural-language sentence skeletons is mentioned in [Sperberg-McQueen et al. 2000], the bulk of the discussion—and all examples—use Prolog predicates. Second, in [Sperberg-McQueen et al. 2000], the primary focus is not on document authoring, but rather on inferences ("licensed"—or legitimized—by the markup) that can be made by the readers of a document. Note that the set of inferences licensed by a piece of markup can be considered to be a description of its semantics (a point of view explicitly adopted by Sperberg-McQueen et al., on p. 233 of their paper). As such, one could hope to use this set for providing semantic support to authors at authoring time. However, a set of inferences might not be the most appropriate semantic description in this context: for one thing, the set might be infinite; even if it is not, it might be hard to compute and/or understand. In a way, the natural-language "sentence skeletons" (combination of **text-before** and **text-after** segments; see Section "Intertextual semantics") that we associate with a piece of markup in our framework can be viewed as conveying the *facts*—in a very general sense—licensed by that piece of markup, facts from which readers will later be able to make inferences.

About this paper

In the remainder of this paper, we will:

1. Review the two common pragmatical approaches to specifying the semantics of structured-document models (Section “Pragmatical semantics for structured-document models”).
2. Review the most common mechanisms currently used to support modeler-author communication in an authoring situation, and argue that they favor the syntactic part of models over their semantic part (Section “Support for modeler-author communication at authoring time”).
3. Introduce a semantic framework, which we provisionally call *intertextual semantics*, and which might help in communicating the semantics of a model to an author in an authoring situation (Section “Intertextual semantics”).
4. Discuss the advantages and disadvantages of the envisioned semantic framework, and its consequences on modeling (Section “Discussion”). We also consider possible uses outside the context of modeler-author communication.
5. Wrap up and discuss avenues for future work (Section “Conclusion and future work”; ideas of future work are also found elsewhere in the paper).

§ Pragmatical semantics for structured-document models

Modeling is often envisioned as the activity of establishing “digital information containers” with shapes best suited to conveying information in a given type of situations. Thus, as an activity of syntactic nature. But modeling also involves specifying the *semantics* of the containers (what it means to put data in some part of the container, as opposed to some other part). As Travis & Waldt [Travis & Waldt 1995] and Maler & El Andaloussi [Maler & El Andaloussi 1996] remind us, Goldfarb's original definition of a DTD *included* the semantics of markup as an essential part ([ISO 8879] and [Golfarb 1990]). Gradually, the concept of DTD became associated with only the syntactic rules; nevertheless, the semantics of a model is as important as its syntax, because it determines how people derive meaning—or make sense—out of conforming instances.

Maybe one reason why the semantics aspect of models is sometimes taken for granted comes from the “human-readability” of XML: since each part of the containers is labeled with a name (e.g., element or attribute name), the modeler may be led to consider that picking appropriate names is enough to make the semantics obvious and self-explanatory. However, the semantics conveyed by the syntactic declarations alone does not in general suffice, and some form of semantics is usually supplied by the modeler, over and above the syntactic declarations.

Two pragmatical approaches to structured-document semantics are *human-readable documentation* and *application development*. We review them here because they will be useful as references in the presentation of our own approach.

The human-readable documentation approach

Goldfarb's definition did not specify exactly *how* the semantics ought to be presented. Strictly speaking, the only mechanisms explicitly available in XML to support the specification of semantics *right into the models* are comments (in DTDs and schemas) and `documentation` elements (in schemas). Although nothing would prevent those from containing machine-readable material, their typical use is for human-readable documentation.

As it were, standard modeling methodologies (for example, [Travis & Waldt 1995] [Maler & El Andaloussi 1996] [Glushko & McGrath 2005]) all present some form of human-readable documentation (whether embedded in the models or separate) as *the* principal means of specifying semantics.

Not suprisingly, all models developed with an ambition of widespread use (which includes proposed and actual standard models), come accompanied with more or less lengthy human-readable “guidelines”: for example, TEI and EAD both have guidelines, MARC has its *Anglo-American Cataloguing Rules*, each totalling many hundred pages. These guidelines are considered as the “Bible” of their respective models, and can legitimately be regarded as *defining* their semantics. For example, they serve as bases for the development of applications.

It is worthwhile noting that, through this mechanism, semantic descriptions can be very precise or very loose, as appropriate. Indeed, since they are expressed natural language, the whole range of precision of that vehicle is available. Thus, a `date` element could be defined very loosely as simply "date" (as, for example, in basic unqualified Dublin Core), or else, very precisely as "date on which the writing of the memo has begun, regardless of the date of its completion and/or sending".

We should point out here the general shape that human-readable documentation usually takes. Typically, it is structured in two rather different sections:

1. a first section that describes the model in general, its underlying goal, application area, and philosophy; and
2. a section that gives a specific description for each name (element or attribute name) in the model.

It should also be noted that the syntactic aspects of the model (such as the content models, data types, and special writing rules) are usually incorporated in the human-readable documentation, even if they might be redundant with the declarations. Thus, the human-readable documentation is usually a complete description of the model, syntax and semantics.

The application approach

Another approach to semantics is that *applications*, and only applications, give semantics to XML markup. That point of view, while certainly legitimate, poses a number of problems that must be dealt with if we want to use it as a semantic framework:

1. By the very goal of reusability inherent to the structured-document approach, it is entirely possible to have many different applications working on the same documents, even applications entirely unthought of at document-authoring time (let alone modeling time). It is thus necessary to identify one specific application (or, in the worst case, a finite set of applications), that must be implemented and operational, before we can precisely talk about the semantics of a document.
2. Applications are usually only guaranteed to work on complete and valid documents. Thus, partial checking (computation) of semantics cannot in general be performed on incomplete, not yet valid documents, or on document fragments.
3. Even if applications are designed to work on document fragments or incomplete/invalid documents, the semantics they give rise to is in general *non-compositional*. Intuitively, that means that the semantics of a document fragment does not necessarily occur as a fragment of the semantics of the whole document. [2]

   HTML and Web browsers provide a perfect example of this phenomenon (browsers, by the way, are usually "tolerant" applications, that accept incomplete or invalid documents). Suppose we define the semantics of HTML by its rendering in a browser. Thus, the semantics of a `<p>` (paragraph) element would be defined as the visual rendering of the paragraph in a Web browser. But if, in a complete document, that `<p>` element is included in a `<del>` (deleted passage) element, its rendering changes: typically, it will be rendered with strike-through text. Thus, the semantics of the fragment is not found intact as part of the semantics of the whole document.
4. Since applications usually (if not always) perform a final rendition of the information (for example, visual or aural), it might be difficult, just by looking at the output of the application, to distinguish what is essential in the semantics, from what is accessory (e.g., font size, colors, etc.). Thus, that information would have to be given separately from the application (and would most likely be given in human-readable form).

§ Support for modeler-author communication at authoring time

Authoring environments vary widely, and can, among other characteristics, be tailored to specific models. But typical environments, such as XML editors XMetal (developed by SoftQuad and now marketed by XMetal, Inc.) and oXygen (from SyncRO Soft), have strong similarities in their support of the modeler-author communication. We base our discussion on such typical environments.

The following inventory of resources and mechanisms is relevant for us, because it will serve as a kind of "check-list" for what we include and do not include in our semantic framework.

Background material

Obviously, modeler-author communication can be direct, even at authoring time. An obvious case is when author and modeler are the same person. Another case is when the author has direct “online” (live or virtual) access to the modeler, and can interact with her. We are not interested by those cases. Rather, we are interested by how modeler and author can communicate *through artefacts* that the modeler has left behind her, and only through those artefacts.

The artefacts do not have to be plain and dull static resources: for example, a session in which the modeler interactively explains her model to some authors may have been videotaped, and be available to the author. Or, an interactive, personalizable tutorial may have been developed (with the direct or indirect collaboration of the modeler) and be available to the author. We *want* to cover those cases. The only thing we require is that the resources be adressable (e.g., possess a URI), be storable on read-only memory (i.e., possess a finite description), and be amenable to human consumption. This, we should point out, includes good old paper books (published or not).

All of those resources, we call *background material* for the model. Normally, at authoring time, all of it should be available to the author, either directly from the authoring environment, if it is an online resource and the authoring environment can link to it, or separately from the authoring environment otherwise. If some of the background material is not available to the author (for example, if part of it is commercial, and is too expensive for the author), the problem is a model-deployment one and, as such, does not interest us (although it might dramatically impact the effective use of the model).

Resources directly available in the authoring environment

We first enumerate resources that are typically available directly in the authoring environment, and later look at the mechanisms used to deploy them:

- From the syntactic part of the model:
  1. Names of elements, attributes, and entities (if applicable).
  2. The replacement text of general entities (if applicable).
  3. Allowed values for enumerated attribute types.
  4. The content models for each element (global or local) in the model.
- From the customization of the environment (may be provided by the modeler):
  1. Developed names, legends, that may be used in the user interface instead of the actual names in the model.
  2. Examples / templates of valid / expected contents.
- From the documentation of the model provided by the modeler:
  1. Element-specific parts of the documentation (typically shown as contextual help or in “tool tip” pop-ups).

     It is interesting to note that the general introductory part of the documentation is usually *not* available directly from the editing window (unless if it is presented as element-specific documentation for the top-level element of the model). If it is available from the authoring environment, then it is usually as background material, from outside the editing-window.

Mechanisms typically available in the authoring environment include:

- Syntactic validation (and displaying of error messages).
- Access to background material.
- From the application semantics of the model (if any): in-application previews, such as XSLT processing.

The above mechanisms are usually available on demand, in application mode (i.e., as a sort of “browser preview”), and not directly “in the editing-window”. The following ones, for their part, are usually available right in the editing-window:

- Suggestions of structurally valid contents to be inserted at the insertion point.

- Insertion of examples/templates of valid/suggested contents at the insertion point.
- Automatic insertion of mandatory sub-elements when a new element is inserted.

In the editing-window: a conversation

The point we want to make here is that what is really happening in the editing-window can be viewed as a *conversation* between the modeler and the author. But it is a *syntactic* conversation.

This mainly comes from the fact that content models are not only used for validation on an on-demand basis; they are also used to *suggest* what comes next in the document at the cursor position. Thus, for instance, if a (DTD) content model reads as `(annex*, appendix*)` and the cursor is positioned after the last annex entered so far, somewhere in the interface, a menu of some sort will be asking the author: You may add another annex, or start with appendices; what do you want to do? To which the author will respond by indicating his choice.

Intuitively, our idea is to turn that conversation into a *semantic* one. So, instead of asking: What *container* do you want to use, from the ones allowed, we would ask: What is it you want to *say* at this point, from the different possibilities offered to you by the modeler? But, instead of doing so in a question-answer mode, we want the "conversation" to take the appearance—as much as possible—of a joint (modeler-author) elaboration of the document.

§ Intertextual semantics

The framework we have in mind is provisionally called *intertextual semantics*. We do not give a complete development, but rather sketch it and illustrate it through examples.

In a sense, the semantic framework we suggest could be viewed as a partial formalization of the rather informal notion of *intertextuality*, introduced in philosophy and literary studies to capture phenomena of interrelationships and interdependences observed among artefacts of human textual production. The term "intertextuality" was coined in 1966 by Bulgarian philosopher Julia Kristeva. A large body of literature exists on the subject, but, to our knowledge, none of it provides an operational formalization of the notions involved. Hypertext and the Web are often presented as examples of intertexuality, and do formalize part of the notions. It is not surprising that our semantic framework is reminiscent of hypertext, and actually includes its most important concept: the hypertextual link. The formalization we are seeking, however, is, in a way, specialized to modeler-author dialogues.

The most unusual aspect of the projected framework is that it uses *natural language* as the basis of its semantic domain (in contrast, most semantic frameworks use more or less "artificial" formalisms as semantic domains, such as first-order logic). This may sound strange, and for some people, this may not be semantics at all. But we think it is justified by the fact that we want to apply the semantics *upstream*, not *downstream*, from document creation. We do not aim at performing any automatic "understanding" or processing of the document, or inference based on it. We just want to accompany the author in the task of writing valid XML; thus, it is plausible—at least *a priori*—that a framework in which the base elements are bits of natural language could be useful: those bits can be assembled in certain ways, and, by their very nature, run a fair chance of being understood by an author (or some other human, for that matter).

The framework is based on natural language segments, that can be thought of—and could concretely be implemented—as finite strings over any reasonable alphabet, say Unicode. But the details of an eventual implementation are not the crucial aspect. The important point is that those segments contain *text intended to be interpreted by humans*. Also, any formatting characteristics that may be given to that text in an eventual implementation (font, color, size, etc.) lie outside of the framework. Thus the *range of the semantic function* is essentially plain, uninterpreted strings of characters over some alphabet, ready, though, for interpretation by humans.

The idea of using text-related techniques to improve systems design, though not widespread, is not new: Smith [Smith 1994], for example, wrote that "[t]alk, theorized as conversation and analyzed as discourse, may provide the models of interaction that we need, in order to improve the design of hypertext systems and to extend the reach of its applications" (p. 281). Applying semiotics in general—not just textual—is the approach to interface design adopted by De Souza [De Souza 2005]. Perhaps the best-known examples of text-related techniques for systems development are Donald Knuth's WEB system and *Literate Programming* in general [Knuth 1984], TEI's ODD (*One Document Does it all*; see for instance [Cover 2005]), and Sperberg-McQueen's SWEB [Sperberg-McQueen 1996].

Sketch and examples

We said the formatting characteristics of the text segments lie outside of the framework. However, for simplicity of exposition, we will consider that parts of the text are *recognizable* in some way (for instance, a different color, or font-style, etc.). Those parts correspond to the text supplied by the modeler, the rest, to the text supplied by the author. In our examples, we will present the recognizable parts (coming from the modeler) in *italics*.

Also, we will assume that some coding convention allows modeler-contributed segments to include hypertextual links, for example, presenting the destination address [between brackets], as in `[http://w3.org]`.

Key idea: The key idea is to oblige (well, encourage) the modeler to, for each element and attribute [3] of the model, express the human-readable documentation through segments of text which, when intertwined with the actual element contents or attribute value, make up a text that, for human readers *with proper background* (i.e., in some *target community*), constitutes the complete "intended meaning" of the filled-in element or attribute. In the present sketch of the framework, the semantics of elements (and attributes) is specified solely by attaching "text-before" and "text-after" segments to them. The idea is not that the resulting text be stylistically elegant or even grammatical, but simply as explicit and efficient as appropriate.

Example: The following example, adapted from Travis & Waldt [Travis & Waldt 1995] (p. 289), illustrates the relationship that may exist between structured information and a textual equivalent:

Table 1: Facts about some US cities

City	Population	Annual snowfall (inches)
Denver	850,000	23
Rochester	240,000	88
Palm Spring	48,000	0

As Travis & Waldt argue, that table conveys essentially the same information as the following paragraph:

> Here are facts about some US cities. The city of Denver has a population of 850,000 and an annual snowfall of 23 inches. The city of Rochester has a population of 240,000 and an annual snowfall of 88 inches. The city of Palm Spring has a population of 48,000 and an annual snowfall of 0 inches.

The authors suggest the following semantic model might be applicable (though it is not their final suggestion, which turns out to be a general table model, but that is not the point) [4]:

```
<!ELEMENT cities (city+) >
<!ELEMENT city (name, population, snowfall) >
```

(#PCDATA declarations are omitted, here and in all model examples in the paper.)

Note, in passing, how two crucial interpretation elements are missing from the model names: the fact that snowfall data is annual, and that it is given in inches. This, we think, is typical of models elaborated with little concern for semantic support to authors.

Let us carry out the modeling once more, but this time applying intertextual semantics. We start with (a slight variant of) the textual formulation, in which we identify the modeler's contributions:

> *Here are facts about some US cities. The city named* Denver *has a population of* 850,000 *and an annual snowfall of* 23 *inches. The city named* Rochester *has a population of* 240,000 *and an annual snowfall of* 88 *inches. The city named* Palm Spring *has a population of* 48,000 *and an annual snowfall of* 0 *inches.*

This is the target intertextual semantics in our example. The modeler starts her task by establishing that target semantics. There are of course many possibilities, but the modeler chooses one.

Note that we have been a bit more explicit than Travis & Waldt in explaining how we want the cities identified ("city *named*"). This is because we have in mind a dialogue with the author, and we want to make sure he knows what to type in at that point in the document.

Remember that in the present sketch of the framework, the semantics of elements (and attributes) is specified solely by attaching "text-before" and "text-after" segments to them. The modeler can achieve the desired semantics with the following "text-before" and "text-after" segments:

Table 2

Element	text-before	text-after
`cities`	"Here are facts about some US cities."	empty
`city`	" The city "	"."
`name`	"named "	empty
`population`	" has a population of "	empty
`snowfall`	" and an annual snowfall of "	" inches"

If this looks like an exceedingly simple stylesheet mechanism, well... it is! In fact, at least this first sketch of the framework could be implemented in a straightforward manner (but only in application mode, not dialogue mode; see "Integration in authoring environments" below) by a very restricted form of XSLT stylesheets. The point is not the complexity (or absence of it), but what the approach forces us to reveal explicitly about the information that has to be managed.

Now, in the modeling scenario we envision, the modeler would start with an empty **Element** column. That is, she would start with intertextual semantics and work her way towards markup. The idea being that the names chosen for elements could stand as reasonable "abbreviations" for the **text-before** and **text-after** segments.

Thus, in our example, we would probably be led to choose slightly more informational names for our elements, such as, for example: `facts-about-US-cities` instead of `cities` and `annual-snowfall-in-inches` instead of `snowfall`.

Of course, depending on practical constraints, other names could be picked for the markup, and the ones we came up with could be used as developed names in the interface. The important point is that, if we choose to let go some part of the semantics, we do it consciously.

With the syntactic mechanism (available in all XML editors) consisting in the automatic insertion of mandatory sub-elements when a new element is inserted, the authoring interface becomes similar to a fill-in sentence. Note that this is *not* the same as a database form, which typically would have only each element (field) name displayed next to an input box. This, in some cases, can yield what Wrightson calls "quasi-natural language", but when it does, it is usually by accident. Our approach is entirely deliberate.

Another example: What would be an appropriate intertextual semantics for the HTML `<em>` (emphasis) element? Of course, this is debatable, because HTML semantics is not actually defined by intertextual semantics. Still, we venture to say that the following is a reasonable candidate:

Table 3

Element	text-before	text-after
`em`	" (and this should be emphasized) "	empty

Imagine that support for that semantics were implemented in an editor. Would it not be unnatural for an author to then "abuse" the tag and use it for, say, formatting mathematics? It would also be unnatural to start a sentence with an `<em>` element, which, arguably, one should not do.

We are not suggesting that all markup should be fully worked out in grammatical text. However, we do claim that, *if the modeler so desires*, intertextual semantics allows her to do it.

The possibility of hypertextual links in modeler-contributed segments is to allow pointers to external material, in case the appropriate semantic precision and/or richness can only be achieved through reference to such material (for example, a glossary of specialized definitions). For general background or training material pertaining to the model as a whole, a pointer can be included in the **text-before** of the top-level element.

Note that intertextual semantics is *not* a proposed final presentation for a document, nor is it meant to include everything that would typically appear in a rendering interface. For example, in an actual presentation of the city data used earlier, the cities would likely be sorted in some order (say, alphabetical), and that fact might be explicitly mentioned in the presentation interface; however, this has no place in the intertextual semantics.

A more complex example

The main purpose of this example is to demonstrate that even the extremely simple form of the framework presented above is not as limited as it first appears to be. We develop intertextual semantics for a general —though very simple—table model. It is a layout-oriented model, in that it is not linked to any particular contents. It is a simplication of the HTML table model:

```
<!ELEMENT table (caption, hr, tr+) >
<!ELEMENT hr (th+) >
<!ELEMENT tr (td+) >
```

Intuitively, `hr` stands for “header row”; other elements are as in HTML. Our city data would correspond to the following instance:

```
<table>
  <caption>Facts about some US cities</caption>
  <hr><th>City name</th><th>Population</th><th>Annual snowfall (inches)</th></hr>
  <tr><td>Denver</td><td>850,000</td><td>23</td></tr>
  <tr><td>Rochester</td><td>240,000</td><td>88</td></tr>
  <tr><td>Palm Spring</td><td>48,000</td><td>0</td></tr>
</table>
```

Now, the intertextual semantics can be as follows:

Table 4

Element	**text-before**	**text-after**
`table`	"¶This paragraph presents "	"¶"
`caption`	"“"	" .” "
`hr`	"Each sentence in the remainder of the paragraph presents information elements pertaining to one entity; elements within each sentence are presented in the following order:"	". "
`th`	" “"	"”"
`tr`	"Information elements for next (or first) entity: "	". "
`td`	" “"	"”"

The “¶” symbol represents a new line character. Note that our use of “ ” as delimiters is not problematic, even if either character were found in the data, because of our convention that modeler-contributed segments are recognizable as such (in italics in our examples).

The semantics of the whole `<table>` is thus:

> ¶*This paragraph presents “*Facts about some US cities*.” Each sentence in the remainder of the paragraph presents information elements pertaining to one entity; elements within each sentence are presented in the following order: “*City name*” “*Population*” “*Annual snowfall (inches)*”. Information elements for next (or first) entity: “*Denver*” “*850,000*” “*23*”. Information elements for next (or first) entity: “*Rochester*” “*240,000*” “*88*”. Information elements for next (or first) entity: “*Palm Spring*” “*48,000*” “*0*”.*¶

You might be thinking: “This is just a dumbed-down version of the raw XML”, or “This is no better (and maybe worse) than comma-delimited”, or “At that point, why not simulate a table with tabs and spaces?” The point is that the raw XML, like the other alternatives mentioned, rely on tacit interpretation conventions, whereas the above prose relies on nothing else than (maybe educated) English comprehension (and, of course, basic knowledge about cities and climate, which the target community is assumed to possess, in addition to English comprehension).

Why does this work, albeit very awkwardly? We can think of three reasons (which are just intuitions): (1) natural language can serve as its own metalanguage; (2) natural language has an extremely high level of *affordance* (property of suggesting spontaneous proper usage of itself) with humans; and (3) natural language can be “typeless”, in that natural language segments can contain *within themselves*—rather than in external metadata—indications (explicit or implicit) of their own *genre*. (Those three properties are not entirely independent of one another.)

Notice, for instance, how some of the segments in the example act as definitions. They set up conventions that the reader (or author) must follow. Of course, this constitutes a cognitive load on the user (author, reader, etc.). *We think the complexity of the prose equivalents to the structures of a model give some idea* *of the cognitive efforts involved in decoding and understanding those structures.* More on this in Section “Discussion”.

A final note: had we started with the prose and worked our way towards markup (with a general table structure in mind, of course), we would undoubtedly have come up with different names, and maybe different content models. But this phenomenon has been illustrated in a previous example, so we need not elaborate on it here.

A last example

This example involves again a general layout-oriented table model. But now, we suppose our target community includes only HTML experts, who fully understand the HTML table model. Thus, to them, a raw HTML table is just as informative (though maybe not as user-friendly) as any equivalent prose presentation. In that situation, a possible approach is to have an “identity” intertextual semantics, i.e., one that maps start- and open-tags to themselves (including angle brackets). The only issue here is to make sure the user (author, reader) is aware that the passage is an HTML table. This could be achieved through a namespace declaration (and then, assuming the users understand the associated conventions), or with the following intertextual semantics for the `<table>` element (all other elements having an “identity” intertextual semantics):

Table 5

Element	text-before	text-after
`table`	"The following is an HTML table: <table> "	</table>

Forcing ourselves to write out this degenerate semantics has at least the virtue of making explicit our reliance on the prerequisite knowledge we take for granted on the part of users.

If we do not wish to take such prerequisite knowledge for granted, a “lazy” solution would be to change the intertextual semantics of the `<table>` to this:

Table 6

Element	text-before	text-after
`table`	"The following is an HTML table (if you are not familiar with HTML tables, please learn about them by consulting [http://www.w3.org/TR/html4/]): <table>"	</table>

Again, forcing ourselves to write out this semantics has the virtue of making explicit the (maybe unacceptable) burden we place on users.

Integration in authoring environments

We noted previously that intertextual semantics would be easily realized by simple XSLT stylesheets. However, at present, this would make the semantics only available as an “application semantics”; thus, on demand, outside the editing-window. True integration in authoring environments would make intertextual semantics available *right in the editing-window*, so the conversation taking place there could really be a semantic one.

More precisely, what we have in mind is offering the author a *continuum of explicitness* of what the choices he has to make really mean, right in the editing-window. At one end of the continuum, we would have the raw XML, with only element and attribute names interspersed with the author's contributions (equivalent to the “text view” of a typical XML editor). At the other end, we would have full intertextual semantics. In-between, we would have, for example, views in which element names are replaced by more developed ones. The key point is that, each new level of terseness should be a reasonable “abbreviation” of the previous level.

§ Discussion

In this section, we discuss the advantages and disadvantages of intertextual semantics, specially for supporting modeler-author communication at document-creation scenarios, and also the consequences that this approach has on modeling itself. We also consider possible uses outside the context of modeler-author communication.

Expliciteness of complexity

We think the main advantage of intertextual semantics is that it makes explicit the complexity of the structural constructions used in a model. It forces the modeler to either make explicit the fact that some competences are taken for granted on the part of the user (author, reader), or provide in the semantics itself all the required explanations. Moreover, by allowing explanations to link to separate external material, it forces the modeler to explicitly provide possible "learning paths" for users who, though they belong to the target community, need training / learning in order to properly interpret some of the structural constructs of the model.

It is this explicitness that we believe makes intertextual semantics suitable for providing adequate semantic support at document-authoring time.

Compositionality and sequentiality

As mentioned earlier, we use a very restrictive form of compositionality: *compositionality with respect to string concatenation*. This is a very strong restriction, one consequence of which is that *distribution* of semantic features ([Sperberg-McQueen et al. 2000]) from parent to children elements is not possible. Whatever "statement" we want to hold for the whole of a parent element, including its subelements, must be placed in the **text-before** of the parent, and must be formulated in such a way that its scope (the whole element) is clear. As restrictive as this might be, we believe this form of compositionality is an advantage, because it makes explicit the cognitive loads that such conventions (distribution of properties) place on users dealing with instances.

Implicit in our use of strings and concatenation is the *sequential* nature of intertextual semantics. Here again, we consider this to be a quality, because we believe many (if not most) reliable and robust sense-making processes in humans have a sequential character (this being of course just an intuition).

Intertextual semantics, at least in the form presented above, is compositional and, thus, sequential. Because we consider those properties to be beneficial, we would strive to preserve them (though not at any price) in any future extension of the framework.

When and how thoroughly should it be done?

Writing out an intertextual semantics for the various elements of existing DTDs or schemas quickly becomes a kind of game. It is sometimes a tricky exercise of stylistics, but it is satisfying once accomplished. It gives the impression that something crucial about a structural construction has been captured.

Of course, specifying useful intertextual semantics for a model (specially models that may be partly layout- or processing-oriented, e.g., a general table model) is an investment. It will likely require a lot of energy. Intuitively, it is going to pay off more for beginning authors than for routined ones, who have developed a "sense" for the model. So maybe the question is how often new authors are going to be confronted to the model. Note that an expert author who has not used the model for a while may temporarily regain the novice status, and thus, benefit again from the kind of accompaniment provided by the intertextual semantics.

Another point is how much *reuse* is going to occur with the model, i.e., how often new applications exploiting conforming documents are going to be developed. Each time a new application is developed, the developers have to understand thoroughly at least part of the model. Thus, even if those persons are not authors, they may benefit from the semantic precision provided by intertextual semantics (as needed and deemed appropriate). In fact, one can ask the question of whether intertextual semantics can also be useful for explaining the model to other people than authors, for example readers and application developers.

A situation in which the potential precision of intertextual semantics may be useful is the case of legal documents (contracts, etc.) that may be filled-out and/or rendered on a range of devices (specially portable ones), and that may thus take a wide range of physical appearances, as much for authors as for readers. Then, there might be a need for a reference, conventional "meaning" of the document, something that could stand as the "face value" of the document. Intertextual semantics may play that role.

Consequences on modeling

The examples presented earlier suggest it is more complicated to specify (useful) intertextual semantics for models that are more processing- or layout-oriented than semantics-oriented (i.e., descriptive). It might be the case that using intertextual semantics encourages the development of semantic models. In fact, maybe the simplicity of the intertextual semantics of a model is a good indication of its degree of "semanticity". This is a question we would like to investigate.

We are not suggesting all processing- or layout-oriented markup should be banned. Again, intertextual semantics simply makes it possible to have available to the author the exact level of semantic richness and precision deemed possible/appropriate by the modeler. In some cases, processing-oriented semantics might be just what is needed. However, we conjecture that, in those cases, the reliance on the knowledge of the processing involved would be made explicit through intertextual semantics.

A side-effect of semantic precision may be the proliferation of semantically specialized elements in a model. This may be a bit of a problem for querying, but could be alleviated by searching in groups of elements, for example, elements that share a content model (or complex type), since elements that are slight semantic variations of one another are likely to have the same content model. Another problem caused by such proliferation could be name conflicts when going from one level of terseness to the other. Here, the possibility of *local elements* offered by W3C schemas is of great help: intertextual semantics would be associated to local, as well as global, elements. Two elements (at least one of them local) could have the same name and the same content model, but different intertextual semantics.

A new look at an old question

Why do mixed content models "feel good"? As is well known, mixed content models can be entirely removed from a model by the introduction of an extra element (`PCDATA`, `text`, or any other name), that has itself `(#PCDATA)` as a content model, and is used instead of `#PCDATA` in all mixed content models. Syntactically, this device does not change the expressivity of the model. Yet, most people feel that the resulting model is not quite right.

Intertextual semantics analysis allows us to give a precise formulation to the intuitive idea that the extra element is not only useless, but harmful. Indeed, the only natural intertextual semantics for that element is empty **text-before** *and* **text-after**. But then, the corresponding tags in the raw XML are not abbreviations of those texts.

Possible uses besides modeler-author communication

Could a semantic framework of the kind presented here be applied to the whole cycle of systems development: communication between the target community (the users) of the model and the modeler, and also, at the other end, between the author and the community (understandability of the documents by their target community)? It would be interesting to investigate that question.

Another much more direct possible use of intertextual semantics is in Natural Language Processing (NLP). Most systems that perform NLP for various purposes (e.g., automatic indexing, condensing, classification, segmentation) are unable to directly treat structured documents. To process such documents, all markup must first be stripped off. A very simple alternative—requiring no modification to the systems or algorithms—is to replace markup by appropriate intertextual semantics. Thus, tags are not simply removed, but rather replaced by periphrases (the modeler-contributed segments of the intertextual semantics) which preserve the natural language nature of the document contents. In that way, the original algorithms, capable of processing natural language, but not markup, can be applied directly.

§ Conclusion and future work

In this paper, we first argued that the usual "pragmatical" approaches to specifying the semantics of XML models do not allow authoring environments to easily provide semantic support to authors. Then, we sketched a semantic framework (provisionally called *intertextual semantics*), which we think could allow modelers to specify the semantics of their models in a form that can be turned into semantic support to authors in authoring environments. Finally, we discussed the pros and cons of the proposed framework.

A possible agenda for future work would be to:

1. Implement this first sketch of the framework with a XSLT mechanism in a real XML editor (in full dialogue mode).
2. Work out intertextual semantics for existing models.
3. Experiment in actual authoring situations.

Another avenue would be to enrich the framework. One thing we have yet to determine is how to integrate the consulting/insertion of examples/templates of valid/expected contents. It would also be interesting to integrate a dereferencing operation, that would allow for inclusion of multimedia and modular inclusion of text segments.

More powerful mechanisms than "text-before" and "text-after" for specifying the semantics should also be examined. However, this should be done with caution, because the semantics might then become non-compositional.

Notes

1. The subtitle of our paper echoes that of [Wrightson 2005]: *Why is some XML so difficult to read?*
2. We use a quite restrictive definition of compositionality: compositionality with respect to *string concatenation*. We can do that because we use character strings in both the intensional (syntactic) and extensional (semantic) domains. See Section "Discussion" for a discussion of why we use such a restrictive definition.
3. We do not discuss attributes thoroughly in this sketch of the framework. One way to treat them would be as subelements. A complete development of the framework would of course include idiosyncratic treatment of attributes.
4. We use the DTD formalism in our examples for simplicity. The ideas are applicable to other validation formalisms, such as W3C schemas.

Bibliography

[Cover 2005] Cover, Robin. *SGML/XML and Literate Programming.* **http://xml.coverpages.org/xmlLitProg.html**

[De Souza 2005] De Souza, C. S. *The semiotic engineering of human-computer interaction.* MIT Press, 2005.

[Devlin 1991] Devlin, Keith. *Logic and Information.* Cambridge University Press, 1991.

[Glushko & McGrath 2005] Glushko, Robert J.; McGrath, Tim. *Document engineering: analyzing and designing documents for business informatics and Web services.* MIT Press, 2005.

[Golfarb 1990] Goldfarb, C.F. *The SGML Handbook.* New York: Oxford University Press, 1990.

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

[Knuth 1984] Knuth, Donald. "Literate Programming." *The Computer Journal*, 27(2), 1984.

[Maler & El Andaloussi 1996] Maler, Eve; El Andaloussi, Jeanne. *Developing SGML DTDs: From Text to Model to Markup.* Prentice Hall PTR, 1996.

[Renear et al. 2002] Renear, Allen; Dubin, David; Sperberg-McQueen, C. M. "Towards a Semantics for XML Markup." *Proceedings of Document Engineering 2002.* **http://portal.acm.org/citation.cfm?doid=585058.585081**

[Smith 1994] Smith, C. T. "Hypertextual thinking." *In:* Selfe, C.; Hilligoss, S. *Literacy and Computers: The Complications of Teaching and Learning with Technology.* New York: MLA, 1994, pp. 264-281.

[Sperberg-McQueen 1996] Sperberg-McQueen, C. M. *SWEB: an SGML Tag Set for Literate Programming.* 1996. **http://www.w3.org/People/cmsmcq/1993/sweb.html**

[Sperberg-McQueen et al. 2000] Sperberg-McQueen, C. M.; Huitfeldt, Claus; Renear, Allen. "Meaning and Interpretation of Markup: Not as Simple as You Think." *Proceedings of Extreme Markup Languages 2000.*

[Travis & Waldt 1995] Travis, B.; Waldt, D. *The SGML Implementation Guide: A Blueprint for SGML Migration.* Springer, 1995.

[Wrightson 2001] Wrightson, Ann. "Some Semantics for Structured Documents, Topic Maps and Topic Map Queries." *Proceedings of Extreme Markup Languages 2001.*

[Wrightson 2005] Wrightson, Ann. "Semantics of Well Formed XML as a Human and Machine Readable Language: Why is some XML so difficult to read?" *Proceedings of Extreme Markup Languages 2005.*

The Author

Yves Marcoux
GRDS - EBSI
Université de Montréal
CP 6128, suc. Centre-ville
Montréal
Québec
Canada
H3C 3J7
yves.marcoux@umontreal.ca
http://www.mapageweb.umontreal.ca/marcoux/

Yves MARCOUX is a faculty member at EBSI [École de bibliothéconomie et des sciences de l'information], University of Montréal, since 1991. He is involved in teaching, research, standardization, and international cooperation activities in the fields of structured documents, information retrieval, database systems, and digital information management. Prior to his appointment at EBSI, Dr. Marcoux has worked for 10 years in systems maintenance and development, in Canada, the U.S., and Europe. He obtained his Ph.D. in theoretical computer science from Université de Montréal in 1991. His main research interests are document theory, structured document implementation methodologies, and information retrieval in structured documents. He is author of many research reports and scientific articles on various aspects of structured documents. Since 1995, he has led numerous projects related to XML and SGML theory and practice. He is sollicited as an expert on digital information management and structured documents on a regular basis. He has been co-responsible for the Digital Information Management Certificate at EBSI, from its creation in 2000, to 2005. Through GRDS [Groupe départemental de recherche sur les documents structurés], his research group at EBSI, he has been principal architect for the *Governmental Framework for Integrated Document Management*, a project funded by the National Archives of Québec and the Québec Treasury Board.

Extreme Markup Languages 2006®
Montréal, Québec, August 7-11, 2006
This paper was formatted from XML source via XSL
by Mulberry Technologies, Inc.