Intertextual semantics generation for structured documents: a complete implementation in XSLT

Yves MARCOUX

GRDS – EBSI – Université de Montréal

Mots-clés : sémantique intertextuelle, documents structurés, langages de balisage, XML, XSLT, descriptions formelles de jeux de balises

Keywords: intertextual semantics, structured documents, markup languages, XML, XSLT, formal tag-set descriptions

Résumé : La sémantique intertextuelle (SI) [1][4] attribue aux documents balisés un sens en *langue naturelle*. Alors que les sémantiques formelles visent une représentation du sens des documents pour la machine, la SI vise l'humain. Dans la forme actuelle de l'approche, la SI d'un modèle (DTD, schéma) est donnée par deux *péritextes* associés à chaque élément: un *texte-avant* et un *texte-après*. La SI d'un document est la concaténation des péritextes et des contenus d'élément dans l'ordre du document. Nous présentons une implantation complète, en XSLT 1.0, de la génération de SI. L'implantation traite les attributs tel que décrit dans [2], et les hyperliens et éléments locaux tel que décrit dans [1]. Elle indente aussi l'extrant pour une meilleure lisibilité tel que suggéré dans [3] et gère les exceptions que sont les éléments et attributs inconnus.

Abstract : Intertextual semantics (IS) [1][4] is a framework in which the meaning of marked-up documents is given in *natural language*. While formal semantics aims at conveying the meaning of documents to machines, IS aims at conveying it to humans. In the current framework, the IS of a model (DTD, schema) is expressed by *peritexts* associated with each element: a *text-before* segment and a *text-after* one. The IS of a document is obtained by concatenating peritexts and element contents in document order. We present a complete implementation, in XSLT 1.0, of the IS generation mechanism. The implementation handles attributes as described in [2], and hyperlinks and local element definitions as described in [1]. It also indents its output for increased readability as suggested in [3] and handles unknown element or attribute exceptions.

1 Introduction

In a structured document (XML, SGML, etc.), what is the "meaning" of the various tags (the *markup*) present in the document? How is the meaning of the document augmented—or otherwise affected—by the presence of markup?

Fundamentally, there are two possible avenues to give an answer to that question: the formal one and the informal one. One can devise a framework in which the meaning of a marked-up document is represented by a set of *formal* statements, for example in first-order logic. Or, one can seek a framework in which the meaning of a marked-up document is represented by a set of sentences in an *informal* language, for example a natural language.

If automatic inferencing (through an inference engine) is aimed at, then a formal approach probably has a leading edge. However, if some other use of the "meaning" of the document is envisioned, which for example involves showing that meaning to humans, then the situation may be reversed.

Formal Tag-Set Descriptions (see for example [6], [7] and [8]) are an example of the approaches along the formal avenue. *Intertextual semantics* [1][2][4] is an approach along the informal avenue. In intertextual semantics (IS), the meaning of a marked-up document is entirely and exclusively represented *in natural language*.

The intertextual semantics (IS) approach is based on the hypothesis (of which traces can be found in, among other places, the works of Wirzbicka [9], Smedslund [5], and even Wittgenstein [10]) that humans ultimately "make sense" of artefacts through the use of *natural language* (NL), and that in designing artefacts, one should be preoccupied by how, and how easily and with how much ambiguity (or unambiguity), humans can derive NL from those artefacts. No matter how useful intermediate formal representations of meaning (including marked-up documents) may be for conciseness, machine processing, etc., they must ultimately be translatable (not necessarily translat*ed*) to NL, and are ever only as "meaningful" as such NL expressions of them are.

In the realm of structured (i.e., marked-up) documents, IS suggests that the creators of tag-sets (modelers) should be preoccupied by how markup can be translated to NL. Even if "end users" never see any marked-up document, some other humans, for example, processing software developers, or archivists, will have to deal with them directly or indirectly, unless the documents are totally pointless. One might say it is even more important to be preoccupied by that translation as the number of intermediate representations increases, because there are then more opportunities for misinterpretations.

IS proposes a mechanism by which NL passages (or whole documents) are generated from marked-up documents, according to an *IS specification* for the tag-set. So far, only very weak NL generation mechanisms have been explored, *and it is extremely important that those mechanisms be weak*, because too powerful mechanisms would "hide under the carpet" inherent interpretation complications which IS, in contrast, seeks to uncover. In the current state of the IS framework, an IS specification takes the form of a table giving, for each element type two NL segments: a "text-before" segment and a "text-after" segment, generically called "peritexts."

Attributes require special attention, but a way of handling them in keeping with the spirit of IS is presented in [2]. They are handled through the possibility of including in the peritexts "guarded segments," segments guarded by an attribute name, that are only included if the corresponding attribute is specified on the element, and that can refer to the attribute value. "Local" elements (in the sense of W3C schemas) are supported, so that different peritexts can be assigned depending on the ancestors of the element.

The IS generation process is akin to styling the document with the peritexts, concatenating peritexts *and* element contents as the document tree is traversed depth-first. The *IS*, or *IS-meaning*, *of the document* is the resulting character string. It is important to stress that, in spite of the similarity between styling and the generation of the IS of a document, the preoccupations of IS are absolutely not at the *presentational* level, but really at the *semantic* level.

In this article, we present a complete implementation, in XSLT 1.0, of the intertextual semantics generation mechanism. The transformation is *model-independent* in that it reads the peritexts from an XML document encoding the IS specification for a given model. It implements attribute handling as defined in [2]; hyperlinks in peritexts or as attribute or element content, as described in [1]; and local element definitions, also as described in [1]. In addition, it performs indentation of the output (in the same line as [3], but more elaborate), for increased readability, and handles *exceptions*, elements for which no peritext exists in the IS specification and unexpected attributes.

2 General approach

As mentioned earlier, the implementation is model-independent: the same XSLT stylesheet is used to process any document. In principle, the association between elements and peritexts is determined by the model (DTD or schema) to which the document conforms. Knowing the model,

the generic stylesheet can read an IS specification (ISS) file, giving the peritexts for all elements, and compute the IS of the instance.
In theory, namespace or schema-location information could be used to identify the appropriate ISS file applicable to a document. However, complications would arise from the fact that the same document can conform to different schemas, and contain elements of different namespaces. Thus, it seems simpler to determine the model of a document for IS purposes independently from namespace and schema-location information. One possibility would be to point explicitly to the ISS file through a processing instruction in the document. We chose a more implicit approach, requiring no model-specific addition to the documents, and which proved flexible enough: the generic ID of the document element of the instance is used to form the filename of the ISS file. More specifically, the file named `genID.iss.xml` in the same directory as the generic stylesheet is used as an ISS file, where `genID` is the generic ID of the top-level element of the document.
The processing performed by the generic stylesheet is one-pass, i.e., it takes as input the document instance and directly generates its IS. Thus, no pipelining environment is necessary. Any current browser with an XSLT 1.0 processor can be used to view the IS of documents directly, provided a link to the generic stylesheet is included in the documents, for example:

```
<?xml-stylesheet type="text/xsl" href="ISG.xsl" ?>
```

3 IS specifications

3.1 Overview

Essentially, an ISS file gives the peritexts (text-before and text-after segments) for all the "elements" in the model. Remember, however, that local elements (in the sense of W3C schemas) are possible [1], so that different peritexts can be assigned to elements with the same generic ID but different ancestral lines. Thus, in effect, a peritext is not assigned to some fixed generic ID, but to a *path*, and will be applicable to elements *matched* by that path. A path P is said to *match* an element E iff E's ancestral line ends with P, i.e., iff P is a suffix of E's ancestral line. For example, a peritext assigned to path `title` is applicable to all elements with generic ID `title`, regardless of their ancestral line, but a peritext assigned to path `titleStmt/title` is applicable only to elements with generic ID `title` that are children of elements with generic ID

`titleStmt`. A peritext assigned to path `/TEI` is applicable only to document (top-level) elements with generic ID `TEI`.[1]
For convenience, peritexts can be assigned simultaneously to more than one path, and peritexts are in fact assigned to paths in *pairs*, consisting each of one text-before segment and one text-after segment.
Peritexts can contain certain delimiters for handling attributes as described in [2] and allowing hyperlinks in the resulting IS as described in [1]. The hyperlink delimiters in [1] were [ ]; however, in [2] and here, those are used for attributes. Thus, we will use {{ }} for hyperlinks.

3.2 ISS files

An ISS file is an XML document. All its elements and attributes belong to the specific namespace:

`http://grds.ebsi.umontreal.ca/ns/ISS/`

Its top-level element is an `iss` element. The content of that element is one or more `rule` elements. Each `rule` element is empty and has three mandatory attributes: `paths`, `text-before`, and `text-after`. The effect of a `rule` element is to assign the pair of peritexts `text-before` and `text-after` to the path or space-delimited paths given in `paths`.
A rule is applicable to an element iff one of its paths matches the element. If more than one rule applies to an element, the one with the most specific (longest) matching path is chosen; if more than one rule has that longest matching path, the first one (in ISS file order) is applied.
The sequences {{ and }} *in peritexts* are hyperlink delimiters, i.e., what is between them is interpreted as a URL and converted to a hyperlink in the IS. It is possible to have {{ in a text-before and }} in the corresponding text-after, but this will only work when the element contains neither sub-elements nor }} character sequences (which would be unusual in a URL).
Peritexts can contain passages "guarded" by an attribute name, such as:

```
@attribName[Some text containing exactly one @.]
```

Such guarded passages in peritexts are included in the resulting IS only if the guarding attribute is present on the element to which the peritext is applied. Otherwise, the entire guarded passage is omitted. When the passage *is* included, the actual value of the attribute is inserted in place of the `@`.
It is possible to use `xmlns` as an attribute to refer to the `namespace-uri` of an element. The guarded passage is then included only if the element belongs to a namespace.

1 Paths play a role similar to match attributes of xsl:template elements in XSLT. However, paths are deliberately much less flexible and cannot, for example, contain wildcards or refer to arbitrary XPath axes.

3.3 Examples

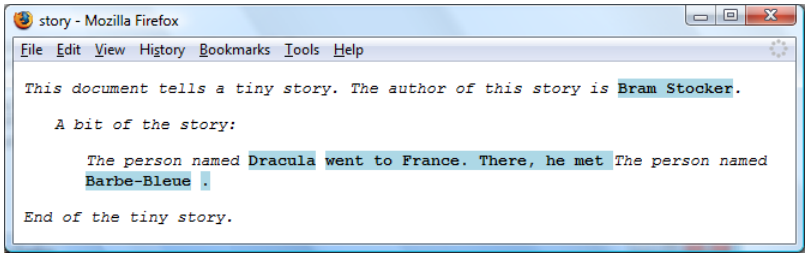
Here is the ISS file used for our examples. It is intended for a top-level element of `story`, and should thus be named `story.iss.xml` and reside in the same directory as the generic stylesheet:

```
<?xml version="1.0"?>
<iss xmlns="http://grds.ebsi.umontreal.ca/ns/ISS/">
<rule paths="story"
  text-before="This document tells a tiny story.@xmlns[ The
     document belongs to the XML namespace &quot;@&quot; (if
     you are not familiar with XML namespaces, you can read
     about them at {{http://www.w3.org/TR/REC-xml-
     names/}}).]@author[ The author of this story is @.]"
  text-after="End of the tiny story."/>
<rule paths="para" text-before="A bit of the story: " text-
     after=""/>
<rule paths="person" text-before="The person named "
  text-after=" @key[{{http://en.wikipedia.org/wiki/@}} ]"/>
<rule paths="place" text-before="The place named "
  text-after=" @key[{{http://en.wikipedia.org/wiki/@}} ]"/>
</iss>
```

Example 1 is the following XML document:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="ISG.xsl" ?>
<story author="Bram Stocker">
  <para><person>Dracula</person> went to France. There, he
     met
      <person>Barbe-Bleue</person>.</para>
</story>
```

Note that `ISG.xsl` is the generic stylesheet and it is here assumed to be in the same directory as the document. The resulting IS, as can be viewed in any XSLT 1.0-compliant web browser, is:

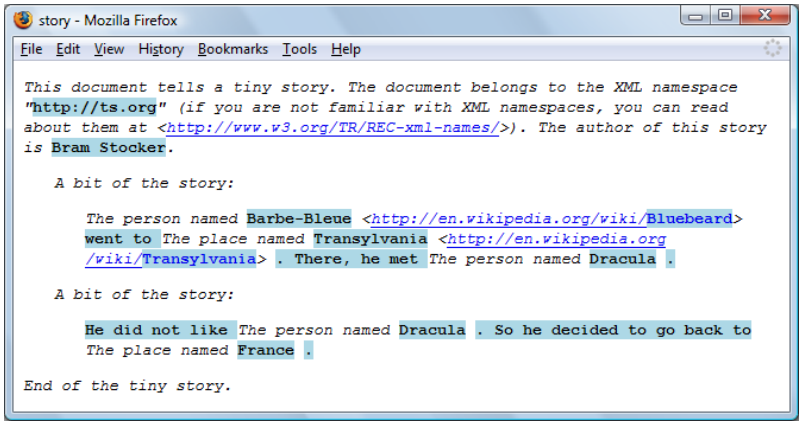


Note that the text contributed by peritexts is typeset in italics and the text contributed by the document is typeset in normal font on blue

background. This is in keeping with the philosophy of IS, which demands that the origin of all text in the IS of a document be clearly identifiable. Example 2 uses more of the richness allowed by the model:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="ISG.xsl" ?>
<story author="Bram Stocker" xmlns="http://ts.org">
  <para>
    <person key="Bluebeard">Barbe-Bleue</person> went to
    <place key="Transylvania">Transylvania</place>. There,
     he met
    <person>Dracula</person>.</para>
  <para>He did not like <person>Dracula</person>. So he
     decided
    to go back to <place>France</place>.</para>
</story>
```

This time, the resulting IS is:



Note that the `key` attributes (of both `person` and `place`) become part of a clickable hyperlink in the IS.

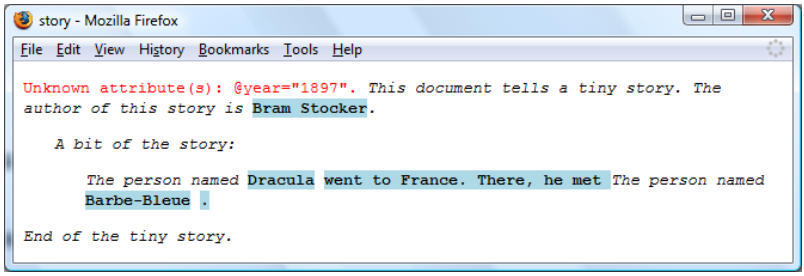
Both Example 1 and Example 2 exhibit a block structure with some parts indented. This feature, globally referred to as "automatic indentation," is not controlled by the ISS file, but rather realized automatically through heuristics. It will be discussed in more detail later.

Example 3 will illustrate exception handling. In the document of Example 1, we add to some element (**`story`**) an attribute (**`year`**) that is not mentioned anywhere in the peritexts of that element. This is considered to be an "unknown attribute:"

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="ISG.xsl" ?>
<story author="Bram Stocker" year="1897">
```

```
  <para><person>Dracula</person> went to France. There, he
     met
      <person>Barbe-Bleue</person>.</para>
</story>
```

The resulting IS is:



Note that exceptions are reported in red, so as to be easily differentiated from "normal" IS text. An element to which no rule in the ISS file applies is also an exception, of type "unknown element."

4 Structure of the generic stylesheet

The general structure of the generic stylesheet (`ISG.xsl` in the above examples) is as follows:

1. Global variables initialization.
2. Overall HTML structure template, matching /.
3. Templates for text-only elements.
4. Templates for all other elements.
5. Templates for text nodes (PCDATA).
6. Called templates for finding the best matching rule.
7. Called templates for processing attributes.
8. Called templates for processing hyperlinks.
9. Called templates for exception handling.

The rules contained in the model-specific ISS file *genID*`.iss.xml`, where *genID* is the generic ID of the top-level element of the document, are read in Part 1 and placed in a global variable. Part 2 produces the overall HTML structure of the output, including an internal CSS stylesheet.
Parts 3, 4, and 5 are actually pairs of templates: one for block formatting and one for flowed formatting. The *indentation heuristics*, mentioned earlier, is realized by the templates in Part 4 and determines how blocks are indented relative to one another and at which level the formatting should be changed from block to flowed.

The templates in Part 6 are used to determine the rule, in the ISS file, that "best" matches an element. As sketched earlier, a rule is a best match for an element E iff one of its paths matches E (i.e., is a suffix of E's ancestral line) and no other rule specifies a longer path matching E. If two rules or more are best matches for an element, the first one (in ISS file order) is chosen.
The templates in Parts 7 and 8 process attributes and hyperlinks, respectively. Processing consists essentially in recursive search-replace of various delimiters and placeholders. The templates in Part 9 are called by those of Parts 3 and 4 to verify the presence of exceptions and, if needed, include the appropriate warnings in the produced IS.

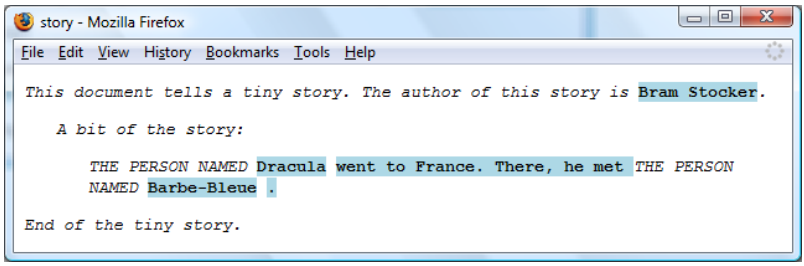
5 Discussion

The examples illustrate that peritexts can be very long. It is an essential feature of IS that there be no limit on their length. They should not be constrained lexically either. However, this is not entirely the case in the current implementation. Indeed, it is currently not possible to include some of the delimiters and placeholders for attribute guarded-passages and hyperlinks as data in the peritexts (and, in a few cases, in element content). One possible improvement would thus be to define and implement conventions for allowing all delimiters and placeholders to be included as data in peritexts (and element content).
A related issue is the validation of the syntax used for attribute guarded-passages and hyperlinks in peritexts. At the moment, no validation or error detection is performed. While this will never cause the abnormal termination of the transformation, it could yield unexpected results. Another possible improvement would thus be to implement full syntactic validation of the peritexts.
Certain delimiters are used internally as placeholders in text variables during processing. Their use relies implicitly on certain character sequences not occurring as textual content in the processed document. This should be replaced by more robust mechanisms.
One of the challenges of developing a generic stylesheet in XSLT 1.0 is to maintain a one-pass approach. Solving the above-mentioned weaknesses would without doubt make this challenge even bigger. Switching to a two-pass approach may thus be an attractive avenue for future developments.
Adopting a two-pass approach can be done in essentially two ways: an external pipelining mechanism (such as XProc <http://www.w3.org/TR/xproc/>) can be used with XSLT 1.0, or multi-passes can be handled internally in XSLT 2.0, through the `node-set` function, which allows some pipeline-like processing. In both cases,

browser integration could be non-trivial. One interesting way to exploit an external pipelining mechanism would be to have a generic stylesheet generate a model-specific stylesheet from the IS specification, then apply the generated stylesheet to the document instance to generate its IS.
Another functionality that could benefit from the enhanced possibilities of multi-pass / XSLT 2.0 processing is the automatic indentation of the output. Right now, the heuristics used is fairly simple, and it can break down even on simple cases. A more sophisticated and robust heuristics should thus be developed, and this would likely be easier with multi-pass / XSLT 2.0 processing.
A question that needs to be investigated through experimentation is that of determining how much of the indentation should be automatic. In [1], indentation was specified explicitly in a conventional manner in the peritexts.
Let us now consider the foreseeable evolutions of the IS framework and how they could impact the IS generation mechanism. Consider the output of Example 1. As textual content, it includes the following passage:

```
There, he met The person named Barbe-Bleue
```

Note that the article `The` has been capitalized. Why? The answer is that it comes from a text-before segment that is sometimes located at the beginning of a sentence, where capitalization is appropriate. But capitalization in the middle of a sentence (as in Example 1) is inappropriate. The source of the problem is that, in the current framework, the same text-before segment must be used consistently, regardless of its position in a sentence.
Remember that in IS, the focus is not on presentation but on meaning. Since the problem at hand only affects presentation and does not hinder comprehension, it must not be considered major. Moreover, it can be alleviated by various devices, such as writing the peritext all in capitals. With Example 1, this gives the following output, which, though still unusual, is not as strange-looking as the original output:



Thus, it is not clear that the framework needs to be modified to accommodate peritexts that vary according to their position in a sentence.

Other possible extensions in the same line would include peritexts that vary with the position of the element relative to its siblings, with the number of children of the element, and with the grammatical gender of a word or expression in the content of some element or attribute. Clearly, adding any such extension to IS would complicate the IS-generation mechanism. For one thing, it might, require the inclusion of additional peritexts in the IS specification of a model. Then, those additional peritexts would have to be appropriately processed during IS-generation.
We believe that, in all cases, experimentation should be used to determine whether an extension is truly necessary or if some workaround without extension is possible. We think extreme parsimony is of utmost importance for the evolution of IS, because the inclusion of too powerful mechanisms could severely impair the explanatory power of the approach.

6 Conclusion

In this article, we presented a complete implementation, in XSLT 1.0, of the intertextual semantics (IS) generation mechanism for XML documents. The implementation is *model-independent*, in that a generic XSLT stylesheet reads the peritexts from an *IS specification file*, an XML document giving the IS specification (ISS) applicable to the document being processed. The implementation handles attributes as described in [2], hyperlinks (in peritexts or as attribute or element content) as described in [1], and local element definitions (in the sense of W3C schemas), also as described in [1]. In addition, it performs indentation of the IS produced (in the same line as [3], but more elaborate), for increased readability, and handles *exceptions*, elements for which no peritexts are given in the IS specification or attributes unexpected by the peritexts.
After describing the format adopted for ISS files, we gave examples illustrating the functionalities of the implementation, then outlined the structure of the generic stylesheet. Finally, we discussed various aspects of the implementation, possible improvements, and the impact that foreseeable generalizations of the IS framework might have on the IS generation mechanism.
The current version of the stylesheet is available through <http://grds.ebsi.umontreal.ca/>. It is published under the Creative Commons “Attribution-Noncommercial-Share Alike 2.5 Canada” license <http://creativecommons .org/licenses/by-nc-sa/2.5/ca/>. We warmly encourage readers to experiment with it, look at the examples, write IS specifications for their models, either extant or under development, and send comments and suggestions.

7 References

[1] Marcoux, Yves. “A natural-language approach to modeling: Why is some XML so difficult to write?” *Proceedings of Extreme Markup Languages 2006*.

[2] Marcoux, Yves; Rizkallah, Élias. “Exploring intertextual semantics: a reflection on attributes and optionality.” *Proceedings of Extreme Markup Languages 2007*.

[3] Marcoux, Yves; Rizkallah, Élias. “Experience with the use of peritexts to support modeler-author communication in a structured-document system.” *Proceedings of SIGDOC 2007*.

[4] Marcoux, Yves; Rizkallah, Élias. “Intertextual semantics: a semantics for information design.” *Journal of the American Society for Information Science & Technology*, Perspectives issue on design. September 2009, *in Press*.

[5] Smedslund, J. *Dialogues about a new psychology*. Chagrin Falls, Ohio: Taos Institute. 2004.

[6] Sperberg-McQueen, C. M., Huitfeldt, C., & Renear, A. “Meaning and interpretation of markup.” Markup Languages: Theory and Practice 2, 3 (2000), 215–234.

[7] Sperberg-McQueen, C. M., Dubin, D., Huitfeldt, C., & Renear, A. “Drawing inferences on the basis of markup.” In Proceedings of Extreme Markup Languages 2002 (Montreal, Canada, August 2002), B. T. Usdin and S. R. Newcomb, Eds.

[8] Sperberg-McQueen, C. M. & Miller, E. “On mapping from colloquial XML to RDF using XSLT.” Proceedings of Extreme Markup Languages 2004.

[9] Wierzbicka, A. *Semantics, culture, and cognition : universal human concepts in culture-specific configurations*. Oxford University Press. 1992.

[10] Wittgenstein, L. *Philosophical investigations*. Oxford: Blackwell. 1953.