7-2005

On Embedding Machine-Processable Semantics into Documents

Krishnaprasad Thirunarayan
Wright State University - Main Campus, t.k.prasad@wright.edu

On Embedding Machine-Processable Semantics into Documents

Krishnaprasad Thirunarayan

Department of Computer Science and Engineering
Wright State University, Dayton, OH 45435, USA.
tkprasad@cs.wright.edu
http://www.cs.wright.edu/~tkprasad

Abstract. Most Web and legacy paper-based documents are available in human comprehensible text form, not readily accessible to or understood by computer programs. Here we investigate an approach to amalgamate XML technology with programming languages for representational purposes. Specifically, we propose a modular technique to embed machine-processable semantics into a text document with tabular data via annotations, and evaluate it vis a vis document querying, manipulation, and integration. The ultimate aim is to be able to author and extract, human-readable and machine-comprehensible parts of a document "hand in hand", and keep them "side by side".

1 Introduction

The World Wide Web currently contains about 16 million web sites hosting more than 3 billion pages, which are accessed by over 600 million users internationally. Most of the information available on the web, including that obtained from legacy paper-based documents, is in human comprehensible text form, not readily accessible to or understood by computer programs. (Quoting from SHOE FAQ, "Web is not only written in a human-readable language (usually English) but in a human-vision-oriented layout (HTML with tables, frames, etc.) and with human-only-readable graphics". [8]) The enormity and the machine incomprehensibility of the available information has made it very difficult to accurately search, present, summarize, and maintain it for a variety of users [1]. Semantic Web initiative attempts to enrich the available information with machine-processable semantics, enabling both computers and humans to complement each other cooperatively [5, 9]. Automated (web) services enabled by the semantic web technology promises to improve assimilation of web content, providing accurate filtering, classification, location, manipulation and summarization.

Every programming language provides syntax to embed documentation in the code. Typically, a comment appears as a clearly delimited text. In contrast, in Orwell, the documentation text is interspersed with cleanly delimited code that yields executable instructions [10]. Donald E. Knuth popularized the approach of combining a programming language with a documentation language under

the "Literate Programming" banner [3]. (In fact, Knuth used the word WEB for this purpose long before CERN grabbed it!)

XML (eXtensible Markup Language) is a meta-language for designing customized markup languages for different document types. Conceptually, an XML document consists of tagged text (that is, markup is inter-twined with text) in which the tag makes explicit the category and the properties of the enclosed text using attribute-value pairs. For example, the properties can be syntactic/display oriented (such as COLOR, FONT, TYPE-FACE, etc) or semantic (such as MEANING, TYPE, NORMAL-FORM, etc). In general, existing XML technology can be applied to formalize, transform, and query text documents [11].

However, the XML technology developed so far cannot be readily used to formalize/render heterogeneous documents (e.g., MS Word document containing text, images, and complex data structures (such as numeric data in tabular form)) in a form that is suitable for semantic web applications. Specifically, the current approaches to document representation and authoring do not directly address the issue of preserving or abstracting the superficial structure of data (e.g., rectangular grid presentation format for tables) that is suitable for human consumption and traceability, while simultaneously making explicit semantics of data for machine processing. Furthermore, in order to tap into existing legacy documents, it is necessary to develop "modular" and "linear" techniques to augment documents with machine processable semantics. For example, tools such as IBM's Majix convert RTF documents into XML [2], but the resulting documents still need semantic augmentation.

In this paper, XML-based programming languages are evaluated to determine how well they can serve as a substrate for embedding machine-processable semantics into text documents containing complex data. In Section 2, we motivate an XML-based programming and representation langauge. In Section 3, we consider an approach to formalizing tabular data embedded in text document, *in-place*, via a concrete example. In Section 4, we evaluate the proposed approach by presenting the pros and the cons, critiquing it in the context of real-world documents such as Materials and Process Specifications. In Section 5, we conclude with suggestions for long-term research.

2 XML-based Programming and Representation Language

XML (eXtensible Markup Language) is a meta-language for developing customized markup languages for different document types. An XML-Schema can be used to specify a standard syntax for information exchange. Even though XML has been criticized as "syntactic arsenic" by Phil Windley and as verbose variant of LISP by Phil Wadler, the power of XML comes from the fact that XML parsers are interpreted. Overlaying domain-specific XML tags on a text document enables *abstraction*, *formalization*, and *in-place embedding* of machine-processable semantics. However, this approach still yields *static declar-*

ative data, not conveniently handled by systems programming languages (such as C++, Java, etc) or scripting languages (such as PERL, Python, etc). The "impedence mismatch" has been dealt with by providing APIs to mediate conversions to and from XML and native data structures. Furthermore, in a number of distributed applications, an XML document can encode behaviors, whose dynamic semantics can only be uncovered by an embedded interpreter in the host language. To remedy the need for multiple languages and to assist in building web services, XML-based programming languages such as Water [7, 12], XPL [13], etc have been developed.

In order to embed machine-processable semantics into a document, we consider XML-based language for knowledge representation and for encoding behavior consisting of:

- data in the form of ordinary text with clearly marked semantic annotations, and
- behavior/program in the form of definitions for functions, classes, etc.

Ordinary text corresponds to human sensible part, while annotation (with their definitions) corresponds to machine processable part. Definitions encapsulate behavior and can be used to provide and carry out machine processable semantics of data. For instance, the text data "`Delhi is the capital of India.`" can be formalized to different levels of detail in terms of domain-specific vocabulary as follows:

```
<city name="New Delhi"> Delhi </> is the capital of
<country name="India"> India </>.
```

or

```
<capital_Of India "New Delhi">
       <city name="New Delhi"> Delhi </> is the capital of
       <country name=India> India </>.
</>
```

Formalization requires recognizing and delimiting text that corresponds to a relevant concept, and then mapping the delimited text into a standard form that captures its intent. Each resulting annotation consists of an associated XML-element that reflects the semantic category to which the corresponding text fragment belongs, and the XML-attributes are bound to a standard representation of the relevant semantic values. To cite an analogy, in Compiler parlance, the delimited text is called a *lexeme*, the semantic category is called a *token type*, and the semantic value is called an *attribute*. Furthermore, *the annotated data can be* interpreted *by viewing it as a function/procedure call, and defining the XML-element as a function/procedure*. The correspondence between formal parameters and actual arguments can be positional or name-based. Additionally, the definition can make explicit the type of each formal argument, or provide additional integrity constraints to be satisfied by the actual arguments, or in general, map the semantic values. For instance, the requirement that `age` must

be a number (static type), or should be in the range from 0 to 125 years (dynamic constraint) can be made explicit in a modular fashion by "defining" `age`. The same annotated data can be interpreted differently by programming-in different behaviors for the XML-element. For instance, one can recover just the text sans the annotations, verify integrity constraints, or even facilitate data querying by mapping it into Prolog-like syntax.

To summarize, the idea of semantic markup of text is analogous to overlaying the abstract syntax (with attributes) on the free-form text such that the resulting XML document can be flexibly processed by associating different collections of behaviors with XML-elements additively.

3 Formalizing Tabular Data In-place

In relational databases, tables contain schema information as row/column headings and data as rows/columns of entries. In the realm of heterogeneous documents, for example, a table (built out of table primitives or just hand formatted) may be present within an MS Word document, that needs to be isolated, abstracted, and saved as plain text and formalized, before any semantic processing can begin. To motivate and illustrate an XML-based approach to providing semantics to complex data found in text, consider representation of such tables. Assume that the table contains both the headings and the data entries. The precise relationships among the various values in a row/column are tacit in the headings, and are normally obvious to the domain expert (human reader). However, this semantics needs to be made explicit to do any machine processing, including querying, integration, and formal reasoning. If, on the other hand, only semantics rich translation is stored, it may not always be conducive to human comprehension. Thus, it is useful to have a representation language that can serve both the goals. That is, the representation language should have the provision to more or less preserve the grid layout of a table so that changes to the original table can be easily incorporated in text, but, at the same time, describe the semantics of each row/column in a way that is flexible, and applicable to all rows/columns for further machine manipulation.

An XML-based programming language seems to provide a balanced way to achieve and integrate "best" for both the worlds:

- to encode data and to make explicit the semantics in a modular fashion, and
- to effectively use this information for formal manipulation.

For example, the following common table form (found in materials and process specifications) can be saved as text

Thickness (mm)	Tensile Strength (ksi)	Yield Strength @0.2% offset (ksi)
0.5 and under	165	155
0.50 - 1.00	160	150
1.00 - 1.50	155	145
...	...	...

and subsequently annotated as shown below:

```
<table type=Strength>
<parameter  name="Yield Offset" value="0.2%"/>
<rowHeadings "Thickness" "Tensile Strength" "Yield Strength"/>
  <rowData    0   0.50         165                155    />
  <rowData   0.50 1.00         160                150    />
  <rowData   1.00 1.50         155                145    />
...
</table>
```

This, when augmented with `Strength` table definition, should yield a `Strength` table appropriately initialized and exhibiting a prescribed semantics. For instance, to give semantics to the second row, the dependency information among the various columns can be made explicit resulting in the following interpretation:

```
IF ( "Thickness" > 0.50 ) AND ( "Thickness" <= 1.00 )
                  THEN   ("Tensile Strength" = 160)           AND
                         ("Yield Strength" = 150) WHEN ("Yield Offset" = 0.2%)
```

This information is typically needed to determine the expected values for the various strengths, or to check for conformance given the test results on a sample. In general, one may also wish to combine multiple documents.

In the absence of an implemented XML-based programming and representation language at this time, for concreteness, we will use Water-like syntax to formalize the semantics of a smaller table below. (Note that the entire example cannot be coded as desired in Water because Water syntax does not permit embedding code in free-form text and there are no global variables. So, to test the following code in Water, the original document text must be deleted or skipped. See also [7] and Chapter 13 of [5].)

```
Thickness (mm)      Tensile Strength (ksi)       Yield Strength (ksi)
table.<setHeading thickness  strength.tensile strength.yield/>
0.50 and under                  165                155
table.<addRow  0   0.50         165                155    />
0.50   -   1.00                 160                150
table.<addRow  0.50 1.00        160                150    />
1.00   -   1.50                 155                145
table.<addRow  1.00 1.50        155                145    />  ...
```

Each row is annotated with a tag which becomes a method invocation on a table object. If the rows require dissimilar treatment, different tags can be used.

```
<defclass thickness value=required=number units="mm"/>
<defclass thicknessRange max=required=number min=optional=0 units="mm"/>
<defclass strength value=required=number units="ksi">
        <defclass tensile/>
        <defclass yield offset="0.2%"/>
</defclass>
```

```
<defclass table rows=required=vector heading=optional=vector>
   <defmethod setHeading t=required ts=required ys=required>
     <set heading=<vector t ts ys/>/>
   </>
   <defmethod addRow smin smax ts ys>
     <set rows=table.rows.<insert <vector smin smax ts ys/>/>/>
   </>
   <defmethod computeYieldStrength>
     <set temp=fluid.Thickness/>
     <if>  <and temp.<less table.rows.0.0/>  temp.<more_or_equal table.rows.0.1/> />
            table.rows.0.3
          <and temp.<less table.rows.1.0/>  temp.<more_or_equal table.rows.1.1/> />
            table.rows.1.3
      else
            table.rows.2.3
     </if>
   </>
  <defmethod computeTensileStrength>
    <set temp=fluid.Thickness/>
    <set i=0/>
    <do>
        <until  <and temp.<less table.rows.<get i/>.1/>
                     temp.<more_or_equal table.rows.<get i/>.0/> /> >
            table.rows.<get i/>.2
        </until>
        <set i=i.<plus 1/>/>
    </do>
  </>
</>
fluid.<set Thickness=0.60>
<try
<set TensileStrength=table.<computeTensileStrength/>/>
 YieldStrength
>
  "TABLE: out of range error occurred"
</try>
</set>         ...
```

The annotated data can processed using constructs that exploit uniformity. For instance, a looping construct can abbreviate dealing with rows, or a primitive function can be used to split a single (suitably delimited) string into component values, *preserving linear relationship between annotated data and the original text*. In the code for `YieldStrength` only conditional construct is illustrated, while in the definition and call of `TensileStrength` more appropriate looping and exception constructs are employed. Observe also that, ideally, tabular data in each document is annotated, while factoring out annotation definitions separately as "background knowledge".

In summary, meta-programming techniques can be explored to directly represent the data that preserves original presentation, and that simultaneously enables making explicit the semantics in a modular fashion. This can be programmatically applied to integrate different related documents or to check for conformance.

4 Evaluation

Microsoft's Smart Tags technology enables recognition of strings from a controlled vocabulary and associate every occurrence of such strings in a document created using MS Office 2003 with a list of predefined actions [6]. In comparison, the technique discussed here advocates tagging an existing text document and programmatically describing actions associated with tags using XML-syntax. This approach can formalize relationships described in text, and enable ultimately to author and extract, human-readable and machine-comprehensible parts of a document "hand in hand", and keep them "side by side". As to the generalization for tabular data is concerned, it is viable if there are fewer distinct table forms (semantics) compared to the number of concrete data tables. Given that the semantics of a rectangular grid of numbers is implicit in the text, this approach provides a means to make the semantics explicit. However, it uses a functional style requiring detailed description of answer extraction process. Thus, it will be useful to consider querying of declarative table data in logic programming style, along the lines of SHOE [8]. That is, as a first cut, allow expression of declarative knowledge through manual embedding of Prolog code, such as the following encoding of the aforementioned example, into annotation that can be culled out and used with a Prolog inference engine.

```
strengthTableRow( 0,   0.50,  165, 155).
strengthTableRow(0.50, 1.00,  160, 150).
strengthTableRow(1.00, 1.50,  155, 145).
...
strengthTable(Thickness, TensileStrength, YieldStrength) :-
                strengthTableRow(L, U, TensileStrength, YieldStrength),
                L =< Thickness, U > Thickness.
thicknessToTensileStrength(Thickness, TensileStrength) :-
                strengthTable(Thickness, TensileStrength, _).
thicknessToYieldStrength(Thickness, YieldStrength) :-
                strengthTable(Thickness, _, YieldStrength).
?- thicknessToYieldStrength(0.6,YS).
```

Overall, the proposed technique is still not convenient for document integration, which requires normalization. The "regular" tables considered so far exemplify relatively tractable scenerio for complex data. In practice, these regular tables may themselves serve as target encodings of more complex data found in legacy documents.

In the long run, a generalization of the approach outlined above has potential to unify data models and behaviors embodied in object-oriented languages with Web standards, facilitate meta-programming and traceability, embed machine processable semantics into text documents, generalize dynamic typing to

constraint checking for reliability and for authoring executable specifications, provide convenient access to Web Services infrastructure, and ultimately yield an expressive representation language for realizing the Semantic Web.

5 Conclusions

This paper made a case for developing an XML-based Programming and Representation language for authoring and extracting extant Web and legacy documents. In order to appreciate the hurdles to be overcome, we considered, as an illustrative example, the formalization of text document containing realistic tabular numeric data. The approach discussed is modular in that the annotation of the original document and the codification of the semantics of annotations can be separated. The approach enables adequate formalization of document, that can support traceability and querying. However, it is still not convenient for document integration that may necessitate defining a suitable normal form.

For the long-term success of the Semantic Web initiative, it is important to understand principles and develop techniques/tools for authoring and formalizing documents in a form that is both human comprehensible and machine processable. This is also essential for extracting, querying and integrating documents. For tractability and concreteness, it may be beneficial to scope the domain of discourse.

References

1. Davies J., Fensel D., and van Harmelen F. (Eds.): *Towards the Semantic Web: Ontology-Driven Knowledge Management*, John Wiley and Sons, Inc., 2003.
2. `http://www-136.ibm.com/developerworks/xml/`
3. Knuth D.: *Literate Programming*, CSLI Lecture Notes Number 27. Center for the Study of Language and Information, Stanford University, CA. 1992.
4. Fensel D.: Semantic Web enabled Web Services, Invited Talk at: In *7th International Workshop on Applications of Natural Language to Information Systems*, June 2002, Stockholm, Sweden.
5. Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W. (Eds.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, The MIT Press, 2003.
6. Kunicki Chris: What's New with Smart Tags in Office 2003, MSDN Library Article, Microsoft Corporation, January 2003.
7. Plusch M.: Water : Simplified Web Services and XML Programming, Wiley Publishing, 2003.
8. `http://www.cs.umd.edu/projects/plus/SHOE/index.html`
9. `http://www.semanticweb.org/`
10. Wadler P.: *An Introduction to Orwell*. Programming Research Group, Oxford University, April 1985.
11. Wadler, P.: XML: Some hyperlinks minus the hype. `http://www.research.avayalabs.com/user/wadler/xml/`.
12. `http://www.waterlanguage.org/`
13. `http://www.vbxml.com/xpl/`