

## **Position Paper / Outline CAS Project**

### **Tentative Title: DSS: Literate Programming for XML Schema**

Kevin Reiss (kevin.reiss [at] gmail.com)

#### ***Introduction***

XML Schema Languages, including DTDs, Relax NG, and XML Schema, provide a way to formally specify, in machine-readable form, the elements of a given markup language, the patterns that those elements can appear in, and the attributes that those elements can have. XML document instances that are marked up using the elements of a particular schema can be checked for conformance to that schema. This process is known as validation. XML schemas provide a mechanism to constrain the contents of a document. There has been a proliferation of XML applications defined using XML schema tools since the 1.0 version of the standard was introduced in 1998.

However, XML schema languages still do not satisfy many of the needs of both XML application developers, web programmers utilizing distributed XML data, and encoders. This is because current XML schema languages lack a formal (machine-readable) mechanism for defining the semantic relationships between the elements and attributes in an XML markup language beyond the generic parent-child and sibling relationships that are implicit in the tree data structure that is produced when an XML document is parsed. The natural language documentation that accompanies is often sparse, inconsistent, and presented in a haphazard manner, when compared to the types of structured and standardized documentation available for object oriented programming

languages such as Java. There are some XML applications, such as the Text Encoding Initiative, with well thought out and thorough documentation, but this is a rarity. For XML to fully support the goals of interoperability and the creation of high-quality structured documents a principled, consistent, system for markup language designers to use needs to be created. It is still up to the designer to populate that system with quality prose and content model declarations, but such a system needs to be provided.

### **What is needed**

There is a convincing case that there is a strong need for a consistent, concise, extensible, modifiable system for the generation of documentation for XML schema. There a number of reasons that support this (1) the large number of different XML schema languages. Document Type Definitions, the initial, SGML-derived, schema language defined for use with XML did not prove adequate for the wide variety of XML users, so a number of different XML schema language languages have been developed. These include rule-based schema languages such as schematron, object oriented efforts like the W3C XML Schema Standard, or Relax NG which is focused on the definition of schema language that uses the generic tree structure, the labeled tree abstraction (Clark, 2003 – Relax NG Book), as the basis for it's model. The creators and maintainers of XML applications may need to generate some or all of these schema languages for users of the application. With the large number of XML applications available it is essential that application developers, and markup encoders have adequate documentation to facilitate the integration of different XML vocabularies into applications. It is important individuals have clear, understandable, and consistent natural language documentation in order to support these types of integrated applications.

The system I propose will address these concerns by working out a methodology to provide a technology that can serve as a general framework for authoring XML schema languages. This framework, called the Documentation, Syntax and Semantics (DSS), will seek to serve the markup language designer who wishes to provide concise, consistent, and efficient documentation for a XML application. The three components that will be represented are:

1. Documentation
2. Syntax
3. Semantics

Consider this example from a Metadata, Encoding, and Transmission Standard (METS) document taken from the example documents available on the Library of Congress website. – Identify appropriate example doc

Consider the accompanying documentation for this element and its available attributes taken from the documentation posted on the LOC site [<http://www.loc.gov/standards/mets/docs/mets.v1-5.html>] – Identify appropriate element

## ***The Problem: Limitations of XML Schema Languages***

### General Problems

1. Schema Proliferation -
2. Web Services – RSS world, OAI-PMH, SRU
3. Schema Combination – Incorporating
4. Mapping from one schema to another – Reducing MODS to Dublin Core

### **General documentation problems**

The current state of XML schema documentation has not progressed much beyond the recommendations for Maler and El Andaloussi's text *Developing SGML DTD's* from

1996. The volume recommends developing a structured dictionary of the schema under development and provided documentation suited to both developers and document encoders. This is about all that most XML applications provide. In the case of many web service applications the documentation may be buried in the WSDL file if it is provided. This is somewhat disheartening because since we lack a formal machine readable mechanism to define the semantics of an XML application the natural language documentation that is available is all the developer has, short of piecing together the schema designers intentions through the study of sample document instances or the content models of the XML application as they presented through the chosen schema language used for the application.

1. Inconsistency in style and availability – [Lubell, et. al Extreme 2006]
2. Too much room for user error in interpretation – [various Bechamel Articles]
3. Not much support at author-time – intertextual semantics paper [Marcoux Extreme 2006]

#### XML Schema Syntactic Limitations

1. Only Content Models – [James Clark – Relax NG]
2. Not really an object-orientated methodology – [Norman Walsh 2004]

#### Limitations of XML Schema Languages

XML schema offers a mechanism to supply the syntax of the element and attribute names, and the order in which they can appear. However constraints do not equal semantics (Daum + Merten, System Architecture with XML 2003). XML has been utilized with most of the major formal modeling methodologies, the entity relational model, object orientation – the W3C XML Schema language is explicitly designed to use the object oriented model exclusively, tree structures, and knowledge representation systems such as RDF, OWL, and Topic Maps. While these methods provide a way to

structure a specific XML application, there is no general purpose way to define the basic relations between the elements and attributes of an XML application.

Renear, Sperberg-McQueen, Dubin, and Huitfeldt (include cite) have detailed the semantic failings of XML. The major problems with the lack of semantics for XML are lack or unclear definition of the following characteristics:

1. Class Relationships
2. Propagation, inheritance, combination
3. Arity and deixis
4. Ontological Ambiguity

The ability to represent this would greatly increase the ability of developers to efficiently develop robust tools to process XML document instances marked up using any type of XML application.

## ***Literate Programming***

### **Knuth**

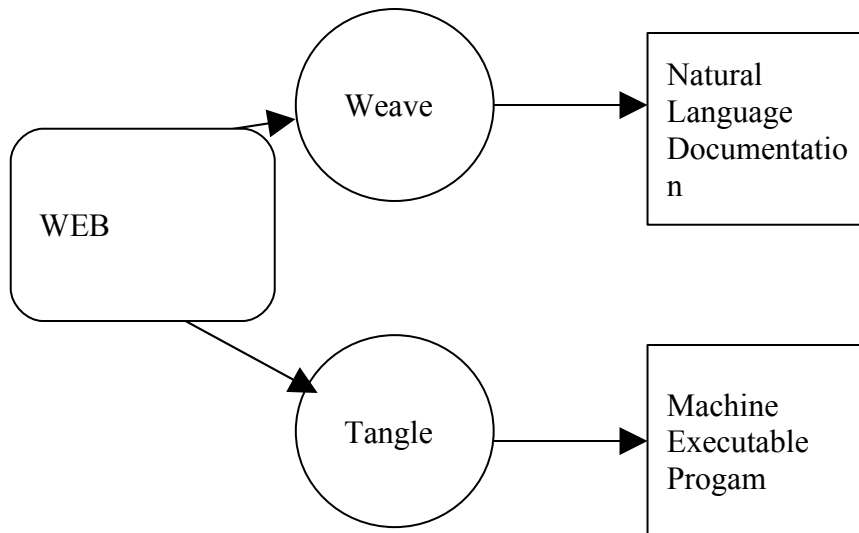
Literate Programming, the methodology first developed by Donald Knuth, can offer an instructive model of the type of system that could be presented to XML developers to facilitate “sustainable” schema development. Such a system would be able to adequately documenting all three important characteristics of an XML schema.

1. Natural Language Documentation
2. Syntax
3. Semantic Information about the model

Literate programming, as defined by Knuth, is an approach to programming that posits that the most important aspect of is the readability of the program. He emphasized

that programmers should think of their work as essayists do, and be concerned with coherent exposition of a program and with consistent style for variable and function names (Knuth, 1984 article). Knuth developed a system entitled “WEB” that supports this methodology. The natural language documentation for a computer and the structured programming code are written in the same source file. This file, which in Knuth’s system used the TeX typesetting language to markup natural language documentation, and PASCAL as the structured programming, was then passed to the web system, where two operations would take place. The single web file is used to generate a typeset, readable, version of the program documentation, that includes a textual representation of the program’s source, and a machine-executable file that can be used to execute the functionality provided by the program. The process of generating the human-readable formatted documentation is known as the “weave”. The process of generating the executable binary code is known as the “tangle.” All literate programming methodologies have subscribed to the basic model.

Knuth’s Literate Programming Paradigm (Adapted from 1984 Article chart)



### **Literate Programming XML Style**

Knuth's system was developed for the structured programming methodology. A markup language such as XML has somewhat different requirements than Knuth's methodology; however the WEB system has proved an important model for a number of XML schema construction methodologies that have been developed.

There have been a number of XML attempts to provide literate programming environments to support XML schema creation. The most important of these is the Text Encoding Initiative's One Document Does it All (ODD) system. The system dates to the early 90s and was originally developed when the TEI was an SGML application. (Cite Sperberg-McQueen article SWEB 1993).

### **The TEI ODD**

The ODD system is an impressive achievement. The TEI itself is undergoing a major revision, the P5, where the entire system has moved to use of Relax NG for the

syntactic declarations of the TEI's content models and the use of Extensible Stylesheet Language Transformation (XSLT) stylesheets to transform the source model to transform the content in The system (Burnard and Rahtz, 2004) is not tied to any single one of the XML, because the TEI maintainers desire to ensure that the model for a TEI document remains independent of any single schema language XML or otherwise. This approach has been successful since the TEI has weathered the transition from SGML DTD, to XML DTD, to Relax NG XML Schema with relatively few problems. (Burnard and Rahtz, 2004, Rahtz, 2005, Baumen and Flanders 2005 – ODD customizations paper). The ODD system currently produces the entire print and online version of the TEI guidelines.

The ODD system is especially impressive in its flexibility and modularity. It is able to support the creation of the highly modular, extensible TEI P5 Relax NG schema. The system also is able to represent the complicated class hierarchy, including both elements and attributes, which are present within the TEI model of a document.

Show example of the ODD source code

### **Lessor Efforts**

Other authors have proposed literate programming systems for XML. Norman Walsh, the primary maintainer of the docbook XML application that supports the encoding of technical documentation, proposed a litprog system in 2002 [Walsh, online article 2002], and worked on some demonstration stylesheets, but seems to have abandoned the idea because no follow-up work on his system has been reported. Coates [Extreme 2002] also proposed a literate programming system at Extreme 2002 (verify?)

but this hasn't been followed up either, the last date on the sourceforge repository for this project is August 6, 2002. There are some other XML lit systems floating around

#### Built-in Schema Documentation Tools

- Relax NG – Discuss Relax's annotation features, incorporation of documentation elements from other namespaces into Relax XML syntax schemas
- W3C XML Schema – Discuss annotation
- Old Fashioned Comments

### ***XML Semantic Efforts***

1. Bechamel (Various Papers)
2. NDR Paper at Extreme 2006 (Lubell, et. al)
3. Intextual Semantics Paper at Extreme 2006 (Marcoux)
4. MDL Paper at Extreme 2006 (Quin)

As Renear, et. al (Doc Engineering 2002) and Marcoux (Extreme 2006) have discussed shown, the lack of a mechanism to formally define the semantics of a markup language is a limitation of current XML authoring environments. This is challenge that must be dealt with by the XML community. However at this time the author will focus upon the creation of a robust, general purpose documentation system that provides a mechanism for specifying via ritualized prose a semantic checklist that can begin to document semantics. This differs from the Renear, Dubin approach of analyzing the document model using a logic environment post-authorship (?). The discussions by Lubell, et. al (2006) regarding the establishment of formal Naming and Design Rules (NDRs) or Marcoux's idea of Intextual semantics are more in line with this author's goal.

These two interesting approaches, presented at Extreme markup 2006, seek to advance the quality of XML documentation systems.

## ***The DSS System***

### **Initial Project Scope**

The DSS system will support the creation of effective, concise, well-documented XML Applications that can be validated with a XML Schema language, rather than the attempt to set up a system to infer the semantics content of already existing schemas or the incorporation of knowledge representation technologies directly into XML authoring tools (Dubin, et. all Extreme 2006). An ideal system for this task would provide a mechanism to formally express, in machine-readable fashion, the semantic relationships and assertions of the content models contained in a XML schema. For the first iteration of the DSS I will not seek to incorporate a knowledge representation technology that can accomplish this, but rather will seek to provide mechanisms to begin addressing the question of the semantic definitions using carefully structured prose documentation that accurately reflects the intentions of the language designer.

An ideal literate programming system to support XML Schema authorship would also allow the schema designer to generate any type of XML Schema language for the XML application from the WEB source document. In the interest of completing the demonstration application I will focus on an environment that generates Relax NG XML Schemas for the defined application. Relax NG is the most flexible XML Schema language available since it places no built-in constraints, other than the generic labeled-tree model inherent in XML, from the modeling methodology used to design the schema language itself (Clark, Relax NG forward 2003).

## **Application Sketch**

Use the TEI ODD system as my basic tool for schema documentation. Extend the system to incorporate improvements to the ability to construct structured and consistent natural language documentation for any XML application. I will develop a general purpose literate programming system for XML using the TEI ODD as the base. I will extend the markup provided for the ODD format, Chapter 27 of the TEI P5 guidelines, to serve as a general purpose tool with enhanced power to produce documentation that is consistent, concise and highly readable. The resulting system will be:

1. Specified in one document
2. Modifiability
3. Customizable
4. Minimization of effort on the part of the author
5. Management of Classes
6. Modularity and Extensibility
7. It will produce documentation that uses natural language to produce a sort of semantic checklist to the reader that adequately and consistently describes the elements and attributes of the application.

## **Why the ODD is an excellent system to base my project on?**

1. It is well-tested, straightforward and already has an excellent set of documentation (TEI Guidelines)
2. It has strong support for class management.
3. It is highly modular.
4. It is extensible.
5. It can incorporate multiple namespaces into a document that can be validated.

6. Stylesheets exist for the TEI that can produce well formatted print and online documentation.
7. Most importantly it already has a well-thought literate programming system in place.

## ***Key Deliverables***

### **A Semantic Checklist**

A key advance of this project will be the exploration of the feasibility of constructing a semantic checklist for XML application documentation. This checklist will create an explicit format and create a stylized routine for constructing comments for an element or an attribute as one is added to an XML application. This checklist will support consistency of language, and seek make clear how the item in question integrates with content models defined within the schema. The questions such a checklist will seek to answer are:

1. Does the declaration apply to the content of the element?
2. Is the declaration of the element about the text contained within the element or attribute? Semantic vs. Structural markup distinction.
3. What does the element describe?
4. Is the element a member of a class?
5. Is the element inherited from another?
6. Does the element serve as a superclass?
7. Parent-child relationship connotations
8. Sibling relationships?

## **Example Documents**

I believe the following XML applications can provide some valuable that will provide valuable illustrations as to why a structured and principled general purpose literate programming system can benefit XML users. I will seek to provide an analysis of particular occurrences in document instances, schemas, and schema documentation to support my points. I will then try and how these could be better represented in my proposed DSS system.

1. METS - Consider the fileGRP element within the METS document format as it is documented within the current METS guidelines. This is an important element that documents the digital objects various manifestations (verify this). The documentation given for the element within the METS guidelines is quite spartan and cryptic.
2. Atom
3. OAI-PMH
4. TEI Document or XHTML to provide a document-centric example

## **Complete Schema**

Ideally I would like to create one complete example of an existing XML application authored under the example DSS framework. I would produce all 3 items for the applications (1) The DSS source code, (2) A Relax NG Schema, (3) and the natural language documentation including the semantic checklist. I will pick the example document that seems to offer the most promise of instructive examples of how DSS can improve on current documentation efforts.