Documenting and Implementing Guidelines with Schematron

Joshua Lubell <lubell@nist.gov>

Abstract

Data exchange specifications must be broad and general in order to achieve acceptance, yet they must also be customizable in a controlled and interoperable manner in order to be useful. This paper describes an approach employing the Schematron language and literate programming ideas for developing the guidelines needed to effectively use data exchange specifications.

Table of Contents

Why Guidelines?

Data exchange specifications for electronic business (e-business) applications describe in a computer-interpretable fashion the information to be transferred between systems. Thanks to the ubiquity and success of the Extensible Markup Language [XML] as an implementation technology, these specifications are often encoded using XML schemas [XSchema]. XML's approval as a World Wide Web Consortium (W3C) Recommendation in 1998 precipitated a deluge of proposed XML schemas for data exchange [Cover], many of them rigidly designed for a specific vertical area of application. Most of these schemas were subsequently abandoned as developers learned that having too many non-interoperable languages hinders rather than enables systems integration.

Recognition of the need for interoperability led to a drive to standardize XML schemas meeting two requirements:

- *Customizability* – to enable interoperable extensions to be added,
- *Inclusiveness* – to facilitate use in multiple contexts.

These customizable and inclusive schemas are more horizontal in nature and fewer in number than the first generation of proposed schemas. Examples from the e-business and e-government domains include the Universal Business Language [UBL], the Open Applications Group Integration Specification [OAGIS], and the National Information Exchange Model [NIEM] standards.

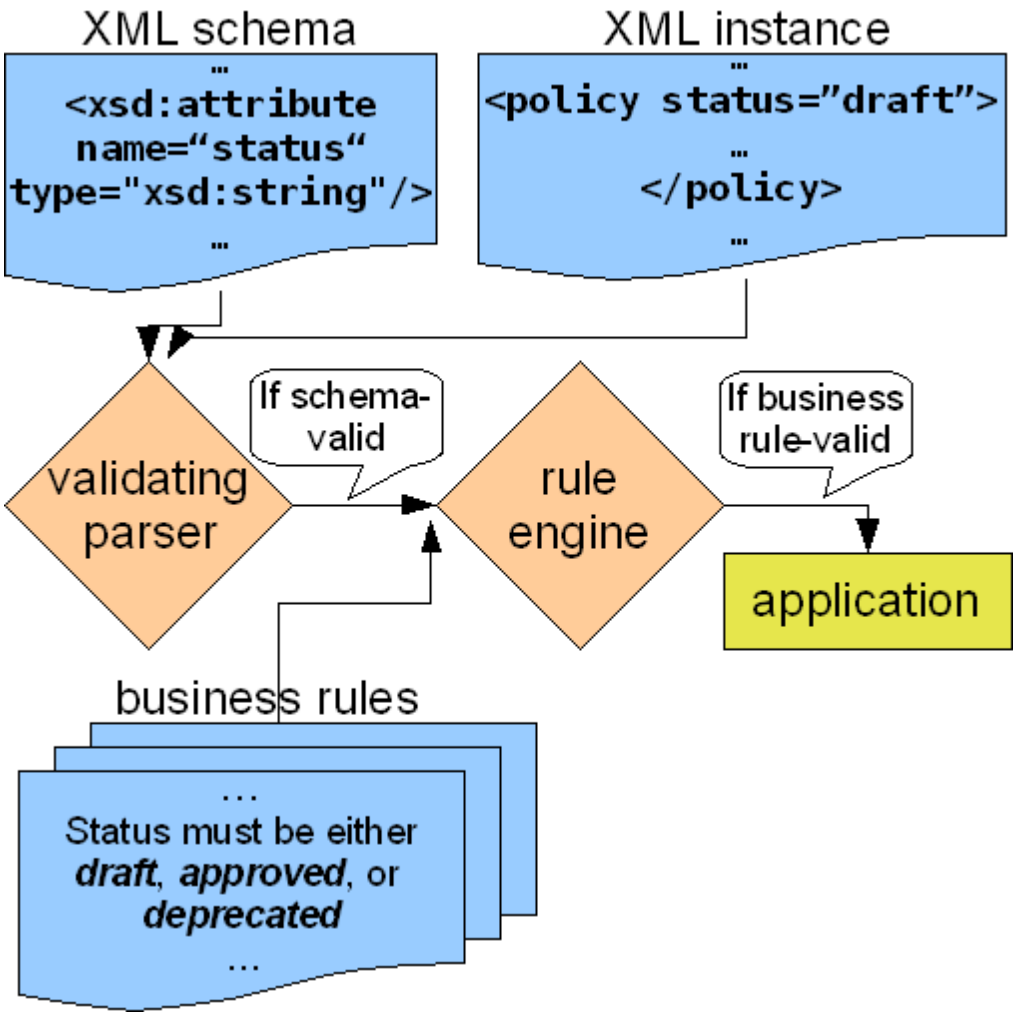
Unlike e-business and e-government, where the use of markup language for data exchange is a relatively recent phenomenon, customizable and inclusive standards for textual documentation have been around for decades and predate the advent of XML. Examples of widely used horizontal and extensible XML schema standards for text documentation include DocBook [DocBook], Darwin Information Typing Architecture

[DITA], the Text Encoding Initiative [TEI], and S1000D [S1000D]. These documentation standards (with the exception of DITA) have been in existence longer than UBL, OAGIS, and NIEM.

Whether they apply to e-business or text processing applications, interoperable data exchange specifications must be both inclusive enough to appeal to a broad community of developers and customizable enough to allow developers to add new elements when needed. This is where *guidelines* — or rules — come into play. A schema that is inclusive is typically loosely specified in order to make it acceptable to the community at large. This means heavy use of optional elements, optional attributes, unrestricted data types and, possibly, unconstrained content models. The result is a schema that often needs to be more tightly constrained for specific use cases, by mutual agreement of the actors involved. Such guidelines are commonly called *business rules*.

As a simple example of a business rule adding an additional constraint to a schema, consider a data exchange schema with an element `policy`, where `policy` has an attribute `status` whose type is a string. Now suppose that a group implementing the schema adds a business rule to further constrain `status` to be one of an enumerated set of values. Validating an XML instance is now a two-step process. First, the instance is validated against the original schema. If the instance is schema-valid, it is then validated against the business rule. Figure 1 shows the validation steps.

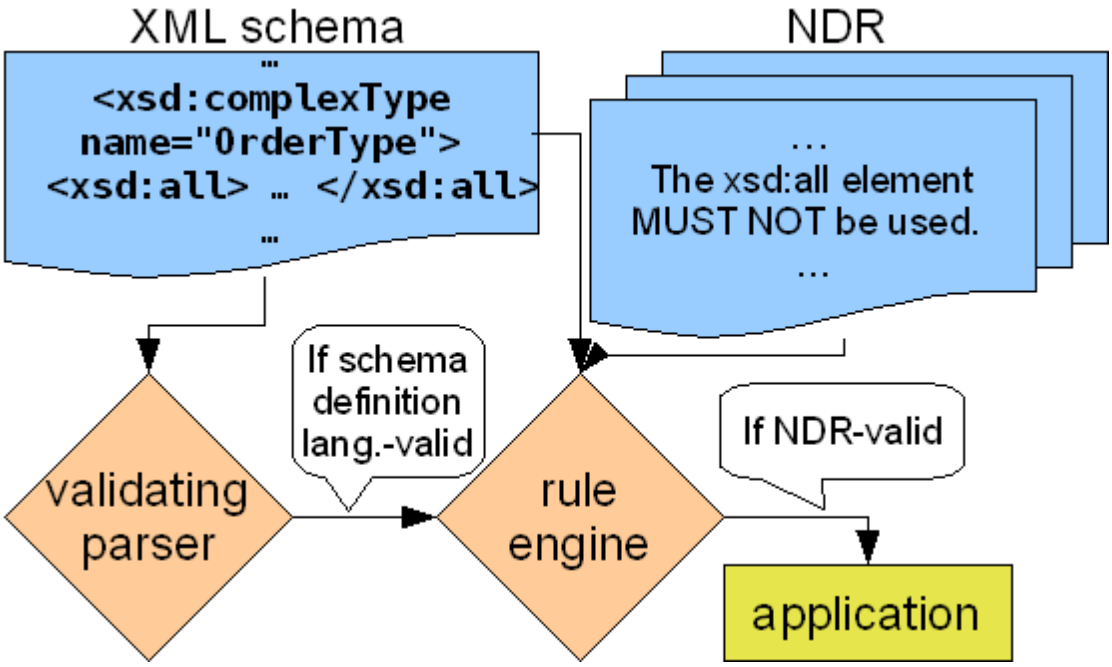
Figure 1.



Validation of an instance against a schema with an added business rule.

XML schema Naming and Design Rules [NDR] are a noteworthy special case of business rules. An NDR is a set of guidelines for developing a new XML schema or extending an existing schema. NDRs are useful both for promoting good design practice and for ensuring that schema developers follow consistent naming conventions. NDR validation is similar to the general case shown in Figure 1, except that the XML schema is a schema defining the syntax of the XML schema definition language, the XML instance is a schema specified in that definition language, and the business rules are an NDR. Figure 2 shows an example where an XML schema containing a complex type definition containing an `all` group is validated against an NDR with a rule forbidding the use of `all`.

Figure 2.



Validation of an XML schema against an NDR.

The NDR approach is applicable to any schema language with an XML syntax and a “schema for schemas” defining the grammar of the schema language. However, NDRs are most often used to promote quality and interoperability of schemas written in the W3C XML schema definition language. This is because many large-scale data exchange specifications are defined using W3C XML schemas, and standards developers have found that the W3C XML schema definition language's many features must be used judiciously to ensure that extensions are compatible with one another and with the base schema.

Implementing Guidelines with Schematron

To summarize the previous section, XML data exchange schemas need to be both inclusive and customizable, guidelines known as *business rules* are used to make an inclusive schema more restrictive to a subset of the exchange schema's users, and guidelines known as *NDRs* are used to control the use of the exchange schema's definition language for the purpose of ensuring that schema extensions are interoperable. Many data exchange standards allow both for the addition of constraints via business rules and for the addition of extensions as permitted by an NDR.

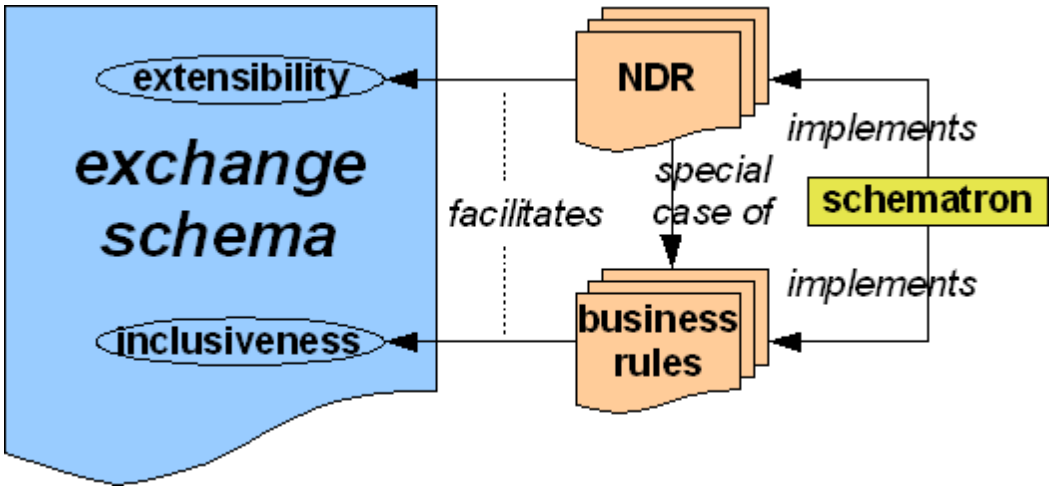
But guidelines in and of themselves are of only limited value unless they are implemented in software. Without automated enforcement, guidelines may be subject to multiple human interpretations. Also, a set of guidelines may be inconsistent, making it impossible for an XML document to satisfy one

guideline without violating another. While implementing guidelines does not prevent these problems, the implementation process often makes ambiguities and inconsistencies more obvious and thus more likely to be discovered and remedied. Most important, automating guideline enforcement makes it more likely that the guidelines will actually be followed as opposed to relying on human diligence alone.

Because NDRs are a special case of business rules, any implementation method for business rules is also applicable for NDRs. Although just about any computer-interpretable language is a potential guideline implementation method, a popular choice is Schematron [Schematron], a schema language for XML. Schematron differs from other schema languages in that it is rule-based and uses XPath [XPath] expressions instead of grammars. Instead of imposing a grammar on an XML document, a Schematron processor applies assertions to specific context paths within the document. If the XML document fails to meet an assertion, a diagnostic message supplied by the author of the Schematron schema is displayed. Because Schematron supports assertions about arbitrary patterns in XML documents, Schematron can enforce constraints that would be hard to enforce using grammar-based schema languages. Also, Schematron's modular, rule-based syntax and author-supplied diagnostic messages make it an attractive method for implementing guidelines[1].

Figure 3 shows how NDRs, business rules, and Schematron fit together in the context of data exchange schema interoperability. The NDR controls how the schema is extended should new definitions need to be added. The business rules facilitate inclusiveness by allowing a broad-based consensus standard to coexist alongside additional constraints required by exchange partners. Both the business rules and the NDR (which is effectively a set of business rules restricting the language — typically the W3C XML schema definition language — used to specify the exchange schema) may be implemented in Schematron. This implementation scenario sets the stage for the next section's discussion of implementing guidelines using a literate programming approach.

Figure 3.



Using NDR, business rules and Schematron to ensure exchange schema interoperability.

A Literate Programming Approach

Literate programming [Knuth, LitProg], a software development method invented by Donald Knuth, seeks to free implementers from the confines of computer language syntax — allowing them to code in a

[1]Version 1.1 of the W3C XML Schema standard [XSchema1.1], a Candidate Recommendation at the time of this writing, adds Schematron-like capabilities to the XML Schema Definition Language. Once implementations become available, XML Schema 1.1 might be worth considering as an alternative to Schematron.

manner more consistent with human thought processes than traditional programming paradigms. Literate programming elevates human narrative to the same status as computer-interpretable code by allowing them to be co-mingled together in a “web.” A literate programming system includes tools for “weaving” formatted human-readable documentation from the source and “tangling” (i.e., rearranging) the source to produce compilable code.

In traditional approaches, documentation for human consumption such as source code comments, algorithm descriptions, requirements, and design rationale has second class status relative to computer-interpretable code. Even when source code in traditionally-developed software is well-documented with comments, the comments serve as an aid to understanding the code rather than something from which the code evolves. Because XML is well suited both for data exchange between software applications and for producing human-readable documentation in numerous formats, XML and literate programming are a good fit. Therefore, it is no surprise that a number of literate programming-inspired XML applications have been developed [Reiss]. One of the more well-known of these is “One Document Does it all” (ODD) [Burnard], a literate programming XML vocabulary from which TEI schemas and documentation can be generated [Roma].

Schematron is a particularly useful literate programming tool for documenting and implementing guidelines. Schematron's syntax mimics an itemized list of guidelines. Schematron gives authors full flexibility in specifying diagnostic messages triggered upon an assertion passing or failing. And Schematron allows for “always-true” assertions, enabling guidelines that are untestable or unimplemented to nevertheless be included in the Schematron schema. Consider the following rule, adapted from [Jelliffe]:

```
<sch:rule context="/">
   <sch:assert test="true()"  role="Untestable" >The document shall
be maintainable</sch:assert>
   <sch:assert test="true()"  role="UnImplemented" >All terms must be
standard English words.</sch:assert>
</sch:rule>
```

A Schematron processor will not check for these two constraints. However, the Schematron language not only allows for expressing text describing the constraints, but also provides constructs such as the `role` attribute for metadata. Additionally, Schematron processors ignore foreign markup and child elements inside foreign markup, making it easy to intersperse the Schematron syntax with content marked up in a text documentation language such as DocBook or the Extensible HyperText Markup Language [XHTML].

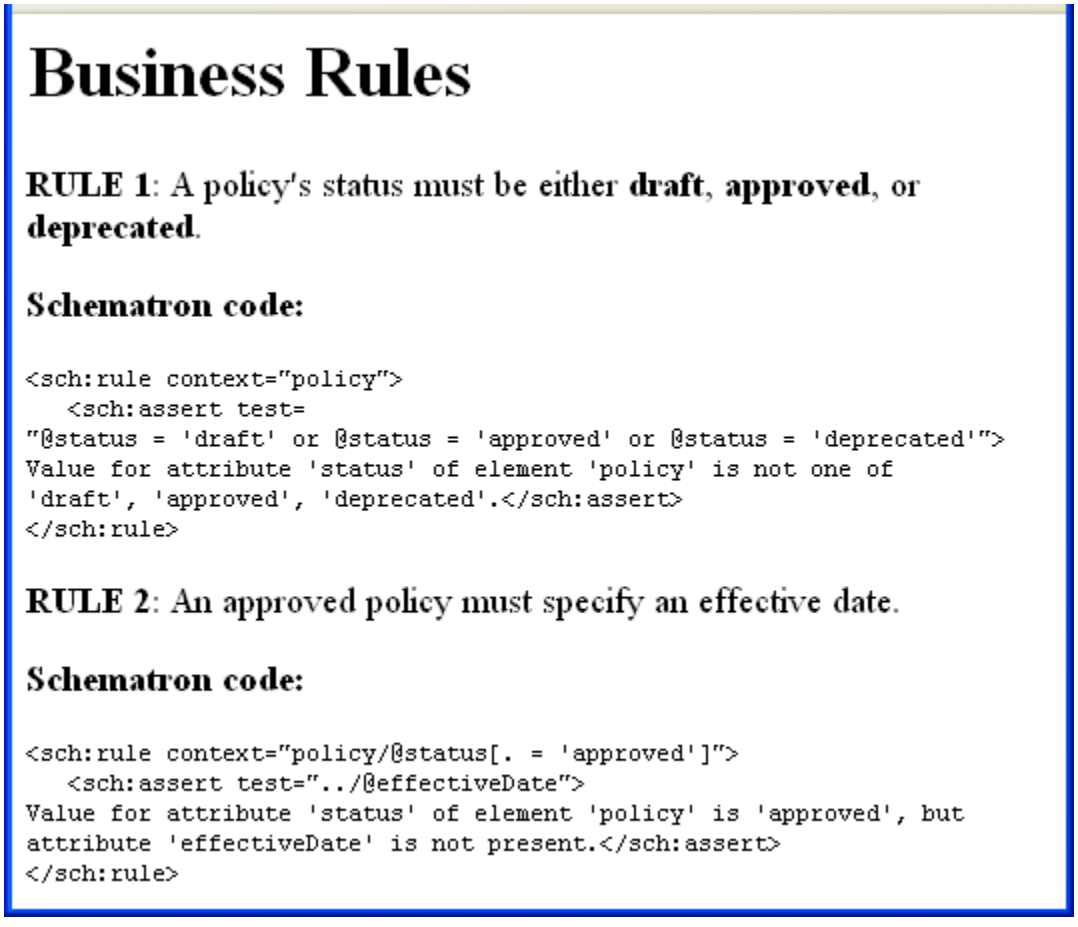
Consider the following Schematron schema containing two business rules. The first rule, from Figure 1, restricts a policy's status to an enumerated set of values. The second rule, a co-occurrence constraint, requires an “approved” policy to have an effective date.

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
     xmlns="http://www.w3.org/1999/xhtml">
  <sch:title>Business Rules</sch:title>
  <sch:pattern>
    <sch:rule context="policy">
      <h3>Rule 1</h3>
      <p>A policy's status must be either <strong>draft</strong>,
<strong>approved</strong>, or <strong>deprecated</strong>.</p>
      <sch:assert test=
"@status = 'draft' or @status = 'approved' or @status = 'deprecated'">
Value for attribute 'status' of element 'policy' is not one of
'draft', 'approved', 'deprecated'.</sch:assert>
    </sch:rule>
```

```
    <sch:rule context="policy/@status[. = 'approved']">
      <h3>Rule 2</h3>
      <p>An approved policy must specify an effective date.</p>
      <sch:assert test="../@effectiveDate">
Value for attribute 'status' of element 'policy' is 'approved', but
attribute 'effectiveDate' is not present.</sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

This is a perfectly valid Schematron schema, even though it contains XHTML markup. The XHTML-enhanced Schematron exemplifies the literate programming ideal of a single source co-mingling processable code with documentation, where the code and documentation have equal status. The Schematron-valid source can be used as-is by a Schematron processor to validate an XML instance against the business rules — no literate programming "tangling" operation is needed. The formatted web page shown in Figure 4 is generated using a very simple transformation accomplishing the literate programming "weaving" operation.

Figure 4.



Business Rules

RULE 1: A policy's status must be either **draft**, **approved**, or **deprecated**.

Schematron code:

```
<sch:rule context="policy">
   <sch:assert test=
"@status = 'draft' or @status = 'approved' or @status = 'deprecated'">
Value for attribute 'status' of element 'policy' is not one of
'draft', 'approved', 'deprecated'.</sch:assert>
</sch:rule>
```

RULE 2: An approved policy must specify an effective date.

Schematron code:

```
<sch:rule context="policy/@status[. = 'approved']">
   <sch:assert test="../@effectiveDate">
Value for attribute 'status' of element 'policy' is 'approved', but
attribute 'effectiveDate' is not present.</sch:assert>
</sch:rule>
```

Documentation generated using a simple transformation of the Schematron schema.

Although the example uses XHTML rather than DocBook for enhancing the Schematron, DocBook may actually be more suitable since the DocBook data model is designed specifically for representing technical documentation. I choose XHTML over DocBook for the examples in this paper purely as a convenience

to myself and to readers. Generating formatted output from XHTML is easy. And anyone familiar with XML technologies has seen XHTML syntax.

The NDRProfile Experience

The previous section presented the case for implementing guidelines using a Schematron-based literate programming approach. This section discusses a recent effort employing a non-literate programming approach to guidelines (specifically NDR) development.

The National Institute of Standards and Technology (NIST) Quality of Design (QOD) application [Morris] provides a mechanism for checking XML schema design quality against NDRs in a collaborative environment. Recognizing the need for an XML vocabulary to represent NDRs, NIST created NDRProfile [Harvey] — an XML schema providing a common format in which NDRs can be exchanged, managed and reused. Although the NDRProfile schema's development was inspired by QOD, the schema can be used independently of QOD, either as a means of authoring an NDR document or as a vehicle for sharing rules between NDRProfile schema-compliant software applications.

Unlike the previous section's example co-mingling Schematron and XHTML, NDRProfile represents a set of guidelines using its own specialized vocabulary. Implementations of each guideline are specified using a `script` element whose content is computer-interpretable code, most commonly Schematron code. The following is an example of an NDRProfile instance containing a single NDR rule for UBL prohibiting the use of the XML Schema `all` element.

```
<NDRProfile NDRProfileID="prd-UBL-NDR-2.0">
  <NDRTitle>Universal Business Language Naming and Design Rules</NDRTitle>
  <OrganizationInformation>
    <OrganizationName>OASIS</OrganizationName>
    <ContactURI>http://www.oasis-open.org</ContactURI>
  </OrganizationInformation>
  <Guidance guidanceID="UBL-2.0-GXS-8">
    <Classification>General</Classification>
    <EnforcementLevel>Mandatory</EnforcementLevel>
    <Status>Draft</Status>
    <Testability>Fully-Testable</Testability>
    <GuidanceText>xsd:all MUST NOT be used.</GuidanceText>
    <TestCases>
      <TestCase testCaseID="UBL-2.0-GXS-8-1" ruleType="ISO-Schematron">
 <TestName>GXS8</TestName>
 <Script><![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <ns uri="http://www.w3.org/3301/XMLSchema" prefix="xsd"/>
  <pattern id="UBL-2.0-GSX-8">
    <rule context="xsd:all">
      <report test="true()">Error: The xsd:all element MUST
NOT be used.</report></rule></pattern></schema>
]]></Script></TestCase></TestCases></Guidance></NDRProfile>
```

This example illustrates a number of disadvantages of the NDRProfile schema design relative to the approach discussed in the previous section. Because NDRProfile does not permit foreign markup, the Schematron code in the `script` element must be either enclosed in a CDATA section, or the left angle brackets must be escaped. This not only makes the instance uglier, but also requires extraction of the

Schematron code before it can be executed by a Schematron processor. In the previous section's example, no extraction is necessary because the document is already a valid Schematron schema.

Another disadvantage is that NDRProfile lacks the elegance of the literate programming approach. The code is a second class citizen relative to the documentation, so the documentation and code do not share equal status. In fact, an NDRProfile developer is likely to write the Schematron code as a separate endeavor from authoring the NDRProfile instance and paste it into the instance after the fact. This is contrary to literate programming, where the code and documentation are created in concert with one another.

Finally, because the NDRProfile schema defines its own vocabulary rather than maximizing reuse of existing XML languages, custom tools must be developed for authoring, validating, and processing it. Thus NDRProfile shares the barriers to success of other newly-invented languages. And it is usually a bad idea to invent a new XML language [Bray].

So why was NDRProfile designed the way it was? The main reason was that the normative form of most NDRs is an unstructured document in a proprietary word processor format. This caused us to think of an NDRProfile as a text document foremost, rather than as an application of literate programming. A second reason was that we wanted to allow for other implementation languages besides Schematron. However, by using “always-true” assertions, `role` metadata, and foreign markup, one can use a Schematron-based vocabulary while at the same time allowing for other implementation methods.

The following example shows how an NDRProfile can be represented using a variant of the previous section's literate programming approach, with XHTML `class` attributes added to provide NDR-specific annotations.

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
     xmlns="http://www.w3.org/1999/xhtml" id="prd-UBL-NDR-2.0">
  <sch:title>Universal Business Language Naming and Design Rules</sch:title>
  <sch:ns uri="http://www.w3.org/2001/XMLSchema" prefix="xsd"/>
  <sch:p class="schemaType">NDRProfile</sch:p>
  <p class="OrganizationInformation">
    <span class="OrganizationName">OASIS</span>
    <a class="ContactURI" href="http://www.oasis-open.org">
http://www.oasis-open.org</a></p>
  <sch:pattern id="UBL-2.0-GXS-8">
    <sch:rule context="xsd:all" id="UBL-2.0-GXS-8-1">
      <p class="Classification">General</p>
      <p class="EnforcementLevel">Mandatory</p>
      <p class="Status">Draft</p>
      <p class="Testability">Fully-Testable</p>
      <p class="GuidanceText"><strong>GXS8</strong>: xsd:all MUST NOT
be used.</p>
      <sch:report test="true()">Error: The xsd:all element MUST
NOT be used.</sch:report></sch:rule></sch:pattern></sch:schema>
```

This approach has neither of the NDRProfile schema's first two disadvantages. The third disadvantage is minimized by reusing existing, well-established XML vocabularies as much as possible.

Discussion

In this paper, I first introduced two kinds of guidelines needed for effective use of data exchange schemas: business rules for further constraining a schema, and NDR for controlling the design of schema extensions.

Next I illustrated how Schematron can be used in conjunction with literate programming principles to simultaneously document and implement guidelines. I then discussed past experience with NDRProfile, an approach to encoding guidelines that at the time seemed intuitive and sensible, but in retrospect is more convoluted and less efficient than the literate programming approach.

My approach emphasizes Schematron because Schematron is an appropriate language for implementing NDRs and business rules as well as (with annotations added) a suitable guideline documentation language. This is not to imply that all XML-based literate programming systems should use Schematron. For applications such as ODD, where the underlying TEI data model is more complex, hierarchical, and harder to represent as a set of independent rules, Schematron's benefits as a literate programming tool may not be as great.

A key enabler of the Schematron-based literate programming approach advocated in this paper is the ability of a single XML document to incorporate multiple vocabularies. This is distinct from the more conventional notion of a single-vocabulary XML document that can be processed multiple ways. An XML document having multiple vocabularies is not a new idea. Back in the mid-1990s, ISO standardized the Architectural Form Definition Requirements [AFDR, Lubell]. Unlike a Document Type Definition (DTD), an “architecture” as defined by the AFDR need not specify a complete document type. Instead, an architecture defines rules known as “architectural forms” that developers can apply in defining their vocabularies. A document can use multiple architectures, and architectures themselves can inherit grammar rules from other architectures.

Although the AFDR never gained widespread traction among XML developers, the AFDR authors were incredibly forward-thinking in foreseeing the need to decouple vocabularies from document types. Software applications today are moving off of desktops and are increasingly being provided as services over the Internet. XML developers are coming to realize that XML documents and vocabularies should mimic the characteristics of the World Wide Web. Since the Web is a decentralized network of chunks of interconnected information, XML document designs should maximize the use of small, independently-developed vocabularies [Costello].

The newly-standardized Namespace-based Validation Dispatching Language [NVDL] provides a method for validating XML documents containing multiple vocabularies defined using different schema languages. NVDL supports a variety of schema languages including the XML Schema Definition Language, DTDs, Schematron and RELAX NG [RelaxNG]. NVDL thus enables the validation of a document such as the previous section's example of a literate programming approach to NDRProfile. Consider the following NVDL document describing the validation of a Schematron schema with embedded XHTML paragraph elements:

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
       xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
       startMode="sch">
  <mode name="sch"><namespace ns="http://purl.oclc.org/dsdl/schematron">
      <validate schema="iso-schematron.rng" useMode="xhtmlpara"/>
    </namespace></mode>
  <mode name="xhtmlpara"><namespace ns="http://www.w3.org/1999/xhtml">
      <validate schema="p.rng"/></namespace></mode></rules>
```

This NVDL document specifies that the Schematron schema with embedded XHTML is validated by initially checking all Schematron elements against the normative RELAX NG schema (assumed to be in the file `iso-schematron.rng`) for the Schematron language. Whenever an XHTML element is encountered, the XHTML is validated against `p.rng`, assumed to be a customization of the normative RELAX NG schema for XHTML allowing `p` as a starting element. The NVDL document does not validate syntactic correctness of NDRProfile annotations. NDRProfile annotation checking could be done

by further customization of `iso-schematron.rng` and `p.rng`, or alternatively by writing a separate Schematron schema specifically for NDRProfile annotation checking and adding an “NDRProfile” mode to the NVDL document.

References

[AFDR] ISO/IEC 10744:1997. Information processing — Time-based Structuring Language (HyTime)- 2d edition. Annex A.3 Architectural Form Definition Requirements (AFDR). [cited 08 Jun 2009]. http://www1.y12.doe.gov/capabilities/sgml/wg8/document/n1920

[Bray] Tim Bray. *Don't Invent XML Languages*. 2006-01-09. [cited 24 Apr 2009]. http://www.tbray.org/ongoing/When/200x/2006/01/08/No-New-XML-Languages

[Costello] Roger Costello. *XML Designers: Take Cue from the Web*. xFront. [citied 09 Jun 2009]. http://xfront.com/xml-designers-take-cue-from-the-web/index.html

[Cover] Cover Pages. *XML Applications and Initiatives*. [cited 21 May 2009]. http://xml.coverpages.org/xmlApplications.html

[DITA] OASIS. *DITA Version 1.1 Specification Overview*. 1 August 2007. [cited 08 May 2009]. http://docs.oasis-open.org/dita/v1.1/overview/overview.html

[DocBook] Norman Walsh. *DocBook 5.0: The Definitive Guide*. ISBN: 156592-580-7. O'Reilly & Associates, Inc. Version 0.0.25. [cited 08 May 2009]. http://docbook.org

[Harvey] Betty Harvey, Joshua Lubell, Puja Goyal, KC Morris. *NDRProfile Schema Version 1.0 User Guide*. National Institute of Standards and Technology. NISTIR 7547. December 2008. [cited 24 Apr 2009]. http://qod.sourceforge.net/ndrprofile/

[Jelliffe] Rick Jelliffe. *Expressing untested and untestable constraints in Schematron*. [cited 23 Apr 2009]. http://www.oreillynet.com/xml/blog/2007/03/expressing_untested_and_untest.html

[Knuth] D.E. Knuth. *Literate Programming*. The Computer Journal 1984 27(2):97-111; doi:10.1093/comjnl/27.2.97. British Computer Society. [cited 11 May 2009]. http://www.literateprogramming.com/knuthweb.pdf

[LitProg] Wikipedia. *Literate programming*. [cited 23 Apr 2009]. http://en.wikipedia.org/wiki/Literate_programming

[Lubell] Joshua Lubell. *Architectures in an XML World*. *Markup Languages: Theory and Practice*. Vol. 3. No. 4. Fall 2001. doi:10.1162/109966202760152167. [cited 08 Jun 2009]. http://www.mel.nist.gov/div826/staff/lubell/xsltoolbox/apex/

[Morris] K.C. Morris et al. *User's Guide for the Quality of Design Testing Tool and the Content Checker*. National Institute of Standards and Technology. NISTIR 7538. October 2008. [cited 11 May 2009]. http://www.mel.nist.gov/publications/view_pub.cgi?pub_id=824715

[NIEM] National Information Exchange Model. [cited 08 May 2009]. http://www.niem.gov

[NDR] Cover Pages. *Naming and Design Rules*. [cited 24 Apr 2009]. http://xml.coverpages.org/ndr.html

[NVDL] ISO/IEC 19757-4. *Information technology — Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language (NVDL)*. First edition 2006-06-01. [cited 04 Jun 2009]. http://www.iso.org/PubliclyAvailableStandards

[RelaxNG] ISO/IEC 19757-2. *Information technology — Document Schema Definition Languages (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*. First edition 2003-12-01. [cited 09 Jun 2009]. http://www.iso.org/PubliclyAvailableStandards

[OAGIS] Open Applications Group Integration Specification (OAGIS) Release 9.3. [cited 07 May 2009]. http://www.oagi.org

[Burnard] Lou Burnard and Sebastian Rahtz. *RelaxNG with Son of ODD*. Proceedings of *Extreme Markup Languages 2004* conference (Montreal, Quebec). August 2004. [cited 05 Jun 2009]. http://conferences.idealliance.org/extreme

[Reiss] Kevin Reiss. *Literate Documentation for XML Schema*. Digital Humanities 2007 poster materials. [cited 24 Apr 2009]. http://kreisscas.blogspot.com/2007/06/digital-humanities-materials.html

[Roma] Roma: generating validators for the TEI. [cited 05 Jun 2009]. http://www.tei-c.org/Roma

[S1000D] AeroSpace and Defence Industries Association of Europe. *S1000D International specification for technical publications*. Issue 4.0. 2008-08-01. [cited 08 May 2009]. http://www.s1000d.org

[Schematron] ISO/IEC 19757-3. *Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron*. First edition 2006-06-01. [cited 23 Apr 2009]. http://www.iso.org/PubliclyAvailableStandards

[TEI] Text Encoding Initiative Consortium. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. edited by Lou Burnard and Syd Bauman. Version 1.3.0. Last updated on February 1, 2009. http://www.tei-c.org

[UBL] OASIS. *Universal Business Language 1.0*. 15 September 2004. [cited 07 May 2009]. http://docs.oasis-open.org/ubl/cd-UBL-1.0/

[XHTML] World Wide Web Consortium. *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation 26 January 2000, revised 1 August 2002. [cited 21 May 2009]. http://www.w3.org/TR/xhtml1/

[XML] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation 26 November 2008. [cited 15 Apr 2009]. http://www.w3.org/TR/xml/

[XPath] World Wide Web Consortium. *XML Path Language (XPath) Version 1.0*. W3C Recommendation 16 November 1999. [cited 23 Apr 2009]. http://www.w3.org/TR/xpath

[XSchema] World Wide Web Consortium. *XML Schema Part 0: Primer Second Edition*. W3C Recommendation 28 October 2004. [cited 15 Apr 2009]. http://www.w3.org/TR/xmlschema-0/

[XSchema1.1] World Wide Web Consortium. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C Candidate Recommendation 30 April 2009. [cited 11 Jun 2009]. http://www.w3.org/TR/xmlschema11-1/