Extreme Markup Languages 2006®

Montréal, Québec
August 7-11, 2006

Implementing XML Schema Naming and Design Rules[1]

Perils and Pitfalls

Joshua Lubell
National Institute of Standards and Technology

Boonserm (Serm) Kulvatunyou
National Institute of Standards and Technology

KC Morris
National Institute of Standards and Technology

Betty Harvey
Electronic Commerce Connection, Inc.

Abstract

We are building a methodology and tool kit for encoding XML schema NDRs [Naming and Design Rules] in a computer-interpretable fashion, enabling automated rule enforcement and improving schema quality. Through our experience implementing rules from various NDR specifications, we discuss some issues and offer practical guidance to organizations grappling with NDR development.

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

Implementing XML Schema Naming and Design Rules[1]

Perils and Pitfalls

Table of Contents

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

Implementing XML Schema Naming and Design Rules[1]

Perils and Pitfalls

Joshua Lubell, Boonserm (Serm) Kulvatunyou, KC Morris, and Betty Harvey

§ Motivation

Organizations developing XML schemas [XSD] often establish NDRs [Naming and Design Rules] in order to maximize interoperability and quality. NDRs are a good way to help enforce best practices, a particular modeling methodology (such as the CCTS [Core Components Technical Specification] [CCTS]), or conformance to standards such as ISO 11179 (Naming and Design Principles for Data Elements) [ISO11179-1] [ISO11179-5]. But no single set of Naming and Design Rules can satisfy everyone's requirements. As a result, NDRs are proliferating. And new groups embarking on XML schema development are asking, “Should we create our own NDR, or should we use a pre-existing one?”

To complicate matters further, most NDRs are standardized as presentation-oriented documents to be read by humans. They are not easily machine-interpretable. Ambiguities and lack of automation make rule enforcement difficult. NDR rules typically have not been created with a mindset to have schemas validated automatically to ensure they have not violated any of the NDR rules.

And that is not the worst of it. NDRs tend to be authored in proprietary binary word processor formats and lack any explicit document structure. As a result, it is hard to compare NDRs with one another or to track changes between different versions of an NDR. Therefore, NDR developers are inclined to “reinvent the wheel” rather than maximize reuse of existing NDRs.

To summarize, the following barriers prevent NDRs from improving schema interoperability and quality.

- NDR proliferation
- Absence of NDR document structure
- Lack of automated rule enforcement
- Limited versioning and traceability

If rules are encoded in a computer-interpretable form, then a single implementation is chosen, ambiguities disappear, and the rules can be applied automatically as part of XML schema validation. Specific benefits derived by having NDRs that can be automated are:

- Enforcement is more consistent and reliable. The current state of affairs, where NDRs are presentation-oriented and intended only for human interpretation, makes enforcement of NDRs problematic at best, nearly impossible at worst.
- The process of encoding forces NDR developers to be more rigorous when specifying rules. Implementation provides a good test of a rule's completeness and clarity. Tested rules result in better NDRs.
- Enforcing the rules results in better XML Schemas. The benefits of the NDR are achieved through automated verification.

The bottom line: cycle time from requirements for a schema to production XML is reduced.

This paper explores methods for removing the barriers to using NDRs effectively. We discuss some of the implementation issues we have discovered by trying to write validation rules for various NDRs. Next, we present a methodology for implementing NDRs and a prototype system using our methodology. We conclude with some observations regarding our experience.

§ Automating a Rule

Consider the following rule from the UBL [Universal Business Language] NDR [UBL][UBLNDR]:

> [ELD1] Each `UBL:DocumentSchema` MUST identify one and only one global element declaration that defines the document `ccts:AggregateBusinessInformationEntity` being conveyed in the Schema expression. That global element MUST include an `xsd:annotation` child element

> which MUST further contain an `xsd:documentation` child element that declares *"This element MUST be conveyed as the root element in any instance document based on this Schema expression."*

This rule says that the schema for a UBL document has to identify a single global element as the root element and that this element's type must conform to the CCTS definition of an ABIE [Aggregate Business Information Entity]. A UBL document is an XML instance valid with respect to one of the set of XML schemas in UBL for common business message structures, such as "Order," "Invoice," etc. An ABIE is defined to be "a collection of related pieces of business information that together convey a distinct business meaning in a specific *business context*." [CCTS]

An example of a schema conforming to this rule is the UBL Order schema. The relevant definitions are as follows:

```
<xsd:element name="Order" type="OrderType">
  <xsd:annotation>
    <xsd:documentation>This element MUST be conveyed as the
root element in any instance document based on this Schema
expression</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:complexType name="OrderType">
  <xsd:annotation>
    <xsd:documentation>
      <ccts:Component>
        <ccts:ComponentType>ABIE</ccts:ComponentType>

          ...

</xsd:complexType>
```

Some observations regarding this rule:

- The rule's label, ELD1 (Element Declaration Rule #1), is a unique identifier, but only with respect to other rules in the UBL NDR. Another standards body or group of schema developers could give the same label to a completely different rule.
- Rule ELD1 cannot be understood in isolation. Its semantics depend on other rules, for example rules saying that the XML Schema namespace has to use "xsd" as its prefix, and Core Components Technical Specification metadata has to use the "ccts" namespace prefix. As a result of this context dependence, enforcing ELD1 requires background knowledge of the UBL NDR as a whole, as well as familiarity with the CCTS.
- Although ELD1 is clearly written, it is written in English and thus requires human interpretation. If the rule were written in a computer-interpretable language, ELD1 enforcement could be automated.
- ELD1 is really two rules in one. The first sub-rule stipulates that there shall be a single global root element. This rule's context is the entire schema. The second sub-rule requires that this root element be an ABIE. This rule's context is limited to the schema's global root element (as determined by the first sub-rule).

This example shows that a seemingly simple rule can become more complex upon closer examination. The complexity becomes even more pronounced when we attempt to implement the rule. As our implementation method, we choose Schematron [ISOSch], a schema language for XML. Schematron can be used to validate any XML document, including an XML schema itself. Schematron differs from most other schema languages in that it is rule-based and uses XPath [XPath] expressions instead of grammars. Instead of enforcing a grammar on an XML document, a Schematron processor applies assertions to specific context paths within the document. If an XML document fails to meet an assertion, a diagnostic message supplied by the author of the Schematron schema can be displayed. Because Schematron supports assertions about arbitrary patterns in XML documents, and because diagnostic messages are author-supplied, Schematron can enforce constraints that would be problematic to enforce using grammar-based schema languages.

The following Schematron schema, which shows Schematron's expressive power, implements rule ELD1. We write this schema for Schematron 1.5 [Jeliffe] because, as of this writing, few Schematron processors support ISO Schematron.

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <ns prefix="xs" uri="http://www.w3.org/2001/XMLSchema"/>
  <ns prefix="ccts" uri=
"urn:oasis:names:specification:ubl:schema:xsd:CoreComponentParameters-1.0"/>
```

```
  <key name="complex-type" match="xs:complexType" path="@name"/>
  <pattern name=
"UBL-100 [Rule ELD1] One root element is defined in a Control schema" id="eld1">
    <rule context="xs:schema">
      <assert test=
"count(xs:element[normalize-space(xs:annotation/xs:documentation) =
'This element MUST be conveyed as the root element in any instance
document based on this Schema expression'])=1">
The document schema MUST identify one and only one global root
element containing the documentation, "This element MUST be
conveyed as the root element in any instance document based on
this Schema expression".
      </assert>
    </rule>
    <rule context=
"xs:schema/xs:element[normalize-space(xs:annotation/xs:documentation) =
'This element MUST be conveyed as the root element in any instance
document based on this Schema expression']">
      <assert diagnostics="root-elt-name" test=
"normalize-space(key('complex-type', @type)//ccts:ComponentType) = 'ABIE'">
The global root element MUST be an Aggregate Business
Information Entity.
      </assert>
    </rule>
  </pattern>
  <diagnostics>
    <diagnostic id="root-elt-name">Global root element
'<value-of select="@name"/>' is not an ABIE.</diagnostic>
  </diagnostics>
</schema>
```

If we remove `<ccts:ComponentType>ABIE</ccts:ComponentType>` from the `OrderType` complex type definition in the UBL Order schema, this Schematron schema produces the following diagnostic message:

```
error: assertion failed:
The global root element MUST be an Aggregate Business
Information Entity.
Global root element 'Order' is not an ABIE.
```

Removing `xsd:element`'s `xsd:annotation` child element results in the following Schematron output:

```
error: assertion failed:
The document schema MUST identify one and only one global
root element containing the documentation, "This element
MUST be conveyed as the root element in any instance document
based on this Schema expression".
```

As one can see, the Schematron implementation of ELD1 is nontrivial. The trickiest part of the implementation is supplying the correct XPath expressions for the assertions and rule contexts. On the other hand, once a rule is implemented, it can be automatically enforced as part of the schema development processes.

As another example of Schematron as an implementation method, consider Documentation rule #2 from the DON [US Department of Navy] NDR [DON]. The rule expresses the requirement that a type derived by restriction from a UDT [Unqualified Data Type] must be documented in a particular way.

> [DOC2] A data type definition MAY contain one or more content component restrictions to provide additional information on the relationship between the data type and its corresponding UDT. If used, the content component restrictions must contain a structured set of annotations in the following patterns:
>
> - RestrictionType (mandatory): Defines the type of format restriction that applies to the content component.
> - RestrictionValue (mandatory): The actual value of the format restriction that applies to the content component.
> - ExpressionType (optional): Defines the type of the regular expression of the restriction value.

The following XPath expression naively defines the context for a Schematron rule implementing DOC2:

```
//xsd:restriction[substring-before(@base,':')='udt']
```

This expression is correct only if the prefix used for the UDT namespace is “udt”. If the DON UDT namespace URI [Uniform Resource Indicator] is bound to a different prefix, the context XPath expression is incorrect. The following context XPath expression does not contain a hard-coded namespace prefix:

```
//xsd:restriction[namespace::*[local-name()=
  substring-before(../@base,':') and
  .='urn:us:gov:dod:don:enterprise:udt:1:0']]
```

This expression is more robust than the previous expression because it relies on the UDT namespace URI, specified in the DON NDR, rather than a namespace prefix. The expression uses XPath's `namespace` axis to access the `xsd:restriction` element's namespace node. Although this expression is more complicated than the previous expression, the complexity is necessary in order to adequately test for text values that are QNames (namespace-qualified) [Skonnard]. Since QName attribute values are common in W3C [World Wide Web Consortium] XML Schemas, the `namespace` axis can be indispensable when implementing an XML Schema NDR in Schematron.

The following Schematron schema partially implements rule DOC2 from the DON NDR. The implementation is only partial because the Schematron code verifies the existence of the required `RestrictionType` and `RestrictionValue` documentation elements, but does not check the element content.

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <ns prefix="xsd" uri="http://www.w3.org/2001/XMLSchema"/>
  <ns prefix="ccts" uri="urn:oasis:names:tc:ubl:CoreComponentParameters:1:0"/>
  <ns prefix="cdp" uri="urn:us:gov:dod:don:enterprise:cdp:1.0"/>
  <pattern id="DOC" name="Documentation Rules">
    <rule id="DOC2"
          context="//xsd:restriction[namespace::*[local-name()=
          substring-before(../@base,':') and
       .  ='urn:us:gov:dod:don:enterprise:udt:1:0']]">
      <assert test=
"../../xsd:annotation/xsd:documentation/cdp:component/cdp:RestrictionType |
../../xsd:annotation/xsd:documentation/cdp:component/ccts:RestrictionType">
[DOC2-1/3]:  RestrictionType (mandatory): Defines the type of format
restriction that applies to the content component.
      </assert>
      <assert test=
"../../xsd:annotation/xsd:documentation/cdp:component/cdp:RestrictionValue |
../../xsd:annotation/xsd:documentation/cdp:component/ccts:RestrictionValue">
[DOC1-2/3]:  RestrictionValue (mandatory): The actual value of the format
restriction that applies to the content component.
      </assert>
    </rule>
  </pattern>
</schema>
```

Although Schematron's rule-based syntax and use of XPath make it particularly handy for NDR implementation, some Naming and Design Rules require additional capabilities. For instance, rules constraining names of elements, attributes, or types might require a dictionary lookup[2]. UBL General Naming Rules #1 and #7 are examples of such rules:

> [GNR1] UBL XML element, attribute and type names MUST be in the English language, using the primary English spellings provided in the Oxford English Dictionary.
>
> [GNR7] UBL XML element, attribute and type names MUST be in singular form unless the concept itself is plural.

Rules GNR1 and GNR7 would be most easily implemented using an interpreted scripting language supporting regular expressions and allowing system calls to external applications (such as a local or online electronic dictionary). Compared with Schematron, interpreted scripting languages have an advantage of greater expressiveness to implement more complex rules. Scripting languages can interact with other applications and systems. They allow complex branching logic and can process regular expressions. Their major disadvantage is that they are not XML native; consequently, implementing rules may require a different thought process that might be unfamiliar to XML developers. Rather than expressing XPaths directly, the scripting language must either use an API [application programmer interface] to access the XML or must require translation of the XML into its own native representation. As a result, scripting language implementations are likely to be more verbose and difficult for XML developers to understand than Schematron implementations.

XPath 2.0 [XPath2], a Candidate W3C [World Wide Web Consortium] Recommendation, has more capabilities than the current XPath W3C Recommendation. As Schematron processors begin to support XPath 2.0, some rules may become more easily implementable.

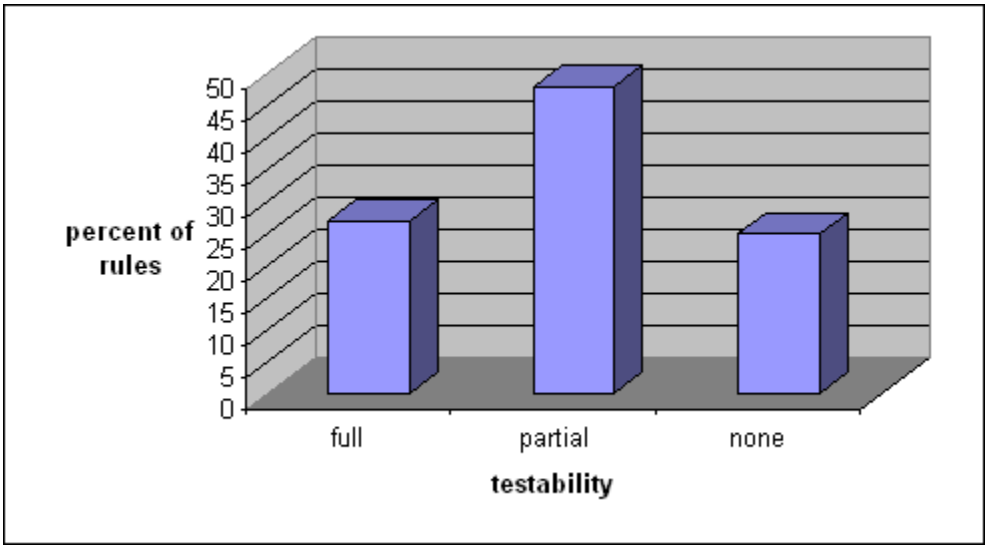
§ Issues Affecting Rule Testability

We recently attempted a Schematron implementation of the DON NDR, a specification containing upwards of one hundred rules. We undertook this project both to help XML developers write better

schemas for Federal government applications and also to gain insight into best practices for NDR development. We chose the DON NDR not to single it out for criticism, but because it is based on the UBL NDR, is publicly available, and is being promoted as a Navy standard.

In our analysis of this NDR (see Figure 1), we determined that only about a quarter of the rules could be fully tested through implementation in Schematron. The proportion of partially testable rules was approximately half. The remaining quarter of the rules was untestable.

Figure 1: DON NDR Testability



As discussed in "Automating a Rule", some rules that cannot be implemented in Schematron can be implemented using a more general-purpose implementation method. However, there are issues that make it difficult or even impossible to implement a rule in *any* programming language. We discuss some of these issues in the following sub-sections.

The Use of MAY

When a rule states that something MAY occur, strictly speaking the rule can always pass. However, the intent of the rule's creator may be otherwise. For example, MAY could mean that, although something is allowed, it should be avoided if possible. Thus it can be hard to tell whether the text is telling the developer that the guidance is recommended or that it should be used only in special circumstances.

Consider the following rule from the DON NDR using MAY:

> [CTD8] Code and ID ccts:BBIE Property complex types MAY use the xsd:choice element to reference global elements defined in standardized ID Scheme or Code List Schema modules.

Testing for `xsd:choice` would result in a positive if it occurs or negative if it does not occur. Both scenarios are allowed.

Although the use of MAY is entirely appropriate for NDRs, NDR developers might consider distinguishing between two levels of MAY, the first level being an implementable rule to be automatically enforced. The second level would serve as guidance. The guidance can be considered to be "best practices" and would not be implemented for automatic enforcement.

An alternative approach is to associate a more specific semantic to MAY in the NDR context to mean "Discourage." Thus a rule implementation can flag the MAY condition as a possible problem and provide a warning rather than an error message during schema validation. For example, an implementation of DON rule CTD8 could generate the following message for a schema legally using `xsd:choice` to reference global elements:

```
warning: referencing global element using xsd:choice
```

Requirement for External Interfaces/Resources

Some rules require the ability to test against resources outside of the NDR environment. For example, consider the following DON rule, a more specialized version of UBL rule GNR1 shown near the end of "Automating a Rule":

> [GNR1] XML element, attribute, and type names MUST be in the English language, using the Oxford English Dictionary for Writers and Editors (Latest Ed.). Where both American and English spellings of the same word are provided, the American spelling MUST be used.

This rule is impossible to automate without access to an electronic OED [Oxford English Dictionary] lookup service. And that assumes the lookup service is up-to-date with respect to the latest OED edition.

Rule Proliferation

DON rule GNR1 also illustrates the problem of proliferation of rules without traceability from one NDR to another. The DON rule is the same as UBL rule GNR1, but with the additional constraints that American spellings be favored and that the latest OED edition be used. However, the DON NDR does not say that this rule is a specialization of a rule from the UBL NDR, nor does it say how the UBL rule is specialized. Adding to the confusion is the fact that the DON NDR categorizes its rules similarly to UBL and uses the same ID (GNR1) for a rule that is similar but not identical to its UBL counterpart.

Suppose the UBL Technical Committee were to decide that Merriam-Webster should be the dictionary lookup source instead of the OED. Should the DON revise rule GNR1 accordingly? Also, can an implementation of DON rule GNR1 reuse existing code implementing UBL rule GNR1? These questions would be easier to answer if NDRs were specified in a manner more amenable to traceability and reuse.

Ambiguous Terminology

A common NDR pitfall is ambiguous or confusing terminology. Many rules are complex in nature, and the verbiage can become very confusing to the reader. It is difficult for the reader to determine exactly what the NDR designer had in mind when creating the rule. Sometimes the NDR reader has to guess the mindset of the author. For an XML developer (especially one with limited experience) there is a high probability of interpreting the rule improperly.

A prime example of ambiguity is the use of the term `xsd:SchemaExpression`. This term is used but not defined in the DON NDR. The UBL NDR unhelpfully defines `xsd:SchemaExpression` to be "a concept." Undefined terminology causes developer confusion and hampers rule implementation.

Mixed Content

An XML element type has mixed content if the element is allowed to contain character data interspersed with child elements. Mixed content is essential for representing semi-structured data, particularly "document-centric" XML. In document-centric XML, block elements such as paragraphs contain character data which may be interspersed with semantically meaningful floating elements such as hyperlinks, bibliographic citations, mathematical notations, or chemical formulas.

Many existing NDRs, including the UBL NDR, were not developed with documents in mind. These NDRs forbid mixed content, which may make sense in an EDI [Electronic Data Interchange] environment where traditionally information is required to have a strict delineated structure. However, an argument can be made that this restriction is overly severe since even in an EDI environment, transformation of the XML can be made to conform to an EDI message and the mixed content can be eliminated during transformation. One can also argue that a lot of important business data is semi-structured rather than tightly structured and that prohibiting mixed content hampers schema developers too much.

The DON NDR allows mixed content, but only if the mixed content is defined by a namespace from a Navy-approved business standard such as XHTML [the Extensible HyperText Markup Language] [XHTML]. Although the DON's policy regarding mixed content is less severe than the UBL policy, the DON policy complicates NDR enforcement. For example, although the DON rule GNR1 requires that elements and types have English language names, many XHTML elements such as `p`, `h1`, and `li` are not English language words. An implementation of GNR1 therefore has to treat XHTML definitions as a special case.

§ An NDR Implementation Tool Kit

So far we have provided examples of rules, discussed implementation pitfalls, and mentioned alternative implementation methods. Although we have touched upon the issues of NDR proliferation and difficulties with reuse of rules, we have not yet presented a solution to these problems. To help address proliferation and reuse, the National Institute of Standards and Technology (NIST) is building a QoD ["Quality of Design"] software tool kit [QoD] [Kul04a] to make it easier for schema developers to choose and apply an appropriate NDR set. A Web-based prototype allows users to upload a schema and select rules from a cross-section of NDRs to check the schema against. The prototype's purpose is to provide a user-friendly environment for checking XML schema design quality in a collaborative environment.

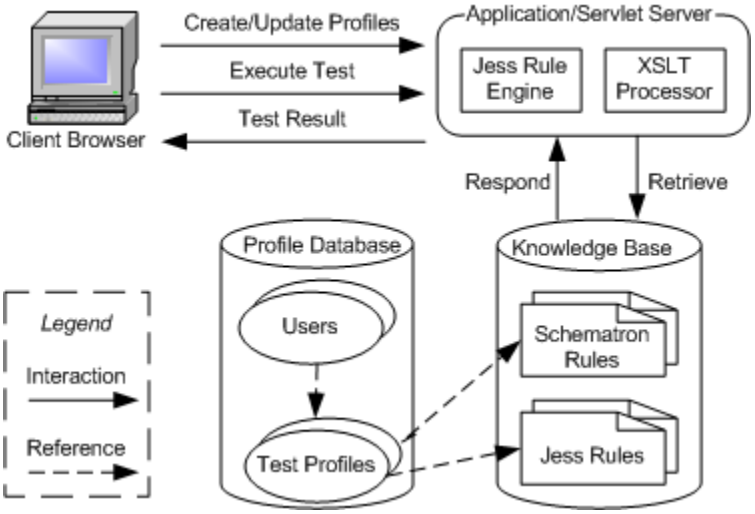
The rules are encoded in either the Schematron assertion language or in the Jess (Java™ Expert System Shell) [Jess] expert system rule language[1]. Other rule or scripting languages may be supported in the future. Because QoD is implemented in Java, any interpreted language implemented in Java should be easy to incorporate into QoD's architecture. For example, Jython [Jython], a Java implementation of the Python scripting language (which has numerous XML processing libraries) could be supported.

Jess is a forward chaining rule engine and scripting environment. Using Jess, one can build Java software that has the capacity to "reason" using knowledge supplied in the form of declarative rules. Jess scripts can also contain embedded Java code. QoD converts the XML document to be tested into a Jess knowledge base. The person writing Jess rules needs to understand this transformation. Discussion of the transformation and examples of Jess rules are beyond the scope of this paper, but the QoD currently includes an implementation of UBL rule GNR1 (see "Automating a Rule").

The QoD architecture is based on a detailed analysis [Kul04b] NIST conducted of NDR documents from a variety of standards bodies and industry groups. For each rule, NIST determined the ease of making the rule computer-interpretable, the rule's rationale, and the rule's scope (i.e., whether the rule applies to a local context, a grouping of definitions such as a complex type, or to an entire schema collection). This NDR metadata not only helped us choose which rules to initially encode for the QoD, but also could be useful to XML developers in assembling an NDR for their project from already-existing NDRs.

QoD uses a *test profile* mechanism for grouping rule implementations into a related set of requirements. When the user executes a test profile, all the included implementations are executed (except those not applicable according to the scope specified by the user). Test results generated from test profile executions are stored for subsequent reviews. Users may also add additional encoded rules to the QoD repository, as well as save and view test results. Figure 2 illustrates the design of the QoD Web application and shows the interactions and data relationships between test profiles, the Schematron and Jess-implemented rules, the application server, and the user's browser client.

Figure 2: QoD Web Prototype Design

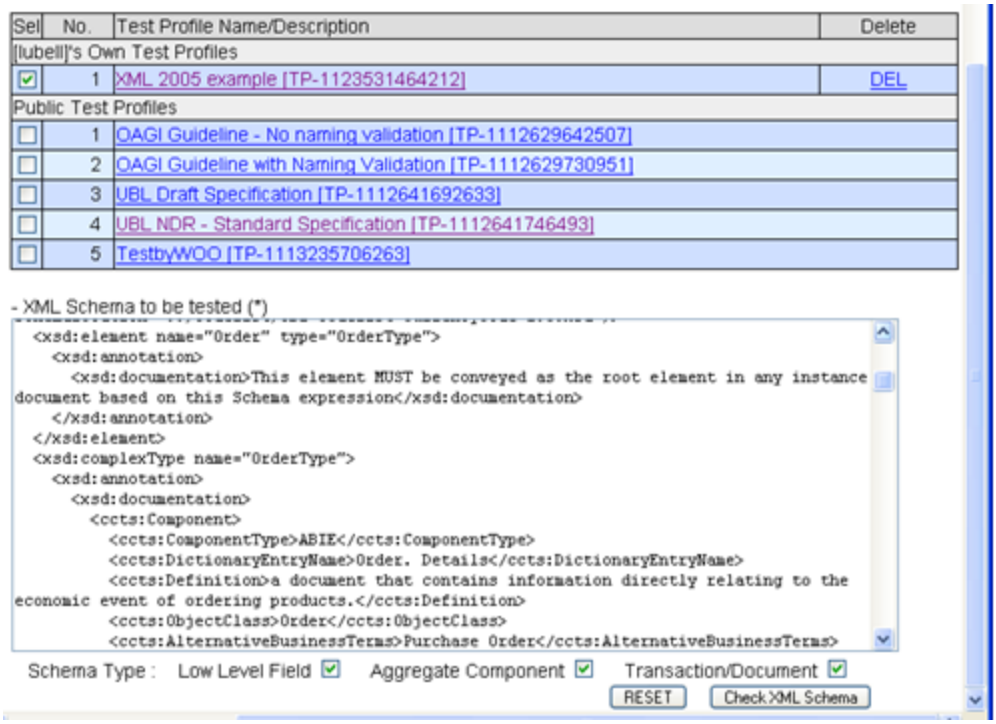


QoD helps ensure that XML schemas comply with designated sets of best practice rules and organizational specific XML schema design requirements. Best practice rules may be drawn from published NDR documents, experienced system integrators, or XML architects. Tests within this category seek to enhance the usability, reusability, and interoperability of the schema such as by increasing the schema's ability to capture and enforce desired semantics, extensibility, its ease of maintenance, and its implementation and processing efficiency. Subsets of those requirements have been implemented and stored within the repository of the Web-based prototype.

Although QoD test profiles are useful tools for encouraging best schema design practices, the test profile mechanism has a potential pitfall. If only a small easy-to-implement subset of an NDR is made available as a test profile, there is a danger XML developers might think that a schema conforming to the test profile also obeys all rules in the NDR. It is therefore important that test profile developers thoroughly and conspicuously document the test profile's limitations to avoid misunderstanding on the part of test profile users.

Figure 3 shows a QoD screen shot of a user checking the UBL Order schema against a test profile created specifically for our ELD1 example. The "Schema Type" checkboxes near the bottom of the figure give the user the option of limiting the application of rules in the test profile according to their locality. This is needed because not all rules are applicable to all schemas. For example, rule ELD1 should only be applied to document schemas and should not be applied to "library" schemas containing reusable components imported by multiple document schemas. In Figure 3, the user has chosen to apply all rules in the test profile.

Figure 3: Executing a UBL NDR Test Profile



QoD also facilitates the management of rules and their implementations. Users may specify more than one rule for a test requirement as well as associate a version (e.g., "draft" versus "final") with each rule. Rules can have more than one implementation, allowing for deprecated implementations, or the possibility of alternative implementation languages for a single rule. For example, a rule could have a cursory Schematron implementation as well as a more thorough Jess implementation. The Schematron implementation, even though it only partially enforces the rule, has the advantage that it could be reused in testing environments without access to a Jess engine.

It is important to remember that the most valuable part of QoD is its knowledge base of implemented Naming and Design Rules. While the QoD Web application is useful for experimentation, it does not serve the needs of users who need their own local NDR testing environment, or who want to keep their test history private. Nor does it serve the needs of users who need to test a collection of schemas against a single test profile, or who want an off-line environment for schema testing.

To give schema developers more flexibility, we have created a QoD XML exchange format so that QoD can export to or import test profiles from other testing applications. We provide a schema governing this format in "Appendix: QoD Exchange Schema". The exchange format attempts to capture QoD metadata that could be useful in third-party testing environments. An XML exchange document representing a UBL NDR test profile might appear as follows:

```
<testProfile>
  <source id="ubl">
    <organization>OASIS</organization>
    <orgURL>http://www.oasis-open.org</orgURL>
    <title>Universal Business Language (UBL) Naming and Design Rules</title>
    <version>1.0</version>
    <date>2004-11-15</date>
    <docURL>http://docs.oasis-open.org/ubl/cd-UBL-NDR-1.0.1</docURL>
  </source>
  <ruleSet id="ELD">
    <name>Element Declaration Rules</name>
    <rule id="ELD1">
      <coverage>full</coverage>
      <schema>D</schema>
      <rationale>structural clarity</rationale>
      <requirement>Each UBL:DocumentSchema MUST identify one and
      ...
      </requirement>
      <implementation figname="example.scmt#eld1" type="schematron"/>
    </rule>

      ...

</testProfile>
```

The child elements within `rule` provide the following metadata:

Table 1

Element	Semantics
`coverage`	**full** means that the rule is fully verifiable. Alternatively, a rule may be partially verifiable or unverifiable.
`schema`	**D** indicates the rule is applicable to document level schemas, i.e., schemas consisting mainly of global element declarations and complex type definitions. Other schema types are low level schemas and aggregate level schemas. See [Kul04b] for additional explanation regarding schema types.
`rationale`	Provides one or more reasons justifying the rule, from a list of nine possible justifications.
`requirement`	The text describing the rule, taken verbatim from the NDR document.
`implementation`	Specifies the URI of the computer-interpretable representation of the rule as well as the implementation method.

§ Conclusion

Experience with the QoD has shown benefits both in terms of developing schemas that consistently comply with an NDR and in terms of specifying an NDR that is usable. The Web application has successfully exposed design issues in various organizations' proposed XML exchange schemas.

QoD has also been used to assemble test profiles corresponding to subsets of NDRs from UBL, the DON, and others, providing a convenient testing platform. The QoD XML exchange format enables test profiles to be shared with third party applications, expanding the utility of implemented rules in the QoD knowledge base. The exchange mechanism also enables developers of XML schema generation tools to use implemented rules to guarantee that generated schemas meet interoperability and quality guidelines. This use case will become increasingly important as software tools improve to the point where specifying a schema using a modeling language such as UML [UML] becomes the norm for XML developers, and writing W3C XML Schema definitions becomes a thing of the past.

Finally, the activity of encoding rules highlights the difficulty and benefits of writing rules that are both usable and testable. High quality rules are a prerequisite for the rules to be consistently applied.

The "Motivation" section lists four obstacles to NDR effectiveness: proliferation, lack of structure, lack of automation, and limited versioning/traceability. QoD clearly addresses lack of automation and, through its test profile mechanism and ability to distinguish between multiple versions of rules and multiple implementations of a rule, provides a foundation for ensuring traceability. Although QoD does not provide a direct solution for the lack of structure in NDR documents, a variant of the QoD exchange schema could be used as an XML schema for structuring NDR documents. And although QoD alone will not stem the proliferation of NDRs, it does facilitate reuse of rules from existing NDRs[3]. By encouraging implementation, the QoD approach can add more discipline to NDR development.

A major issue that QoD does not address is ensuring the correctness of rule implementations. In "Automating a Rule", we showed that implementing rules is non-trivial and potentially error-prone. How can a developer be sure that an implementation actually checks what the rule says should be checked?

Correctness of rule implementation starts with clear and concise rule definition. That clarity can be achieved by encoding the rules. It is in the encoding process that intention and ambiguity are flushed out. "Issues Affecting Rule Testability" described some pitfalls to be avoided in encoding rules. It also touched on the expectation of what can be achieved by encoding and where mandates and guidance diverge. A rigorous testing framework is needed to manage the encoding process. Perhaps existing approaches to software testing can be adapted to facilitate the effort.

As difficult as NDR implementation is, automating rules is easy relative to the challenge of agreeing on a set of rules in the first place. Developing a consensus standard requires political skills as well as technical skills. Our approach does not by itself foster agreement among disparate organizations with a need to exchange data. However, through removing ambiguity and encouraging reuse, QoD should at least make it clearer to standards developers specifically what it is they are agreeing to while at the same time allowing them to leverage the work of other organizations.

§ Appendix: QoD Exchange Schema

We present a proposed exchange schema for representing the Naming and Design Rules in a QoD test profile. The schema is provided both in RELAX NG [RELAXNG] and in the W3C XML Schema Definition Language, the latter generated from the former using James Clark's *Trang* [Trang] software.

RELAX NG Schema (Compact Syntax)

```
start =
   element testProfile { TestProfile }

## specifies a unique identifier
id.att =
   attribute id { xsd:ID }

## specifies a cross-reference to a unique identifier
ref.att =
   attribute ref { xsd:IDREF }

## collection of sets of naming and design rules
TestProfile =
   element source { Source }+,
   element ruleSet { RuleSet }*

## information describing a naming and design rules document
Source =
   id.att?,
   element organization { text },
   element orgURL { xsd:anyURI }?,
   element title { text },
   element version { text }?,
   element date { xsd:date }?,
   element docURL { xsd:anyURI }?

## collection of naming and design rules
RuleSet =
   id.att?,
   element name { text },
   element owner { text }?,
   element ownerEmail { xsd:anyURI }?,
   element source { ref.att }?,
   element rule { Rule }+

## individual naming and design rule
Rule =
```

```
   id.att?,
   element coverage { "full" | "partial" | "not applicable" }?,
   element schema { "L" | "LA" | "A" | "AD" | "D" | "LAD" }?,
   element rationale { Rationale }*,
   element source { ref.att }?,
   element requirement { text },
   element implementation { Implementation }*

## computer-interpretable code implementing a rule
Implementation =
   attribute type { "jess" | "schematron" },
   attribute file { xsd:anyURI }

## justification for a rule
Rationale =
   "validation and model clarity" |
   "structural clarity" |
   "clarity" |
   "extensibility" |
   "common symbolic syntax" |
   "maintainability" |
   "performance" |
   "interoperability" |
   "model validity"
```

W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="testProfile" type="TestProfile"/>
  <xs:attributeGroup name="id.att">
    <xs:annotation>
      <xs:documentation>specifies a unique identifier</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" use="required" type="xs:ID"/>
  </xs:attributeGroup>
  <xs:attributeGroup name="ref.att">
    <xs:annotation>
      <xs:documentation>specifies a cross-reference to a unique
      identifier</xs:documentation>
    </xs:annotation>
    <xs:attribute name="ref" use="required" type="xs:IDREF"/>
  </xs:attributeGroup>
  <xs:complexType name="TestProfile">
    <xs:annotation>
      <xs:documentation>collection of sets of naming and design
      rules</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="source" type="Source"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="ruleSet"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ruleSet" type="RuleSet"/>
  <xs:complexType name="Source">
    <xs:annotation>
      <xs:documentation>information describing a naming and design
      rules document</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="organization"/>
      <xs:element minOccurs="0" ref="orgURL"/>
      <xs:element ref="title"/>
      <xs:element minOccurs="0" ref="version"/>
      <xs:element minOccurs="0" ref="date"/>
      <xs:element minOccurs="0" ref="docURL"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
  </xs:complexType>
  <xs:element name="organization" type="xs:string"/>
  <xs:element name="orgURL" type="xs:anyURI"/>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="version" type="xs:string"/>
  <xs:element name="date" type="xs:date"/>
  <xs:element name="docURL" type="xs:anyURI"/>
  <xs:complexType name="RuleSet">
    <xs:annotation>
      <xs:documentation>collection of naming and design
      rules</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element minOccurs="0" ref="owner"/>
      <xs:element minOccurs="0" ref="ownerEmail"/>
      <xs:element minOccurs="0" name="source">
        <xs:complexType>
          <xs:attributeGroup ref="ref.att"/>
        </xs:complexType>
```

```
      </xs:element>
      <xs:element maxOccurs="unbounded" ref="rule"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
  </xs:complexType>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="owner" type="xs:string"/>
  <xs:element name="ownerEmail" type="xs:anyURI"/>
  <xs:element name="rule" type="Rule"/>
  <xs:complexType name="Rule">
    <xs:annotation>
      <xs:documentation>individual naming and design rule</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element minOccurs="0" ref="coverage"/>
      <xs:element minOccurs="0" ref="schema"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="rationale"/>
      <xs:element minOccurs="0" name="source">
        <xs:complexType>
          <xs:attributeGroup ref="ref.att"/>
        </xs:complexType>
      </xs:element>
      <xs:element ref="requirement"/>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="implementation"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
  </xs:complexType>
  <xs:element name="coverage">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="full"/>
        <xs:enumeration value="partial"/>
        <xs:enumeration value="not applicable"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="schema">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="L"/>
        <xs:enumeration value="LA"/>
        <xs:enumeration value="A"/>
        <xs:enumeration value="AD"/>
        <xs:enumeration value="D"/>
        <xs:enumeration value="LAD"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="rationale" type="Rationale"/>
  <xs:element name="requirement" type="xs:string"/>
  <xs:element name="implementation">
    <xs:complexType>
      <xs:attributeGroup ref="Implementation"/>
    </xs:complexType>
  </xs:element>
  <xs:attributeGroup name="Implementation">
    <xs:annotation>
      <xs:documentation>computer-interpretable code implementing
      a rule</xs:documentation>
    </xs:annotation>
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:enumeration value="jess"/>
          <xs:enumeration value="schematron"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="file" use="required" type="xs:anyURI"/>
  </xs:attributeGroup>
  <xs:simpleType name="Rationale">
    <xs:annotation>
      <xs:documentation>justification for a rule</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:enumeration value="validation and model clarity"/>
      <xs:enumeration value="structural clarity"/>
      <xs:enumeration value="clarity"/>
      <xs:enumeration value="extensibility"/>
      <xs:enumeration value="common symbolic syntax"/>
      <xs:enumeration value="maintainability"/>
      <xs:enumeration value="performance"/>
      <xs:enumeration value="interoperability"/>
      <xs:enumeration value="model validity"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Notes

1. Commercial equipment and materials are identified in order to describe certain procedures. In no case does such identification imply recommendation or endorsement by the authors or their organization, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose. Java is a trademark or registered trademark of Sun Microsystems, Inc. Other company, product, and service names may be trademarks or service marks of others.
2. If a digital dictionary with an XML interface were available, a Schematron processor might be able to use the XSLT [Extensible Style Language Transformation] `document()` function [XSLT] to implement the lookup.
3. QoD is its current form is not very proactive about helping NDR designers discover whether and how they can reuse existing rules and implementations. However, the structured representation of rules in the QoD's knowledge base could support such a discovery capability. This would be a ripe area for future research.

Acknowledgements

The authors thank Puja Goyal, Gary Kramer, Alexander Roth, Ronny Jopp, and the Extreme Markup Languages peer review panel for their helpful comments on earlier drafts of this paper.

Bibliography

[CCTS] United Nations Centre for Trade Facilitation and Electronic Business, *Core Components Technical Specification - Part 8 of the ebXML Framework*, 15 November 2003, Version 2.01.

[DON] Office of the DON: Chief Information Officer, *Department of the Navy XML Naming and Design Rules*, Final Version 2.0, January 2005, **http://www.doncio.navy.mil**.

[ISO11179-1] ISO/IEC 11179-1:2004(E), *(2nd edition) Information technology Metadata Registries (MDR) Part 1: Framework*, 2004, **http://metadata-standards.org/11179/**.

[ISO11179-5] ISO/IEC 11179-5:1995(E), *(1st edition) Information technology Metadata Registries (MDR) Part 5: Naming and identification principles*, 1995, **http://metadata-standards.org/11179/**.

[ISOSch] ISO/IEC FDIS 19757-3:2004, *Document Schema Definition Languages (DSDL) - Part 3: Rule-based validation - Schematron*, 2004, **http://www.schematron.com**.

[Jeliffe] Jeliffe, R., *The Schematron Assertion Language 1.5*, 2002-10-01, **http://xml.ascc.net/schematron/**.

[Jess] Jess, the Rule Engine for the Java™ Platform, **http://herzberg.ca.sandia.gov/jess**.

[Jython] Jython Home Page, **http://www.jython.org**.

[Kul04a] Kulvatunyou, B., Ivezic, N., and Bhuwan, J., *Testing Requirements to Manage Data Exchange Specifications in Enterprise Integration - A Schema Design Quality Focus*, 8th World Multiconference on Systems, Cybernetics and Informatics (SCI), Orlando, Florida, July 2004, **http://www.nist.gov/msidlibrary/doc/testing_design.pdf**.

[Kul04b] Kulvatunyou, Boonserm, Morris, KC , *XML Schema Design Quality Test Requirements*, National Institute of Standards and Technology, NIST IR 7175, 2004, **http://www.nist.gov/msidlibrary/doc/Schema_Design.pdf** .

[QoD] Manufacturing Systems Integration Quality of Design Tool, **http://www.mel.nist.gov/msid/QOD**.

[RELAXNG] OASIS, *RELAX NG Compact Syntax*, Committee Specification, 21 November 2002, **http://www.oasis-open.org/committees/relax-ng/compact-20021121.html** .

[Skonnard] Skonnard, Aaron, *The XML Files: Object Graphs, XPath, String Comparisons, and More*, MSDN Magazine, January 2002, **http://msdn.microsoft.com/msdnmag/issues/02/01/**.

[Trang] Trang, **http://www.thaiopensource.com/relaxng/trang.html**.

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

[UBLNDR] OASIS, *Universal Business Language (UBL) Naming and Design Rules*, cd-UBL-NDR-1.0.1, 15 November 2004, **http://www.oasis-open.org/committees/ubl**.

[UML] Object Management Group, *Unified Modeling Language: Superstructure*, version 2.0, August 2005.

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

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

[XPath2] World Wide Web Consortium, *XML Path Language (XPath) Version 2.0*, W3C Candidate Recommendation, 3 November 2005, **http://www.w3.org/TR/xpath20**.

[XSD] World Wide Web Consortium, *XML Schema: Primer Second Edition*, W3C Recommendation, 28 October 2004, **http://www.w3.org/TR/xmlschema-0/**.

[XSLT] World Wide Web Consortium, *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, 16 November 1999, **http://www.w3.org/TR/xslt**.

The Authors

Joshua Lubell
National Institute of Standards and Technology
Gaithersburg
Maryland
USA

Josh Lubell works at NIST [the National Institute of Standards and Technology] where he applies markup technology toward solving data exchange problems between manufacturing applications. He is a contributor to various standards efforts and speaks regularly at XML-related conferences. His pre-NIST experience includes artificial intelligence systems design and prototyping as well as software development for the building materials industry. He has an M.S. in computer science from the University of Maryland at College Park and a B.S. in mathematics from Binghamton University.

Boonserm (Serm) Kulvatunyou
National Institute of Standards and Technology
Gaithersburg
Maryland
USA

Serm is a principal architect of the Manufacturing B2B Testbed at the National Institute of Standards and Technology. He has helped the auto and the capital facility industry in XML standards development and conformance testing. He has also contributed to the ebXML Business Process and Core Components standards.

KC Morris
National Institute of Standards and Technology
Gaithersburg
Maryland
USA

KC Morris is a computer scientist at the National Institute of Standards and Technology. Her work involves methods and mechanisms for supporting the specification and development of information interface standards and the testing of those standards. Currently she is working on projects to develop and test XML Schema-based interfaces. Earlier in her career she worked on STEP, a.k.a. ISO 10303 The Standard for the Exchange of Product Model Data. STEP is a family of ISO standards for sharing technical data about products. She was a primary contributor to the development and validation of STEP's implementation mechanisms. She has architected and contributed to the development of numerous prototypes of the STEP Data Access Interface, including NIST's public-domain STEP Toolset. KC has been a contributor to the ISO working groups developing STEP and to several industrial consortiums, including FIATECH, PDES, Inc. and National Industrial Information Infrastructure Protocols Program (NIIIP). Her current research interests include data sharing for engineering and manufacturing and techniques for managing and testing such systems.

Betty Harvey
Electronic Commerce Connection, Inc.
Glen Burnie
Maryland
USA

Ms. Harvey is President of Electronic Commerce Connection, Inc. Ms. Harvey has participated with many commercial and Government organization in planning and executing their migration to structured information. She has been involved in many international XML standards initiatives such as ebXML, UBL, STEP and multiple vertical industry XML industry standards. She has authored numerous articles in international publications on XML technologies as well as spoken at many technology conferences.

Ms. Harvey started and coordinates the Washington, D.C. Area SGML/XML Users Group. Prior to starting ECC, Inc., Ms. Harvey worked in Scientific and Engineering Computing at David Taylor Model Basic, NSWC.

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