A Tool Kit for Implementing XML Schema Naming and Design Rules

Joshua **Lubell**

Boonserm (Serm) **Kulvatunyou**

Abstract

A tool kit being developed at the National Institute of Standards and Technology (NIST) encodes XML schema Naming and Design Rules in a computer-interpretable fashion, enabling automated rule enforcement and improving schema quality.

Table of Contents

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.

1. Motivation

Organizations developing XML schemas [XSD] often establish Naming and Design Rules (NDRs) in order to maximize interoperability and quality. NDRs are a good way to help enforce best practices, a particular modeling methodology (such as the Core Components Technical Specification (CCTS) [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. On the other hand, 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. If schemas are automatically generated using a software tool, then NDR conformance can be built into the schema generation process.

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

This paper explores methods for removing these barriers. We first discuss some implementation issues, using a rule from the Universal Business Language (UBL) NDR as an example. Next, we present a methodology for implementing NDRs and a prototype system using our methodology. We conclude with some observations regarding our experience.

2. Automating a Rule: An Example

Consider the following rule from the UBL NDR [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 Aggregate Business Information Entity (ABIE). 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. [UBL] 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 creating a grammar for an XML document, a Schematron schema makes assertions applied to a specific context path within the document. If an XML document fails to meet the 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 is more expressive than grammar-based schema languages.

The following Schematron schema, which shows Schematron's expressive power, implements rule ELD1:

```
<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.

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. 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, it is more difficult to write XML schema Naming and Design Rules. Rather than expressing XPaths directly, the scripting language must use an application programmer interface (API) to access the XML. As a result, scripting language implementations are likely to be more verbose and difficult to understand than Schematron implementations.

3. An NDR Implementation Tool Kit

So far we have provided examples of rules and discussed alternative implementation methods. We have not yet talked about the issues of NDR proliferation and difficulties with reuse of rules. To help address these issues, the National Institute of Standards and Technology (NIST) is building a “Quality of Design” (QoD) 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 (see the note regarding commercial products at the beginning of this paper). 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 previous section).

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 has a local context, applies to grouping of definitions such as a complex type, or applies 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 1 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.

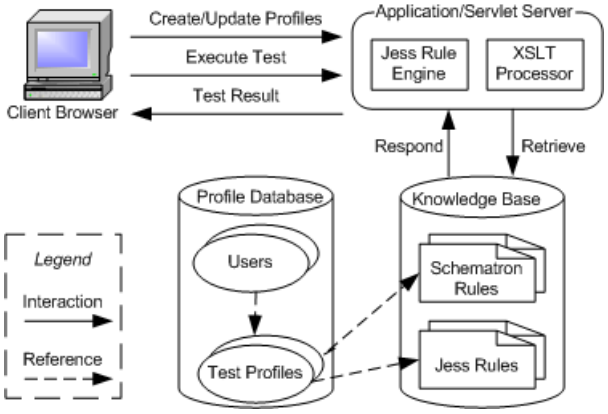

Figure 1. QoD Web Prototype Design

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.

Figure 2 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 for 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 2, the user has chosen to apply all rules in the test profile.

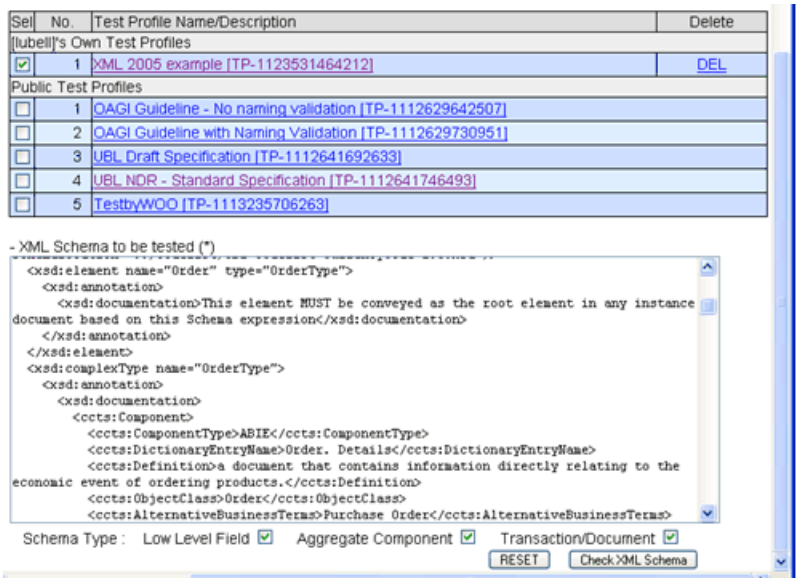

Figure 2. Executing a UBL NDR Test Profile

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 an appendix. 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 file="example.scmt#eld1" type="schematron"/>
    </rule>

      ...

</testProfile>
```

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

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 Uniform Resource Indicator (URI) of the computer-interpretable representation of the rule as well as the implementation method.

4. Conclusion

Initial experience with the QoD approach is promising. The Web application has successfully exposed design issues in various organizations' XML exchange schemas, including the US Department of Defense. QoD has also been used to assemble test profiles corresponding to subsets of UBL and other NDRs, providing a convenient conformance 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.

In the Motivation section, we listed 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. 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 the Automating a Rule section, 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? A rigorous testing framework is needed. Perhaps existing approaches to software testing can be adapted for this purpose.

But 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.

Acknowledgements

We wish to thank Puja Goyal and Fred Byers for their helpful comments in reviewing an earlier draft 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.

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

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

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

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

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

[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. Available at 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. Available at http://www.nist.gov/msidlibrary/doc/Schema_Design.pdf .

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

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

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

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

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

A. 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 World Wide Web Consortium (W3C) XML Schema Definition Language, the latter generated from the former using James Clark's *Trang* [Trang] software.

1. RELAX NG Schema (Compact Syntax)

```
# $Id: xml2005.xml,v 1.16 2005/09/20 13:59:10 lubell Exp $

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"
```

2. W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <!-- $Id: xml2005.xml,v 1.16 2005/09/20 13:59:10 lubell Exp $ -->
  <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>
```

Biography

Joshua **Lubell**
National Institute of Standards and Technology [http://www.nist.gov/]
Gaithersburg
Maryland
United States of America

Josh Lubell works at US National Institute of Standards and Technology (NIST) 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 [http://www.nist.gov/]
Gaithersburg
Maryland
United States of America

Serm is a principle 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 BP and CC standards.