ISO/IEC JTC 1/SC 34

Date: 2004-05-31

ISO/IEC CD 19757-4

ISO/IEC JTC 1/SC 34/WG 1

Secretariat: Standards Council of Canada

Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language — NVDL

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International Standard
Document subtype:
Document stage: (30) Committee
Document language: E

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

ISO/IEC 19757-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 34, Document Description and Processing Languages.

ISO/IEC 19757 consists of the following parts, under the general title *Document Schema Definition Languages (DSDL)*:

— *Part 1: Overview*

— *Part 2: Regular-grammar-based validation — RELAX NG*

— *Part 3: Rule-based validation —Schematron*

— *Part 4: Namespace-based validation dispatching language — NVDL*

— *Part 5: Datatypes*

— *Part 6: Path-based integrity constraints*

— *Part 7: Character repertoire validation*

— *Part 8: Declarative document manipulation*

— *Part 9: Datatype- and namespace-aware DTDs*

— *Part 10: Validation management*

Introduction

This International Standard defines a set of Document Schema Definition Languages (DSDL) that can be used to specify one or more validation processes performed against Extensible Markup Language (XML) documents. A number of validation technologies are standardized in DSDL to complement those already available as standards or from industry.

The main objective of this International Standard is to bring together different validation-related technologies to form a single extensible framework that allows technologies to work in series or in parallel to produce a single or a set of validation results. The extensibility of DSDL accommodates validation technologies not yet designed or specified.

The motivations of this part of ISO/IEC 19757 are twofold. One is to allow the interworking of schemas describing different markup vocabularies. The other is to allow these schemas to be written in different schema languages. For this purpose, this part of ISO/IEC 19757 specifies a Namespace-based Validation Dispatching Language (NVDL).

The structure of this part of ISO/IEC 19757 is as follows. Clause 5 describes the data model, which is the abstraction of an XML document used throughout the rest of the document. Clause 6 describes the full syntax and the simple syntax of NVDL scripts, and further describes the transformation from the full syntax to the simple syntax. Clause 7 describes primitive operations for the NVDL data model, which are used for defining the NVDL semantics. Clause 8 describes the semantics of a correct NVDL script in the simple syntax; the semantics specify how elements and attributes in a given document are dispatched to different validators and which schema is used by each of these validators. Clause 9 describes conformance requirements for NVDL dispatchers. Annex A and Annex B define the full syntax and the simple syntax using RELAX NG, respectively. Annex C defines the full syntax using NVDL and RELAX NG. Finally, Annex D provides examples of the application of NVDL.

The history of NVDL is as follows. First, JIS TR X 0044[2] was created and then submitted to ISO/IEC JTC1 as a fast-track ISO/IEC DTR 22250-2[3] by the Japanese national member body for SC 34. A simplification of this DTR became the first committee draft of this part of the International Standard in 2002. MNS[4], Namespace Switchboard[5], and NRL[6] were designed under the influence of RELAX Namespace and the committee draft. On the basis of these inputs, the second committee draft of this part was created in 2004.

Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language — NVDL

1 Scope

This part of the International Standard specifies a Namespace-based Validation Dispatching Language (NVDL). An NDVL schema controls the dispatching of elements or attributes in a given XML document to different validators, depending on the namespaces of the elements or attributes. An NDVL schema also specifies which schemas are used by these validators. These schemas may be written in any schema languages, including those specified by this International Standard.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Each of the following documents has a unique identifier that is used to cite the document in the text. The unique identifier consists of the part of the reference up to the first comma.

RELAX-NG, *ISO/IEC 19757-2, Document Schema Definition Languages (DSDL) — Part 2: Grammar-based validation — RELAX NG*

W3C XML, *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation, 04 February 2004, available at <http://www.w3.org/TR/2004/REC-xml-20040204/>

W3C XML-Names, *Namespaces in XML*, W3C Recommendation, 14 January 1999, available at <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

W3C XLink, *XML Linking Language (XLink) Version 1.0*, W3C Recommendation, 27 June 2001, available at <http://www.w3.org/TR/2001/REC-xlink-20010627/>

W3C XML-Infoset, *XML Information Set (Second Edition)*, W3C Recommendation, 4 February 2004, available at <http://www.w3.org/TR/2004/REC-xml-infoset-20040204/>

W3C XML Schema Part 2, *XML Schema Part 2: Datatypes*, W3C Recommendation, 02 May 2001, available at <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, Internet Standards Track Specification, November 1996, available at <http://www.ietf.org/rfc/rfc2045.txt>

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, Internet Standards Track Specification, November 1996, available at <http://www.ietf.org/rfc/rfc2046.txt>

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, Internet Standards Track Specification, August 1998, available at <http://www.ietf.org/rfc/rfc2396.txt>

IETF RFC 2732, *Format for Literal IPv6 Addresses in URL's*, Internet Standards Track Specification, December 1999, available at <http://www.ietf.org/rfc/rfc2732.txt>

IETF RFC 3023, *XML Media Types*, Internet Standards Track Specification, August 1998, available at <http://www.ietf.org/rfc/rfc3023.txt>

3 Terms and definitions

For the purposes of this part of ISO/IEC 19757, the following terms and definitions apply.

3.1
resource
something with identity, potentially addressable by a URI
[RELAX-NG]

3.2
URI
compact string of characters that uses the syntax defined in IETF RFC 2396 to identify an abstract or physical resource
[RELAX-NG]

3.3
URI reference
URI or relative URI and optional fragment identifier
[RELAX-NG]

3.4
relative URI
form of URI reference that can be resolved with respect to a base URI to produce another URI
[RELAX-NG]

3.5
base URI
URI used to resolve relative URIs
[RELAX-NG]

3.6
fragment identifier
additional information in a URI reference used by a user agent after the retrieval action on a URI has been successfully performed
[RELAX-NG]

3.7
media type
a two-part identifier specifing the nature of the referenced data

3.8
instance
XML document from which validation candiates are created

3.9
whitespace character
character with the code value #x20, #x9, #xA or #xD
[RELAX-NG]

3.10
name
pair of a URI and a local name
[RELAX-NG]

3.11
namespace URI
URI that is part of a name
[RELAX-NG]

3.12
local name
NCName that is part of a name

[RELAX-NG]

3.13
NCName
string that matches the NCName production of W3C XML-Names
[RELAX-NG]

3.14
full syntax
syntax of an NVDL grammar before simplification

NOTE This term is used differently in RELAX-NG.

3.15
simple syntax
syntax of an NVDL script after simplification

NOTE This term is used differently in RELAX-NG.

3.16
simplification
transformation of an NVDL script in the full syntax to a script in the simple syntax

NOTE This term is used differently in RELAX-NG.

3.17
valid with respect to a schema
member of the set of XML documents described by the schema
[RELAX-NG]

3.18
schema
specification of a set of XML documents
[RELAX-NG]

3.19
NVDL script
specification of namespace-based validation dispatching

3.20
validator
software module that determines whether a schema is correct and whether an instance is valid with respect to a schema
[RELAX-NG]

3.21
dispatcher
software module that determines whether an NVDL script is correct, creates validation candidates from an instance, and invokes validators for these validation candidates.

3.22
path
list of NCNames separated by / or //
[RELAX-NG]

3.23
path expression
a list of one or more choices separated by |, where each choice is a list of one or more NCNames separated by /, optionally preceded by /

3.24
infoset
an abstraction of an XML document defined by W3C XML-Infoset

[RELAX-NG]

3.25
RELAX NG data model
abstract representation of an XML document defined by RELAX-NG

3.26
NVDL data model
abstract representation of an XML document defined by this part of ISO/IEC 19757

3.27
XML document
string that is a well-formed XML document as defined in W3C XML
[RELAX-NG]

3.28
section
either an attribute section or element section

3.29
attribute section
a non-empty set of attributes having the same namespace URI

3.30
element section
an element for which a single namespace URI applies to itself and all descendants

3.31
slot node
either an attribute slot node or element slot node

3.32
attribute slot node
a place holder for an attribute section

3.33
element slot node
a place holder for an element section

3.34
validation candidate
an element not having any slot nodes as descendants

NOTE Different elements in a validation candidate may belong to different namespaces. Different attributes of an element in a validation candidate may belong to different namespaces.

3.35
mode
a mapping from namespaces to actions

3.36
action
validate, reject, allow, attach, or unwrap element

4 Notation

< [..] >: the context of an element in the NVDL data model,

< {..} >: the namespace URI of an element in the NVDL data model

(..,..): a sequence of elements

{..,..}: a set of attributes

x+y: the concatenation of two element sequences *x* and *y*

e <- *x*: the result of attaching an element sequence or attribute section *x* to *e*

|*e*|: the number of elements in *e*

removeSlots(*s*): the result of removing slot nodes from an element section *s* as specified in 7.5

dummyElement(*s*): the result of converting an attribute section *s* to an empty element as specified in 7.6

path(*s*): the path from the root of *s'* to the parent element of the place holder of *s*, where *s'* is the parent element section of *s*

5 Data model

5.1 General

NVDL deals with XML documents representing both schemas and instances through an abstract data model. XML documents representing schemas and instances shall be well-formed in conformance with W3C XML and shall conform to the constraints of W3C XML-Names.

This abstract NVDL data model is an extension of the data model of RELAX-NG. Names, contexts, attributes, and strings in the NVDL data model are identical to those in the RELAX NG data model. Elements in the NVDL data model are extended as describe in this clause.

An element consists of:

— a name,

— a context,

— a set of attributes or attribute slot nodes, and

— an ordered sequence of zero or more children; each child is either an element, a non-empty string, or an element slot node; the sequence never contains two consecutive strings.

An element (say *e*) is said to be a descendant of another element (say *e'*) when *e* occurs in the child sequence of *e'* or that of some descendant element of *e*. Meanwhile, *e* is said to be an ancestor of *e'* when *e'* is a descendant of *e*.

NOTE 1 An element (say *e*) is not a descendant of another element (say *e'*) if *e'* has an element slot node that is the place holder for *e*.

An attribute slot node is a place holder for a non-empty set of attributes . An attribute slot node is said to belong to an element (say *e*) when this attribute slot node is contained by the set of attributes or attribute slot nodes of *e* or that of some descendant element of *e*.

An element slot node is a place holder for an element. An element slot node is said to belong to an element (say *e*) when this element slot node occurs in the child sequence of *e* or that of some descendant element of *e*.

NOTE 2 Attribute or element slot nodes do not exist in the RELAX NG data model.

5.2 Creating a data model from the infoset

This process is identical to the process defined in RELAX-NG.

EXAMPLE1 Consider an XML document as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:foo xmlns:ns1="http://www.example.com/one" xmlns:ns2="http://www.example.com/two">
<!-- comment-example -->
<?pi-example?>
   <ns1:foo1>
      text1
      <ns2:foo11/>
   </ns1:foo1>
   <ns2:foo2>
      text2
      <ns1:foo21/>
      text3
      <ns1:foo22/>
   </ns2:foo2>
   <ns1:foo3>
      <ns1:foo31/>
   </ns1:foo3>
</ns1:foo>
```

We use an XML-like syntax for representing data models, but there are two changes. First, in each start tag, the tag name is replaced by a URI, local name, and context. Second, each end tag is represented by "</>".

The data model created from this document is shown below. (Whitespace is slightly changed for readability.) Every element in this data model references to a context (say cx1) that maps ns1 to "http://www.example.com/one" and ns2 to"http://www.example.com/two", respectively.

```
<{http://www.example.com/one}foo [cx1]>
   <{http://www.example.com/one}foo1 [cx1]>
      text1
      <{http://www.example.com/two}foo11 [cx1]></>
   </>
   <{http://www.example.com/two}foo2 [cx1]>
      text2
      <{http://www.example.com/one}foo21 [cx1]></>
      text3
      <{http://www.example.com/one}foo22 [cx1]></>
   </>
   <{http://www.example.com/one}foo3 [cx1]>
      <{http://www.example.com/one}foo31 [cx1]></>
   </>
</>
```

Observe that the XML declaration and any comments or processing instructions in the input XML document do not appear in the data model and that there are no empty-element tags.

EXAMPLE2 The following XML document is obtained by adding attributes to the document in the previous example.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:foo xmlns:ns1="http://www.example.com/one" xmlns:ns2="http://www.example.com/two">
<!-- comment-example -->
<?pi-example?>
   <ns1:foo1 bar1="_" bar2="_">
      text1
      <ns2:foo11/>
   </ns1:foo1>
   <ns2:foo2 ns1:bar1="_" ns1:bar2="_" ns2:bar1="_" ns2:bar2="_" bar1="_" bar2="_">
      text2
      <ns1:foo21/>
      text3
      <ns1:foo22/>
```



```
    </ns2:foo2>
    <ns1:foo3 ns2:bar1="_" ns2:bar2="_">
      <ns1:foo31/>
    </ns1:foo3>
  </ns1:foo>
```

The data model created from this document is shown below. cx1 is the same as in Example 1. Each attribute name is replaced by a URI and local name.

```
<{http://www.example.com/one}foo [cx1]>
  <{http://www.example.com/one}foo1 [cx1]
      {}bar1="_" {}bar2="_">
    text1
    <{http://www.example.com/two}foo11 [cx1]></>
  </>
  <{http://www.example.com/two}foo2 [cx1]
      {http://www.example.com/one}bar1="_"
      {http://www.example.com/one}bar2="_"
      {http://www.example.com/two}bar1="_"
      {http://www.example.com/two}bar2="_"
      {}bar1="_" {}bar2="_">
    text2
    <{http://www.example.com/one}foo21 [cx1]></>
    text3
    <{http://www.example.com/one}foo22 [cx1]></>
  </>
  <{http://www.example.com/one}foo3 [cx1]
      {http://www.example.com/two}bar1="_"
      {http://www.example.com/two}bar2="_">
    <{http://www.example.com/one}foo31 [cx1]></>
  </>
</>
```

6 Syntax

6.1 General

NVDL has the full syntax and the simple syntax. A correct NVDL script is always required to be in the full syntax. Meanwhile, the simple syntax is purely an internal artifact for specifying the semantics of NVDL. NVDL scripts in the full syntax may be transformed to NVDL scripts in the simple syntax.

6.2 Full syntax

An NVDL script in the full syntax shall be an XML document valid against the RELAX NG schema shown in Annex A.

NOTE Most elements consisting NVDL scripts in full syntax belong to the namespace "http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0". The elements in this namespace have local names as below:

— rules

— mode

— namespace

— anyNamespace

— cancel

— validate

— allow

— reject

— schema

— message

— attach

— unwrap

— option

— context

6.3 Simple syntax

The simple syntax is a subset of the full syntax. An NVDL script in the simple syntax shall be an XML document valid against the RELAX NG schema shown in Annex B.

NOTE Most elements consisting NVDL scripts in simple syntax belong to the namespace "http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0". The elements in this namespace have local names as below:

— rules

— mode

— namespace

— anyNamespace

— validate

— schema

— message

— attach

— unwrap

— option

— context

6.4 Simplification

6.4.1 General

The full syntax is transformed into the simple syntax by applying the following transformation rules in order. The effect shall be as if each transformation rule was applied to all elements in the schema before the next transformation rule is applied. A transformation rule may also specify constraints that shall be satisfied by a correct schema. The transformation rules are applied at the data model level. Before the transformations are applied, the schema is parsed into an element in the data model.

6.4.2 Annotations

Qualified attributes are removed, unless they belong to descendant elements of schema elements or they are xml:lang attributes of message elements. Elements not belonging to the namespace "http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0" are removed unless they are subordinate to schema elements.

NOTE It is safe to remove xml:base attributes at this stage because xml:base attributes are used in determining the [base URI] of an element information item, which is in turn used to construct the base URI of the context of an element. Thus, after a document has been parsed into an element in the data model, xml:base attributes can be discarded.

6.4.3 Whitespace

For each element other than message and schema, each child that is a string containing only whitespace characters is removed.

Leading and trailing whitespace characters are removed from the value of each startMode, useMode, match, mustSupport, and schemaType attribute and removed from the value of each name attribute of mode.

6.4.4 schema attribute

The value of each schema attribute is transformed by escaping disallowed characters as specified in Section 5.4 of W3C XLink.

6.4.5 name attribute of option element

The value of the name attribute on an option element is transformed by escaping disallowed characters as specified in Section 5.4 of W3C XLink.

6.4.6 message attribute

The message attribute on a validate, allow, reject, attach or unwrap element is transformed into a message child element.

6.4.7 mustSupport attribute

If an option element does not have the mustSupport attribute, mustSupport="false" is added.

6.4.8 schemaType attribute of rules elements

If the top-level rules element has the attribute schemaType, this attribute is copied to those validate elements which have neither a schemaType attribute nor a schema element. Then, the attribute schemaType of the top-level rules element is removed.

6.4.9 rules without mode children

The top-level rules element is transformed so that it has mode elements as children. If its child elements are namespace or anyNamespace elements, they are wrapped in a mode element. Then, a name attribute and startMode attribute are added to the mode element and rules element, respectively. These attributes shall have the same value, which shall be different from any other mode name.

6.4.10 mode elements in validate, allow, reject, attach or unwrap elements

In this transformation rule, mode elements in validate, allow, reject, attach or unwrap elements are removed.

If a mode element occurs as a child of a validate, allow, reject, attach or unwrap element that is in turn a child (namespace or anyNamespace) element of a child (mode) element of the top-level rules element, this mode element is moved as the youngest child of the rules element. Then, a name attribute is added to the mode element, and a useMode attribute is added to the validate, allow, reject, attach or unwrap element. These attributes shall have the same value, which shall be different from any other mode name. This transformation rule is applied repeatedly until there are no mode elements in validate, allow, reject, attach or unwrap elements.

6.4.11 match attribute of namespace or anyNamespace elements

If a namespace or anyNamespace element does not specify a match attribute, match="elements" is added.

If a namespace or anyNamespace element specifies a string containing tokens "elements" as well as "attributes", this element is replaced by a sequence of two copies of it. The match attribute of the first copy is transformed into match="elements", while that of the second is transformed into match="attributes".

6.4.12 mode inclusion

In this transformation rule, mode elements are transformed so that they do not have child mode elements.

Define the lowest-level modes as mode elements without child mode elements.

For each lowest-level mode *md*, the following transformations are applied in sequence.

— *md* shall not have any cancel elements as descendants.

— Each child anyNamespace element of *md* is removed if some sibling anyNamespace element of *md* specifies the same value for the match attribute and has a cancel element.

— Each child namespace element of *md* is removed if some sibling namespace element of *md* specifies the same value for the match attribute, specifies the same value for the ns attribute, and has a cancel element.

— *md* is replaced by its children

This transformation rule is applied repeatedly until mode elements do not have child mode elements.

NOTE Editor's note: Perhaps, this feature is least mature in NVDL. Comments are very welcome.

6.4.13 collision in mode

The match attributes of two sibling anyNamespace elements shall not be identical.

Two sibling namespace elements shall not compete with each other.

We define competition formally. Let w_1 and ns_1 (resp. w_2 and ns_2) be the wildcard and URI specified by the first (resp. second) namespace element. If the value of the wildcard attribute is "", we assume that some character not appearing in ns_1 or ns_2 is specified as a wildcard. A pair (ns_1, w_1) and another pair (ns_2, w_2) compete if and only if:

— Case 1: both ns_1 and ns_2 are empty,

— Case 2: one string is empty and the other consists of a single wildcard (i.e., either (1) ns_1="" and ns_2="w_2" or (2) ns_2="" and ns_1="w_1"),

— Case 3: both strings begin with the same non-wildcard charcter and their tails compete (where the tail of a string is obtained by stripping the first character), or

— Case 4: either (1) ns_1 begins with w_1 or (2) ns_2 begins with w_2, and either (i) ns_1 and the tail of ns_2 compete or (ii) ns_2 and the tail of ns_1 compete

NOTE This restriction ensures that only one namespace or anyNamespace in a mode is triggered by a namespace URI, so that at most one unwrap or attach is invoked.

6.4.14 default anyNamespace

For each mode element, default anyNamespace elements are added.

If a mode element does not have an anyNamespace element with match="elements",

```
<anyNamespace match="elements"><reject/></anyNamespace>
```

is added.

If a mode element does not have an anyNamespace element with match="attributes",

```
<anyNamespace match="attributes"><attach/></anyNamespace>
```

is added.

6.4.15 allow and reject

Each allow element is replaced by a validate element. Its content shall be the content of the allow element followed by

```
<schema><allow xmlns="http://purl.oclc.org/dsdl/nvdl/ns/allow/1.0"/></schema>,
```

where allow in the namespace "http://purl.oclc.org/dsdl/nvdl/ns/allow/1.0" is a schema that allows any document.

Each reject element is replaced by a validate element. Its content shall be the content of the allow element followed by

```
<schema><reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0"/></schema>,
```

where reject in the namespace "http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0" is a schema that allows no documents.

NOTE The motivation for these predefined schemas is merely the ease of specifiying the semantics.

6.4.16 useMode attribute

For each validate, attach or unwrap element without a useMode attribute, a useMode attribute is added. The value shall be copied from the name attribute of the parent (mode) element of the parent (namespace or anyNamespace) element of the validate, attach or unwrap element.

7 Primitive operations

7.1 General

This clause introduces primitive operations for the NVDL data model. These operations are used for defining the NVDL semantics in Clause 8.

7.2 Creating element sections and attribute sections

This process decomposes an XML instance into element sections and attribute sections. An attribute section is a non-empty set of attributes belonging to the same namespace. An element section is an element such that it and its descendant elements belong to the same namespace.

Decomposition is achieved using the following rules:

First, an attribute slot node and attribute section are created for each (*e*, *n*) pair, where *e* is an element and *n* is the namespace URI of some attribute of *e*. This attribute section contains those attributes of *e* which belong to *n*, and the attribute slot node is the place holder for the attribute section. The attribute set of *e* shall be modified by introducing the attribute slot node and removing the attributes in the attribute section.

Second, an element slot node and element section are created for each element (say *e*), if *e* and its parent element (if any) belong to different namespaces. The element section is represented by *e*, and the element slot node is the place holder for the element section. Then, the child sequences of the parent element of *e* is modified by replacing *e* with the element slot node.

EXAMPLE1 Five element sections are generated from the data model in Example 1 in 5.2. The first element section represents the document. It has two element slot nodes: esn1 and esn2.

```
<{http://www.example.com/one}foo [cx1]>
  <{http://www.example.com/one}foo1 [cx1]>
    text1
    $esn1
  </>
  $esn2
  <{http://www.example.com/one}foo3 [cx1]>
    <{http://www.example.com/one}foo31 [cx1]></>
  </>
</>
```

The second element section corresponds to esn1. It has no element slot nodes.

```
<{http://www.example.com/two}foo11 [cx1]></>
```

The third element section corresponds to esn2. It has two element slot nodes: esn3 and esn4.

```
<{http://www.example.com/two}foo2 [cx1]>
  text2
  $esn3
  text3
  $esn4
</>
```

The fourth element section corresponds to esn3. It has no element slot nodes.

```
<{http://www.example.com/one}foo21 [cx1]></>
```

The fifth element section corresponds to esn4. It has no element slot nodes.

```
<{http://www.example.com/one}foo22 [cx1]></>
```

EXAMPLE2 Five element sections and five attribute sections are generated from the data model in Example 2 in 5.2. The first element section represents the document. It has two element slot nodes: esn1 and esn2, and two attribute slot nodes: asn1 and asn2.

```
<{http://www.example.com/one}foo [cx1]>
  <{http://www.example.com/one}foo1 [cx1]
      $asn1>
    text1
    $esn1
  </>
  $esn2
  <{http://www.example.com/one}foo3 [cx1]
      $asn2>
    <{http://www.example.com/one}foo31 [cx1]></>
  </>
</>
```

The first attribute section corresponds to asn1. It has two attributes of the namespace "".

```
{{}bar1="_", {}bar2="_"}
```

The second attribute section corresponds to asn2. It has two attributes of the namespace "http://www.example.com/two".

```
{{http://www.example.com/two}bar1="_", {http://www.example.com/two}bar2="_"}
```

The second element section corresponds to esn1. It has no element or attribute slot nodes.

```
<{http://www.example.com/two}foo11 [cx1]></>
```

The third element section corresponds to esn2. It has two element slot nodes: esn3 and esn4, and three attribute slot nodes: asn3, asn4, and asn5.

```
<{http://www.example.com/two}foo2 [cx1] $asn3 $asn4 $asn5>
  text2
  $esn3
  text3
  $esn4
</>
```

The third attribute section corresponds to asn3. It has two attributes of the namespace "http://www.example.com/one".

```
{{http://www.example.com/one}bar1="_", {http://www.example.com/one}bar2="_"}
```

The fourth attribute section corresponds to asn4. It has two attributes of the namespace "http://www.example.com/two".

```
{{http://www.example.com/two}bar1="_", {http://www.example.com/two}bar2="_"}
```

The fifth attribute section corresponds to asn5. It has two attributes of the namespace "".

```
{{}bar1="_", {}bar2="_"}
```

The fourth element section corresponds to esn3. It has no element or attribute slot nodes.

```
<{http://www.example.com/one}foo21 [cx1]></>
```

The fifth element section corresponds to esn4. It has no element or attribute slot nodes.

```
<{http://www.example.com/one}foo22 [cx1]></>
```

7.3 Attaching attribute sections to elements

An attribute section can be attached to an element (say *e*). This is done by replacing the place-holder attribute slot node for this attribute section with those attributes in it. If this attribute slot node does not belong to *e*, NVDL dispatchers should issue warning message.

NOTE When an attribute section is attached to an element in 8.4, it is not guaranteed that the attribute slot node for the attribute section belongs to this element.

EXAMPLE If the third attribute section is attached to the third element section in EXAMPLE 2 in 7.2, we have:

```
<{http://www.example.com/two}foo2  [cx1]
    {http://www.example.com/one}bar1="_"
    {http://www.example.com/one}bar2="_" $asn4 $asn5>
  text2
  $esn3
  text3
  $esn4
</>
```

7.4 Attaching element sequences to elements

An element (say *e*) is said to be reachable from an element slot node when

— this element slot node is the place holder for *e*, or

— this element slot node is the place holder for some element *e′* and *e* is reachable from another element slot node that belongs to *e′*.

A sequence of elements (say e_1, e_2, ..., e_m) can be attached to an element (say *e*) if e_1, e_2, ..., and e_m are all reachable from some element slot node in *e*. This is done by replacing this element slot node with e_1, e_2, ..., e_m.

NOTE When a sequence of elements is attached to an element in 8.4, it is guaranteed that every element in the sequence is reachable from some element slot node belonging to this element.

EXAMPLE1 If the fourth element section is attached to the third element section in EXAMPLE 1 in 7.2, we have:

```
<{http://www.example.com/two}foo2 [cx1]>
  text2
  <{http://www.example.com/one}foo21 [cx1]></>
  text3
  $esn4
</>
```

EXAMPLE2 If the fourth element section followed by the fifth element section is attached to the first element section in EXAMPLE 1 in 7.2, we have:

```
<{http://www.example.com/one}foo [cx1]>
  <{http://www.example.com/one}foo1 [cx1]
      $asn1>
    text1
    $esn1
  </>
  <{http://www.example.com/one}foo21 [cx1]></>
  <{http://www.example.com/one}foo22 [cx1]></>
  <{http://www.example.com/one}foo3 [cx1]
      $asn2>
    <{http://www.example.com/one}foo31 [cx1]></>
```

```
    </>
</>
```

7.5 Removing slot nodes from elements

An element in the NVDL data model can be converted so that there are no element or attribute slot nodes. This is done as follows. All attribute or element slot nodes of this element or its descendant elements are discarded. If multiple text nodes become adjacent, they are concatenated to form a single text node.

NOTE The RELAX NG data model, as well as the NVDL data model, allows the resulting element.

EXAMPLE From the result in EXAMPLE 2 in 7.4, we obtain:

```
<{http://www.example.com/one}foo [cx1]>
  <{http://www.example.com/one}foo1 [cx1]>
    text1
  </>
  <{http://www.example.com/one}foo21 [cx1]></>
  <{http://www.example.com/one}foo22 [cx1]></>
  <{http://www.example.com/one}foo3 [cx1]>
    <{http://www.example.com/one}foo31 [cx1]></>
  </>
</>
```

7.6 Converting attribute sections to empty elements

An attribute section can be converted to an empty element. This is done as follows. An empty element is created so that the namespace URI is "", the local name is dummy, the child sequence is empty, and the attribute set is the attribute section.

NOTE 1 As in NRL, should we also perform a corresponding transformation on schemas? But how?

NOTE 2 What should be the namespace URI and local name of this empty element?

7.7 Invoking non-NVDL validators for elements

The NVDL dispatcher passes elements in the NVDL data model to validators. There are no element or attribute slot nodes in such elements. Validators are required to deal with such elements.

NOTE Since elements in the NVDL data model do not have namespace declarations, prefixes, processing instructions, comments, or document type declarations, validators cannot recieve them.

8 Semantics

8.1 General

The semantics of NVDL script are defined using a reference model for dispatching. This reference model has five stages:

— Stage 1: Constructing interpretations

— Stage 2: Combining sections

— Stage 3: Filtering of the combined sections

— Stage 4: Creating validation candidates

— Stage 5: Validation

NOTE NVDL dispatchers do not have to follow this reference model, as long as the invocation of validators is the same. In particular, NVDL dispatchers may use stream APIs without creating documents in the main memory.

8.2 Preliminaries

In prepration, we define some predicates.

match(*nsValue*, *wc*, *uri*) holds if and only if a string *uri* is obtained from *nsValue* by replacing every occurrence of the character *wc* in *nsValue* with some string. Different occurrences need not be replaced with the same string.

match(*nsValue*, *uri*) holds if and only if *uri* is equal to *nsValue*.

matchElemSec(*nsOrAny*, *uri*) holds when:

— Case 1: *nsOrAny* is a namespace element having ns="*nsValue*", wildCard="*wc*", and match="elements", and match(*nsValue*, *wc*,*uri*) holds, or

— Case 2: *nsOrAny* is an anyNamespace element having match="elements", and matchElemSec(*nsOrAny'* , *uri*) does not hold for any sibling namespace element *nsOrAny'* of *nsOrAny*.

matchAttSec(*nsOrAny*, *uri*) holds when:

— Case 1: *nsOrAny* is a namespace element having ns="*nsValue*", wildCard="*wc*", and match="attributes", and match(*nsValue*, *wc*, *uri*), or

— Case 2: *nsOrAny* is an anyNamespace element having match="attributes", and matchAttSec(*nsOrAny'*, *uri*) does not hold for any sibling namespace element *nsOrAny'* of *nsOrAny*.

matchPathExp(*pathExp*, *p*) holds when a path *p* matches *pathExp* as defined in XSLT.

applies(*action*, *p*, *mode*) holds when:

— Case 1: *mode* is specified by the useMode attribute of *action*, and for any subordinate context element of *action*, matchPathExp(*pathExp*, *p*) does not hold, where *pathExp* is the value of the path attribute of the context, or

— Case 2: for some subordinate context element of *action*, matchPathExp(*pathExp*, *p*) holds where *pathExp* is the value of the path attribute of this context, and *mode* is specified by the useMode attribute of this context, and no other subordinate context element of *action* specifies a more specific path attribute such that matchPathExp(*pathExp'*, *p*).

NOTE 1 Needs a definition of "specific".

NOTE 2 Should we syntactically prohibit equally-specific path expressions? or should we raise runtime errors?

elemTrans(*mode*, *uri*, *p*, *mode'*, *a*) holds when, for some namespace or anyNamespace element *nsOrAny* in *mode*, *a* is a subordinate action of *nsOrAny*, matchElemSec(*nsOrAny*, *uri*), and applies(*a*, *p*, *mode'*).

attTrans(*mode*, *uri*, *a*) holds when, for some namespace or anyNamespace element *nsOrAny* in *mode*, matchAttSec(*nsOrAny*, *uri*), *a* is a subordinate action of *nsOrAny*,

8.3 Stage 1: Constructing interpretations

This stage constructs interpretations of an instance against an NDVL script. An interpretation associates each section with an action and mode.

An interpretation *I* is a function that maps a section (*s*) in the given instance to a pair of an action (say $I_A(s)$) and a mode (say $I_M(s)$) such that:

— If an element section *s* is the root element section, then elemTrans(m_{rt}, namespace(*s*), epsilon, $I_M(s)$, $I_A(s)$), where m_{rt} is the root mode of the NVDL script and epsilon is a path comprising no NCNames.

— If an element section *s* is the parent section of *s'* then elemTrans($I_M(s)$, namespace, path(*s'*), $I_M(s')$, $I_A(s')$)

— If an element section *s* is the parent section of an attribute section *a* then attTrans($I_M(s)$, namespace(a), $I_A(a)$).

NOTE Should we allow attribute sections to have modes?

More than one interpretation may be constructed for a single instance when a single namespace or anyNamespace element has multiple actions.

8.4 Stage 2: Combining sections

For each interpretation of an instance, this stage combines sections by executing attach and unwrap actions.

Two auxiliary functions are required:

First, for each interpretation *I*, attBubble[*I*] is a function that maps an attribute section in the given instance to an attribute set such that:

— attBubble[*I*](*s*) = *s* when $I_A(s)$ is an attach action

— attBubble[*I*](*s*) = {} otherwise

Second, for each interpretation *I*, elemBubble[*I*] is a function that maps an element section in the given instance to an element sequence such that:

— elemBubble[*I*](*s*) = () when $I_A(s)$ is a validate action

— elemBubble[*I*](*s*) = (*s* <- elemBubble[*I*](s_1) <- elemBubble[*I*](s_2) <- ... <- elemBubble[*I*](s_m) <- attBubble[*I*](s'_1) <- attBubble[*I*](s'_2) <- ... <- attBubble[*I*](s'_n)) where (1) s_1, s_2, ..., s_m are child element sections of *s*, (2) s'_1, s'_2, ..., s'_n are child attribute sections of *s*, when $I_A(s)$ is an attach action

NOTE The number (m) of element sections and that (n) of attribute sections may be different.

— elemBubble[*I*](*s*) = elemBubble[*I*](s_1)+elemBubble[*I*](s_2)+...+elemBubble[*I*](s_m), where s_1, s_2, ..., s_m are child element sections of *s*, when $I_A(s)$ is an unwrap action.

To define how sections are combined, for each interpretation *I*, we introduce a partial function syn[*I*] that maps an element section *s* in the given instance to an element when $I_A(s)$ is a validate action. This is a partial function since it is defined only when the associated action is validate.

— syn[*I*](*s*) = *s* <- elemBubble[*I*](s_1) <- elemBubble[*I*](s_2) <- ... <- elemBubble[*I*](s_m) where s_1, s_2, ..., s_m are child element sections of *s*, when *s* is an element section and $I_A(s)$ is a validate action.

8.5 Stage 3: Filtering of the combined sections

For each interpretation of an instance, this stage constructs a set ValElem of (*v*, *e*) and another set ValAtt of (*v*, *s*), where *v* is a validate element in the NVDL script, *e* is an element, and *s* is an attribute section.

Let X = {(*s*, I_A(*s*), syn[*I*](*s*)) | *s* is an element section and *I* is an interpretation}. Then:

— ValElem = { (*a*, *e*) | (*s*, *a*, *e*) in X, *a* is a validate action, if (*s*, *a*, *e'*) is in X then |*e*| >= |*e'*|}.

NOTE |*e*| >= |*e'*| is meant to choose the "biggest" validation candidate.

— ValAtt = {(*a*, *s*) | *s* is an attribute section, *a* is a validate action, and *a* = I_A(*s*) for some interpretation *I*}.

8.6 Stage 4: Creating validation candidates

This stage creates two sets PlanElem and PlanAtt of (*v*, *e*) from ValElem and ValAtt, respectively, where *v* is a validate action and *e* is an element.

— PlanElem = {(*v*, removeSlots(*e*)) | (*v*, *e*) is in ValElem}

NOTE How about "dummy" and "root-only" of validate elements?

— PlanAtt = {(*v*, dummyElement(*s*)) | (*v*, *s*) is in ValAtt}

8.7 Stage 5: Validation

This stage dispatches each (*v*, *e*) in the union of PlanElem and PlanAtt to a validator. The validator is chosen from the schema language that is used for writing the schema for *v*.

If *v* contains a schema element as a child, its content is used as a schema. When the content is a string and *v* has the schemaType attribute , its value shall be a MIME media type (see IETF RFC 2046) and be used for determining the schema language. When the content is an element, its namespace is used for determining the schema language.

If *v* references to a schema by the schema attribute, the schema is constructed by using its value and the schema language is determined from the schema and the schemaType attribute of *v*.

This shall be done as follows. The URI reference consists of the URI itself and an optional fragment identifier. The resource identified by the URI is retrieved. The result is a MIME entity (see IETF RFC 2045): a sequence of bytes labeled with a MIME media type (see IETF RFC 2046).

— When the media type is not available, the MIME entity shall be used as a schema. The schema language shall be determined from the schemaType attribute.

— When the media type is an XML media type (i.e.,application/xml or text/xml or*/*+xml), the MIME entity shall be parsed as an XML document in accordance with the applicable RFC (at the term of writing IETF RFC 3023). In particular, the charset parameter shall be handled as specified by the RFC. The href attribute shall not include a fragment identifier unless the registration of the media type of the resource identified by the attribute defines the interpretation of fragment identifiers for that media type. The schema language is determined from the namespace of the root element of the XML document.

NOTE 1 IETF RFC 3023 does not define the interpretation of fragment identifiers forapplication/xml,text/xml, or */*+xml.

— When the media type is not an XML media type, the MIME entity shall be used as a schema. The schema language shall be determined from the schemaType attribute.

When (*v*, *e*) is dispatched to a validator, every option specified as child element of *v* shall be passed to the validator. If the validator does not support an option such that the value of mustSupport is either 1 or true, an error shall be reported.

NOTE 2 Should this part of the standard provide a list of options? Certainly, we do not want to publish an amendment when a new schema language appears.

NOTE 3 Should we allow relative URIs as options (as in NRL)?

The result of dispatching is a success if the result of every validation is successful.

9 Conformance

A conforming NVDL dispatcher shall be able to determine for any XML document whether it is a correct NVDL script. A conforming NVDL dispatcher shall be able to create validation candidates from any instance, invoke validators for these validation candidates, and report whether these validators succeed or fail.

However, the requirements in the preceding paragraph do not apply if the NVDL script uses a schema language that the NVDL dispatcher does not support. A conforming NVDL dispatcher is not required to support any schema languages except the schemas of the namespaces "http://purl.oclc.org/dsdl/nvdl/ns/allow/1.0" and "http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0".

Annex A
(normative)

Full syntax in RELAX NG

The namespace URI http://www.w3.org/2001/XMLSchema-datatypes references to the datatypes and facets specified by W3C XML Schema Part 2.

NOTE What should be the namespace URI for NVDL schemas? At present, we use http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0. Another possibility is urn:publicid:ISO/IEC 19757-4:2004//NVDL//EN (see RFC 3151).

```
<grammar ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
 datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
 xmlns="http://relaxng.org/ns/structure/1.0"
 xmlns:nvdl="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
 xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">

 <start>
  <element name="rules">
   <interleave>
    <group>
     <optional>
      <ref name="schemaType"/>
     </optional>
     <choice>
      <zeroOrMore>
       <ref name="rule"/>
      </zeroOrMore>
      <group>
       <attribute name="startMode">
        <data type="NCName"/>
       </attribute>
       <oneOrMore>
        <ref name="mode"/>
       </oneOrMore>
      </group>
     </choice>
    </group>
    <ref name="foreign"/>
   </interleave>
  </element>
 </start>

 <define name="mode">
  <element name="mode">
   <interleave>
    <group>
     <attribute name="name">
      <data type="NCName"/>
     </attribute>
     <zeroOrMore>
      <ref name="includeMode"/>
     </zeroOrMore>
     <zeroOrMore>
      <ref name="rule"/>
     </zeroOrMore>
    </group>
    <ref name="foreign"/>
   </interleave>
  </element>
```

```
</define>

<define name="includeMode">
  <element name="mode">
    <interleave>
      <group>
        <optional>
          <attribute name="name">
            <data type="NCName"/>
          </attribute>
        </optional>
        <zeroOrMore>
          <ref name="includeMode"/>
        </zeroOrMore>
        <zeroOrMore>
          <ref name="rule"/>
        </zeroOrMore>
      </group>
      <ref name="foreign"/>
    </interleave>
  </element>
</define>

<define name="rule">
  <choice>
    <element name="namespace">
      <interleave>
        <group>
          <attribute name="ns">
            <data type="string"/>
          </attribute>
          <optional>
            <attribute name="wildcard">
              <data type="string">
                <param name="length">1</param>
              </data>
            </attribute>
          </optional>
          <ref name="ruleModel"/>
        </group>
        <ref name="foreign"/>
      </interleave>
    </element>
    <element name="anyNamespace">
      <interleave>
        <ref name="ruleModel"/>
        <ref name="foreign"/>
      </interleave>
    </element>
  </choice>
</define>

<define name="ruleModel">
  <optional>
    <attribute name="match">
      <ref name="elementsOrAttributes"/>
    </attribute>
  </optional>
  <ref name="actions"/>
</define>

<define name="elementsOrAttributes">
  <list>
    <choice>
      <group>
        <value>elements</value>
```

```
        <value>attributes</value>
      </group>
      <group>
        <value>attributes</value>
        <value>elements</value>
      </group>
      <value>elements</value>
      <value>attributes</value>
    </choice>
  </list>
</define>

<define name="actions">
  <choice>
    <ref name="cancelAction"/>
    <group>
      <zeroOrMore>
        <ref name="noResultAction"/>
      </zeroOrMore>
      <choice>
        <ref name="noResultAction"/>
        <ref name="resultAction"/>
      </choice>
      <zeroOrMore>
        <ref name="noResultAction"/>
      </zeroOrMore>
    </group>
  </choice>
</define>

<define name="cancelAction">
  <element name="cancel">
    <ref name="foreign"/>
  </element>
</define>

<define name="noResultAction">
  <choice>
    <element name="validate">
      <interleave>
        <group>
          <optional>
            <ref name="schemaType"/>
          </optional>
          <zeroOrMore>
            <choice>
              <ref name="message"/>
              <ref name="option"/>
            </choice>
          </zeroOrMore>
          <ref name="schema"/>
          <ref name="modeUsage"/>
        </group>
        <ref name="foreign"/>
      </interleave>
    </element>
    <element>
      <choice>
        <name>allow</name>
        <name>reject</name>
      </choice>
      <interleave>
        <group>
          <zeroOrMore>
            <ref name="message"/>
          </zeroOrMore>
```

```
        <ref name="modeUsage"/>
       </group>
       <ref name="foreign"/>
      </interleave>
    </element>
  </choice>
</define>

<define name="schema">
  <choice>
    <attribute name="schema">
      <data type="anyURI"/>
    </attribute>
    <element name="schema">
      <choice>
        <text/>
        <ref name="foreignElement"/>
      </choice>
      <zeroOrMore>
        <ref name="foreignAttribute"/>
      </zeroOrMore>
    </element>
  </choice>
</define>

<define name="message">
  <choice>
    <attribute name="message"/>
    <element name="message">
      <group>
        <text/>
        <zeroOrMore>
          <ref name="xmlAttribute"/>
        </zeroOrMore>
      </group>
      <zeroOrMore>
        <ref name="nonXMLForeignAttribute"/>
      </zeroOrMore>
    </element>
  </choice>
</define>

<define name="resultAction">
  <element>
    <choice>
      <name>attach</name>
      <name>unwrap</name>
    </choice>
    <interleave>
      <group>
        <zeroOrMore>
          <ref name="message"/>
        </zeroOrMore>
        <ref name="modeUsage"/>
      </group>
      <ref name="foreign"/>
    </interleave>
  </element>
</define>

<define name="option">
  <element name="option">
    <interleave>
      <group>
        <attribute name="name">
          <data type="anyURI"/>
```

```
      </attribute>
      <optional>
        <attribute name="arg"/>
      </optional>
      <optional>
        <attribute name="mustSupport">
          <data type="boolean"/>
        </attribute>
      </optional>
    </group>
    <ref name="foreign"/>
  </interleave>
 </element>
</define>

<define name="modeUsage">
 <optional>
  <choice>
   <attribute name="useMode">
    <data type="NCName"/>
   </attribute>
   <ref name="nestedMode"/>
  </choice>
 </optional>
 <zeroOrMore>
  <element name="context">
   <interleave>
    <group>
     <attribute name="path">
      <ref name="path"/>
     </attribute>
     <optional>
      <choice>
       <attribute name="useMode">
        <data type="NCName"/>
       </attribute>
       <ref name="nestedMode"/>
      </choice>
     </optional>
    </group>
    <ref name="foreign"/>
   </interleave>
  </element>
 </zeroOrMore>
</define>

<define name="nestedMode">
 <element name="mode">
  <interleave>
   <group>
    <zeroOrMore>
     <ref name="includeMode"/>
    </zeroOrMore>
    <zeroOrMore>
     <ref name="rule"/>
    </zeroOrMore>
   </group>
   <ref name="foreign"/>
  </interleave>
 </element>
</define>

<define name="schemaType">
 <attribute name="schemaType">
  <ref name="mediaType"/>
 </attribute>
```

```
</define>

<define name="mediaType">
  <data type="string">
    <param name="pattern">\s*[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*\\[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*\s*</param>
  </data>
</define>

<define name="path">
  <data type="string">
    <param name="pattern">\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*(\|\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*)*</param>
  </data>
</define>

<define name="foreignElement">
  <element>
    <anyName>
      <except>
        <nsName/>
      </except>
    </anyName>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <mixed>
      <zeroOrMore>
        <ref name="anyElement"/>
      </zeroOrMore>
    </mixed>
  </element>
</define>

<define name="anyElement">
  <element>
    <anyName/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <mixed>
      <zeroOrMore>
        <ref name="anyElement"/>
      </zeroOrMore>
    </mixed>
  </element>
</define>

<define name="foreignAttribute">
  <attribute>
    <anyName>
      <except>
        <nsName/>
        <nsName ns=""/>
      </except>
    </anyName>
  </attribute>
</define>

<define name="nonXMLForeignAttribute">
  <attribute>
    <anyName>
      <except>
        <nsName ns="http://www.w3.org/XML/1998/namespace"/>
```

```
        <nsName/>
        <nsName ns=""/>
       </except>
     </anyName>
    </attribute>
   </define>

   <define name="xmlAttribute">
    <choice>
     <attribute name="xml:lang"/>
     <attribute name="xml:space">
      <choice>
       <value>default</value>
       <value>preserve</value>
      </choice>
     </attribute>
     <attribute name="xml:base">
      <ref name="anyURI"/>
     </attribute>
    </choice>
   </define>

   <define name="foreign">
    <zeroOrMore>
     <ref name="foreignAttribute"/>
    </zeroOrMore>
    <zeroOrMore>
     <ref name="foreignElement"/>
    </zeroOrMore>
   </define>

  </grammar>
```

Annex B
(normative)

Simple syntax in RELAX NG

```
<grammar ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
 datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
 xmlns="http://relaxng.org/ns/structure/1.0"
 xmlns:nvdl="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
 xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">

 <start>
  <element name="rules">
   <attribute name="startMode">
    <data type="NCName"/>
   </attribute>
   <oneOrMore>
    <ref name="mode"/>
   </oneOrMore>
  </element>
 </start>

 <define name="mode">
  <element name="mode">
   <attribute name="name">
    <data type="NCName"/>
   </attribute>
   <zeroOrMore>
    <ref name="rule"/>
   </zeroOrMore>
  </element>
 </define>

 <define name="rule">
  <choice>
   <element name="namespace">
    <attribute name="ns">
     <data type="string"/>
    </attribute>
    <optional>
     <attribute name="wildcard">
      <data type="string">
       <param name="length">1</param>
      </data>
     </attribute>
    </optional>
    <ref name="ruleModel"/>
   </element>
   <element name="anyNamespace">
    <ref name="ruleModel"/>
   </element>
  </choice>
 </define>

 <define name="ruleModel">
  <attribute name="match">
   <ref name="elementsOrAttributes"/>
  </attribute>
  <ref name="actions"/>
 </define>
```

```
<define name="elementsOrAttributes">
  <choice>
    <value>elements</value>
    <value>attributes</value>
  </choice>
</define>

<define name="actions">
  <zeroOrMore>
    <ref name="noResultAction"/>
  </zeroOrMore>
  <choice>
    <ref name="noResultAction"/>
    <ref name="resultAction"/>
  </choice>
  <zeroOrMore>
    <ref name="noResultAction"/>
  </zeroOrMore>
</define>

<define name="noResultAction">
  <element name="validate">
    <optional>
      <ref name="schemaType"/>
    </optional>
    <zeroOrMore>
      <choice>
        <ref name="message"/>
        <ref name="option"/>
      </choice>
    </zeroOrMore>
    <ref name="schema"/>
    <ref name="modeUsage"/>
  </element>
</define>

<define name="schema">
  <choice>
    <attribute name="schema">
      <data type="anyURI"/>
    </attribute>
    <element name="schema">
      <choice>
        <text/>
        <ref name="foreignElement"/>
      </choice>
    </element>
  </choice>
</define>

<define name="message">
  <element name="message">
    <interleave>
      <text/>
      <attribute name="xml:lang"/>
    </interleave>
  </element>
</define>

<define name="resultAction">
  <element>
    <choice>
      <name>attach</name>
      <name>unwrap</name>
    </choice>
    <zeroOrMore>
```

```
      <ref name="message"/>
    </zeroOrMore>
    <ref name="modeUsage"/>
  </element>
</define>

<define name="option">
  <element name="option">
    <attribute name="name">
      <data type="anyURI"/>
    </attribute>
    <optional>
      <attribute name="arg"/>
    </optional>
    <attribute name="mustSupport">
      <data type="boolean"/>
    </attribute>
  </element>
</define>

<define name="modeUsage">
  <attribute name="useMode">
    <data type="NCName"/>
  </attribute>
  <zeroOrMore>
    <element name="context">
      <attribute name="path">
        <ref name="path"/>
      </attribute>
      <attribute name="useMode">
        <data type="NCName"/>
      </attribute>
    </element>
  </zeroOrMore>
</define>

<define name="schemaType">
  <attribute name="schemaType">
    <ref name="mediaType"/>
  </attribute>
</define>

<define name="mediaType">
  <data type="string">
    <param name="pattern">\s*[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*\\[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*\s*</param>
  </data>
</define>

<define name="path">
  <data type="string">
    <param name="pattern">\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*(\|\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*)*</param>
  </data>
</define>

<define name="foreignElement">
  <element>
    <anyName>
      <except>
        <nsName/>
      </except>
    </anyName>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
```

```
      <mixed>
        <zeroOrMore>
          <ref name="anyElement"/>
        </zeroOrMore>
      </mixed>
    </element>
  </define>

  <define name="anyElement">
    <element>
      <anyName/>
      <zeroOrMore>
        <attribute>
          <anyName/>
        </attribute>
      </zeroOrMore>
      <mixed>
        <zeroOrMore>
          <ref name="anyElement"/>
        </zeroOrMore>
      </mixed>
    </element>
  </define>

</grammar>
```

Annex C
(informative)

An NVDL script and RELAX NG schema for the full syntax

C.1 General

As specified in 6.4.2, some attributes and elements are removed by simplification. Since the RELAX NG schema in6.2 explicitly permits these attributes and elements, it is not easy to understand. Here we provide a RELAX NG schema that does not allow these attributes and elements, and further provide an NVDL script for allowing them.

C.2 RELAX NG schema

```
<grammar ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
 datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
 xmlns="http://relaxng.org/ns/structure/1.0"
 xmlns:nvdl="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
 xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">

 <start>
  <element name="rules">
   <optional>
    <ref name="schemaType"/>
   </optional>
   <choice>
    <zeroOrMore>
     <ref name="rule"/>
    </zeroOrMore>
    <group>
     <attribute name="startMode">
      <data type="NCName"/>
     </attribute>
     <oneOrMore>
      <ref name="mode"/>
     </oneOrMore>
    </group>
   </choice>
  </element>
 </start>

 <define name="mode">
  <element name="mode">
   <attribute name="name">
    <data type="NCName"/>
   </attribute>
   <zeroOrMore>
    <ref name="includeMode"/>
   </zeroOrMore>
   <zeroOrMore>
    <ref name="rule"/>
   </zeroOrMore>
  </element>
 </define>

 <define name="includeMode">
  <element name="mode">
   <optional>
    <attribute name="name">
     <data type="NCName"/>
    </attribute>
```

```
    </optional>
    <zeroOrMore>
      <ref name="includeMode"/>
    </zeroOrMore>
    <zeroOrMore>
      <ref name="rule"/>
    </zeroOrMore>
  </element>
</define>

<define name="rule">
  <choice>
    <element name="namespace">
      <attribute name="ns">
        <data type="string"/>
      </attribute>
      <optional>
        <attribute name="wildcard">
          <data type="string">
            <param name="length">1</param>
          </data>
        </attribute>
      </optional>
      <ref name="ruleModel"/>
    </element>
    <element name="anyNamespace">
      <ref name="ruleModel"/>
    </element>
  </choice>
</define>

<define name="ruleModel">
  <optional>
    <attribute name="match">
      <ref name="elementsOrAttributes"/>
    </attribute>
  </optional>
  <ref name="actions"/>
</define>

<define name="elementsOrAttributes">
  <list>
    <choice>
      <group>
        <value>elements</value>
        <value>attributes</value>
      </group>
      <group>
        <value>attributes</value>
        <value>elements</value>
      </group>
      <value>elements</value>
      <value>attributes</value>
    </choice>
  </list>
</define>

<define name="actions">
  <choice>
    <ref name="cancelAction"/>
    <group>
      <zeroOrMore>
        <ref name="noResultAction"/>
      </zeroOrMore>
      <choice>
        <ref name="noResultAction"/>
```

```
        <ref name="resultAction"/>
      </choice>
      <zeroOrMore>
        <ref name="noResultAction"/>
      </zeroOrMore>
    </group>
  </choice>
</define>

<define name="cancelAction">
  <element name="cancel">
    <empty/>
  </element>
</define>

<define name="noResultAction">
  <choice>
    <element name="validate">
      <optional>
        <ref name="schemaType"/>
      </optional>
      <zeroOrMore>
        <choice>
          <ref name="message"/>
          <ref name="option"/>
        </choice>
      </zeroOrMore>
      <ref name="schema"/>
      <ref name="modeUsage"/>
    </element>
    <element>
      <choice>
        <name>allow</name>
        <name>reject</name>
      </choice>
      <zeroOrMore>
        <ref name="message"/>
      </zeroOrMore>
      <ref name="modeUsage"/>
    </element>
  </choice>
</define>

<define name="schema">
  <choice>
    <attribute name="schema">
      <data type="anyURI"/>
    </attribute>
    <element name="schema">
      <choice>
        <text/>
        <ref name="foreignElement"/>
      </choice>
    </element>
  </choice>
</define>

<define name="message">
  <choice>
    <attribute name="message"/>
    <element name="message">
      <interleave>
        <text/>
        <attribute name="xml:lang"/>
      </interleave>
    </element>
```

```
  </choice>
</define>

<define name="resultAction">
  <element>
    <choice>
      <name>attach</name>
      <name>unwrap</name>
    </choice>
    <zeroOrMore>
      <ref name="message"/>
    </zeroOrMore>
    <ref name="modeUsage"/>
  </element>
</define>

<define name="option">
  <element name="option">
    <attribute name="name">
      <data type="anyURI"/>
    </attribute>
    <optional>
      <attribute name="arg"/>
    </optional>
    <optional>
      <attribute name="mustSupport">
        <data type="boolean"/>
      </attribute>
    </optional>
  </element>
</define>

<define name="modeUsage">
  <optional>
    <choice>
      <attribute name="useMode">
        <data type="NCName"/>
      </attribute>
      <ref name="nestedMode"/>
    </choice>
  </optional>
  <zeroOrMore>
    <element name="context">
      <attribute name="path">
        <ref name="path"/>
      </attribute>
      <optional>
        <choice>
          <attribute name="useMode">
            <data type="NCName"/>
          </attribute>
          <ref name="nestedMode"/>
        </choice>
      </optional>
    </element>
  </zeroOrMore>
</define>

<define name="nestedMode">
  <element name="mode">
    <zeroOrMore>
      <ref name="includeMode"/>
    </zeroOrMore>
    <zeroOrMore>
      <ref name="rule"/>
    </zeroOrMore>
```

```
  </element>
</define>

<define name="schemaType">
  <attribute name="schemaType">
    <ref name="mediaType"/>
  </attribute>
</define>

<define name="mediaType">
  <data type="string">
    <param name="pattern">\s*[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*\\[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*\s*</param>
  </data>
</define>

<define name="path">
  <data type="string">
    <param name="pattern">\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*(\|\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*)*</param>
  </data>
</define>

<define name="foreignElement">
  <element>
    <anyName>
      <except>
        <nsName/>
      </except>
    </anyName>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <mixed>
      <zeroOrMore>
        <ref name="anyElement"/>
      </zeroOrMore>
    </mixed>
  </element>
</define>

<define name="anyElement">
  <element>
    <anyName/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <mixed>
      <zeroOrMore>
        <ref name="anyElement"/>
      </zeroOrMore>
    </mixed>
  </element>
</define>

<define name="foreignAttribute">
  <attribute>
    <anyName>
      <except>
        <nsName/>
        <nsName ns=""/>
      </except>
    </anyName>
  </attribute>
```

```
  </define>

  <define name="foreign">
    <zeroOrMore>
      <ref name="foreignAttribute"/>
    </zeroOrMore>
    <zeroOrMore>
      <ref name="foreignElement"/>
    </zeroOrMore>
  </define>

</grammar>
```

C.3 NVDL script

```
<rules startMode="root"
  xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">

  <mode name="root">
    <namespace ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">
      <validate schema="nvdl.rng">
        <mode>
          <namespace ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
            match="attributes">
            <reject/>
          </namespace>
          <namespace ns=""
            match="attributes">
            <attach/>
          </namespace>
          <anyNamespace match="elements attributes">
            <allow>
              <mode>
                <anyNamespace>
                  <attach/>
                </anyNamespace>
              </mode>
            </allow>
          </anyNamespace>
        </mode>
        <context path="schema">
          <mode>
            <namespace ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
              match="attributes">
              <reject/>
            </namespace>
            <namespace ns=""
              match="attributes">
              <attach/>
            </namespace>
            <anyNamespace match="attributes">
              <allow/>
            </anyNamespace>
            <anyNamespace match="elements">
              <attach>
                <mode>
                  <anyNamespace>
                    <attach/>
                  </anyNamespace>
                </mode>
              </attach>
            </anyNamespace>
          </mode>
        </context>
        <context path="message">
```

```
        <mode>
          <namespace ns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
            match="attributes">
            <reject/>
          </namespace>
          <namespace ns=""
            match="attributes">
            <attach/>
          </namespace>
          <namespace ns="http://www.w3.org/XML/1998/namespace"
            match="attributes">
            <attach/>
          </namespace>
          <anyNamespace match="attributes">
            <allow/>
          </anyNamespace>
          <anyNamespace match="elements">
            <reject/>
          </anyNamespace>
        </mode>
      </context>
    </validate>
  </namespace>
 </mode>

</rules>
```

Annex D
(informative)

Example

D.1 General

This appendix shows two examples of NVDL. The first NVDL script handles RDF embedded within XHTML, while the second script handles the combination of XHTML2 and XForms. For each example, we demonstrate normalization of an NVDL script and then dispatching of an XML document.

D.2 RDF embedded within XHTML

D.2.1 Normalization

The following NVDL script combines schemas for XHTML and RDF.

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">

 <namespace ns="http://www.w3.org/1999/xhtml">
  <validate schema="xhtml.rng">
   <mode>
    <namespace ns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
     <validate schema="rdfxml.rng">
      <mode>
       <anyNamespace>
        <attach/>
       </anyNamespace>
      </mode>
     </validate>
    </namespace>
   </mode>
  </validate>
 </namespace>

</rules>
```

This NVDL script is normalized as follows.

```
<rules startMode="root"
 xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">

 <mode name="root">
  <namespace ns="http://www.w3.org/1999/xhtml">
   <validate schema="xhtml.rng"
    useMode="body"/>
  </namespace>
 </mode>

 <mode name="body">
  <namespace ns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <validate schema="rdfxml.rng"
    useMode="inRDF"/>
  </namespace>
 </mode>

 <mode name="inRDF">
  <anyNamespace>
   <attach/>
```

```
    </anyNamespace>
  </mode>

</rules>
```

D.2.2 Dispatching

D.2.2.1 General

Consider an XML document shown below:

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Some Page</title>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description rdf:about="http://www.w3.org/" dc:title="W3C Homepage"/>
    </rdf:RDF>
  </head>
  <body>
  </body>
</html>
```

This document has two element sections: xhtml and rdf. xhtml is for the namespace "http://www.w3.org/1999/xhtml" while rdf is for the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#". rdf has two attribute sections: @rdf and @dc. @rdf is for the namespace "http://www.w3.org/1999/xhtml" while @dc is for the namespace "http://purl.org/dc/elements/1.1/".

```
xhtml
  rdf
    @rdf
    @dc
```

D.2.2.2 Stage 1

There is one interpretation.

```
(validate against xhtml.rng, body)
  (validate against rdfxml.rng, inRDF)
    (attach, inRDF)
    (attach, inRDF)
```

D.2.2.3 Stage 2

rdf <- @rdf <- @dc is created. The value is shown below:

```
<{http://www.w3.org/1999/02/22-rdf-syntax-ns#}RDF [cx1]>
  <{http://www.w3.org/1999/02/22-rdf-syntax-ns#}Description [cx1]
      {http://www.w3.org/1999/02/22-rdf-syntax-ns#}about="http://www.w3.org/"
      {http://purl.org/dc/elements/1.1/}title="W3C Homepage">
  </>
</>
```

where cx1 is a context that maps rdf to "http://www.w3.org/1999/02/22-rdf-syntax-ns#" and dc to "http://purl.org/dc/elements/1.1/".

D.2.2.4 Stage 3

ValElem has two pairs, namely:

— (validate against xhtml.rng, xhtml)

— (validate against rdfxml.rng, rdf <- @rdf <- @dc)

ValAtt is empty.

D.2.2.5 Stage 4

PlanElem has two pairs, namely:

— (validate against xhtml.rng, removeSlots(xhtml))

— (validate against rdfxml.rng, removeSlots(rdf <- @rdf <- @dc))

xhtml has an element slot node (which is the place holder for rdf), but it is removed by removeSlots. rdf <- @rdf <- @dc does not have any slot nodes.

PlanAtt is empty.

D.2.2.6 Stage 5

The RELAX NG validator validates the following NVDL data model against xforms.rng.

```
<{http://www.w3.org/1999/xhtml}html [cx1]>
  <{http://www.w3.org/1999/xhtml}head [cx1]>
    <{http://www.w3.org/1999/xhtml}title [cx1]>Some Page</>
  </>
  <{http://www.w3.org/1999/xhtml}body [cx1]>
  </>
</>
```

The RDF validator validates the following NVDL data model against rdfxml.rng.

```
<{http://www.w3.org/1999/02/22-rdf-syntax-ns#}RDF [cx1]>
  <{http://www.w3.org/1999/02/22-rdf-syntax-ns#}Description [cx1]
      {http://www.w3.org/1999/02/22-rdf-syntax-ns#}about="http://www.w3.org/"
      {http://purl.org/dc/elements/1.1/}title="W3C Homepage">
  </>
</>
```

D.3 XHTML2 and XForms

D.3.1 Normalization

The following NVDL script combines schemas for XHTML2 and XForms.

```
<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0">

  <namespace ns="http://www.w3.org/2002/06/xhtml2">
    <a:documentation>We begin with XHTML2 sections.</a:documentation>
    <validate schema="xhtml2.rng">
```



```
    <a:documentation>This action invokes the XHTML2 schema.</a:documentation>
    <mode>
     <namespace ns="http://www.w3.org/2002/xforms">
      <a:documentation>XForms in XHTML2</a:documentation>
      <validate schema="xforms.rng">
       <a:documentation>The first action invokes the XForms schema
                for XForms in XHTML2.</a:documentation>
       <mode>
        <namespace ns="http://www.w3.org/2002/xforms">
         <a:documentation>XForms in ... in *XForm* in XHTML2</a:documentation>
         <attach message="Attaching a descendant XForms section."/>
        </namespace>
        <namespace ns="http://www.w3.org/2002/06/xhtml2">
         <a:documentation>XHTML2 in ... in *XForm* in XHTML2</a:documentation>
         <unwrap message="Skipping a descedant XHTML2 section."/>
        </namespace>
       </mode>
      </validate>
      <unwrap>
       <a:documentation>This action skip XForms in XHTML2.</a:documentation>
       <mode>
        <namespace ns="http://www.w3.org/2002/xforms">
         <a:documentation>XForms in ... in XForm in *XHTML2*</a:documentation>
         <unwrap message="Skipping a descendant XForm section."/>
        </namespace>
        <namespace ns="http://www.w3.org/2002/06/xhtml2">
         <a:documentation>XHTML2 in ... in XForm in *XHTML2*</a:documentation>
         <attach message="Attaching a descendant XHTML2 section."/>
        </namespace>
       </mode>
      </unwrap>
     </namespace>
    </mode>
   </validate>
  </namespace>

 </rules>
```

This NVDL script is normalized as follows.

```
 <rules startMode="root"
  xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">

  <mode name="root">
   <namespace ns="http://www.w3.org/2002/06/xhtml2"
    match="elements">
    <validate schema="xhtml2.rng"
     useMode="InXhtml2"/>
   </namespace>
   <anyNamespace match="elements">
    <validate useMode="root">
     <schema>
      <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0" />
     </schema>
    </validate>
   </anyNamespace>
   <anyNamespace match="attributes">
    <attach useMode="root"/>
   </anyNamespace>
  </mode>

  <mode name="InXhtml2">
   <namespace ns="http://www.w3.org/2002/xforms"
    match="elements">
    <validate schema="xforms.rng"
```

```
      useMode="InXformsInXhtml2"/>
    <unwrap useMode="SkippingXFormsInXhtml2"/>
   </namespace>
   <anyNamespace match="elements">
    <validate useMode="InXhtml2">
     <schema>
      <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0" />
     </schema>
    </validate>
   </anyNamespace>
   <anyNamespace match="attributes">
    <attach useMode="InXhtml2"/>
   </anyNamespace>
  </mode>

  <mode name="InXformsInXhtml2">
   <namespace ns="http://www.w3.org/2002/xforms"
    match="elements">
    <attach useMode="InXformsInXhtml2">
     <message>Attaching a descendant XForms section.</message>
    </attach>
   </namespace>
   <namespace ns="http://www.w3.org/2002/06/xhtml2"
    match="elements">
    <unwrap useMode="InXformsInXhtml2">
     <message>Skipping a descedant XHTML2 section.</message>
    </unwrap>
   </namespace>
   <anyNamespace match="elements">
    <validate useMode="InXformsInXhtml2">
     <schema>
      <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0" />
     </schema>
    </validate>
   </anyNamespace>
   <anyNamespace match="attributes">
    <attach/>
   </anyNamespace>
  </mode>

  <mode name="SkippingXFormsInXhtml2">
   <namespace ns="http://www.w3.org/2002/xforms"
    match="elements">
    <unwrap useMode="SkippingXFormsInXhtml2">
     <message>Skipping a descendant XForm section.</message>
    </unwrap>
   </namespace>
   <namespace ns="http://www.w3.org/2002/06/xhtml2"
    match="elements">
    <attach useMode="SkippingXFormsInXhtml2">
     <message>Attaching a descendant XHTML2 section.</message>
    </attach>
   </namespace>
   <anyNamespace match="elements">
    <validate useMode="SkippingXFormsInXhtml2">
     <schema>
      <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/reject/1.0" />
     </schema>
    </validate>
   </anyNamespace>
   <anyNamespace match="attributes">
    <attach/>
   </anyNamespace>
  </mode>

 </rules>
```

D.3.2 Dispatching

D.3.2.1 General

Consider an XML document shown below:

```
<?xml version="1.0"?>
<table
    xmlns:xforms="http://www.w3.org/2002/xforms"
    xmlns="http://www.w3.org/2002/06/xhtml2">
  <xforms:repeat id="lineset" nodeset="/my:lines/my:line">
    <tr>
      <td>
        <xforms:input ref="my:price">
          <p>
            <xforms:label>Line Item</xforms:label>
          </p>
        </xforms:input>
      </td>
    </tr>
  </xforms:repeat>
</table>
```

This document has six element sections. Three of them (say, xhtml1, xhtml2, and xhtml3) belong to the XHTML2 namespace and the others (say xforms1, xforms2, and xforms3) belong to the XForms namespace.

```
xhtml1
 xforms1
  xhtml2
   xforms2
    xhtml3
     xforms3
```

D.3.2.2 Stage 1

There are two interpretations.

```
(validate against xhtml2.rng, InXhtml2)
 (validate against xforms.rng, InXformsInXhtml2)
  (unwrap, InXformsInXhtml2)
   (attach, InXformsInXhtml2)
    (unwrap, InXformsInXhtml2)
     (attach, InXformsInXhtml2)

(validate against xhtml2.rng, InXhtml2)
 (unwrap, SkippingXFormsinXhtml2)
  (attach, SkippingXFormsinXhtml2)
   (unwrap, SkippingXFormsinXhtml2)
    (attach, SkippingXFormsinXhtml2)
     (unwrap, SkippingXFormsinXhtml2)
```

D.3.2.3 Stage 2

For the first interpretation, xhtml1, and xforms1 <- xforms2 <- xforms3 are created, where xforms1 <- xforms2 <- xforms3 is shown below:

```
<{http://www.w3.org/2002/xforms}repeat [cx1]
    {}id="lineset" nodeset="/my:lines/my:line">
        <{http://www.w3.org/2002/xforms}input  [cx1]
```

```
            {}ref="my:price">
              <{http://www.w3.org/2002/xforms}label  [cx1]>Line Item</>
          </>
    </>
```

where cx1 is a context that maps xforms to "http://www.w3.org/2002/xforms" and the empty prefix to "http://www.w3.org/2002/06/xhtml2".

For the second interpretation, xhtml1 <- xhtml2 <- xhtml3 is created, where xhtml1 <- xhtml2 <- xhtml3 is shown below:

```
<{http://www.w3.org/2002/06/xhtml2}table [cx1]
    xmlns:xforms="http://www.w3.org/2002/xforms"
    xmlns="">
      <{http://www.w3.org/2002/06/xhtml2}tr  [cx1]>
        <{http://www.w3.org/2002/06/xhtml2}td  [cx1]>
            <{http://www.w3.org/2002/06/xhtml2}p [cx1]>
            </>
        </>
      </>
</>
```

where cx1 is the same as in the previous example.

Whitespace in the examples is ajusted for readability.

D.3.2.4 Stage 3

ValElem has three pairs, namely:

— (validate against xhtml2.rng, xhtml1)

— (validate against xforms.rng, xforms1 <- xforms2 <- xforms3)

— (validate against xhtml2.rng, xhtml1 <- xhtml2 <- xhtml3)

ValAtt is empty.

D.3.2.5 Stage 4

PlanElem has two pairs, namely:

— (validate against xforms.rng, removeSlots(xforms1 <- xforms2 <- xforms3))

— (validate against xhtml2.rng, removeSlots(xhtml1 <- xhtml2 <- xhtml3))

Since no slot nodes occur in xforms1 <- xforms2 <- xforms3 or xhtml1 <- xhtml2 <- xhtml3, removeSlots in this example does not do anything.

PlanAtt is empty.

D.3.2.6 Stage 5

The RELAX NG validator validates xforms1 <- xforms2 <- xforms3 against xforms.rng, and

The RELAX NG validator validates xhtml1 <- xhtml2 <- xhtml3 against xhtml2.rng.

Bibliography

[1] *Guidelines for using W3C XML Schema Datatypes with RELAX NG*, OASIS Committee Specification, 7 September 2001, available at <http://www.oasis-open.org/committees/relax-ng/xsd-20010907.html>

[2] *RELAX Namespace (in Japanese)*, JIS Technical Report X 0044, 2001, available at <http://www.y-adagio.com/public/standards/tr_relax_ns/toc.htm>

[3] *RELAX Namespace*, ISO/IEC DTR 22250-2, 2002, available at <http://www.y-adagio.com/public/standards/iso_tr_relax_ns/dtr_22250-2.doc>

[4] *Modular Namespaces*, James Clark, 2003-01-31, available at <http://www.thaiopensource.com/relaxng/mns.html>

[5] *Namespace Switchboard*, Rick Jelliffe, 2003-04-02, available at <http://www.topologi.com/resources/NamespaceSwitchboard.html>

[6] *Namespace Routing Language*, James Clark, 2003-06-13, available at <http://www.thaiopensource.com/relaxng/nrl.html>