**ISO/IEC JTC 1/SC 34**

Date: 2005-12-21

**ISO/IEC FDIS 19757-4:2005(E)**

ISO/IEC JTC 1/SC 34/WG 1

Secretariat: Standards Council of Canada

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

Document type: International Standard
Document subtype:
Document stage: (50) Approval
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 *Information technology — 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 description language - CRDL*

— *Part 8: Document schema renaming language - DSRL*

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

— *Part 10: Validation management*

## Introduction

ISO/IEC 19757 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 ISO/IEC 19757 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 origin of NVDL is JIS TR X 0044[4], which was created and then submitted to ISO/IEC JTC1 as a fast-track ISO/IEC DTR 22250-2[5] by the Japanese national member body for SC 34.

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

## 1 Scope

This part of ISO/IEC 19757 specifies a Namespace-based Validation Dispatching Language (NVDL). An NVDL 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 NVDL schema also specifies which schemas are used by these validators. These schemas may be written in any schema languages, including those specified by ISO/IEC 19757.

## 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.

NOTE 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.

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 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 (Second Edition)*, W3C Recommendation, 28 October 2004, available at http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

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

IETF RFC 3986, *Uniform Resource Identifiers (URI): Generic Syntax*, Internet Standards Track Specification, January 2005, available at http://www.ietf.org/rfc/rfc3986.txt

IETF RFC 3987, *Internationalized Resource Identifiers (IRIs)*, Internet Standards Track Specification, January 2005, available at http://www.ietf.org/rfc/rfc3987.txt

## 3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19757-2 and the following apply.

NOTE Where a term is defined in this part of ISO/IEC 19757 and in ISO/IEC 19757-2, the definition given in this part of ISO/IEC 19757 applies.

**3.1**
**action**
a validate, reject, allow, attach, unwrap, cancelNestedActions, or attachPlaceholder element within an NVDL script

**3.2**
**attribute section**
a non-empty set of attributes having the same namespace name

**3.3**
**attribute slot node**
a slot for an attribute section

**3.4**
**NVDL dispatcher**
software module that determines whether an NVDL script is correct, creates validation candidates from an instance, and invokes validators for these validation candidates

**3.5**
**element section**
an element such that a single namespace name applies to itself and all descendant elements

**3.6**
**element slot node**
a slot for an element section

**3.7**
**full syntax**
syntax of an NVDL grammar before simplification

**3.8**
**instance**
XML document from which validation candidates are created

**3.9**
**media type**
a two-part identifier specifying the nature of the referenced data

**3.10**
**mode**
a mapping from namespaces to actions

**3.11**
**NVDL data model**
abstract representation of an XML document defined by this part of ISO/IEC 19757

**3.12**
**NVDL script**
specification for namespace-based validation dispatching

**3.13**
**parent element section**
an element section containing a slot node for another section

**3.14**
**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 /

**3.15**
**RELAX NG data model**
abstract representation of an XML document defined by ISO/IEC 19757-2

**3.16**
**section**
either an attribute section or element section

**3.17**
**simple syntax**
syntax of an NVDL script after simplification

**3.18**
**simplification**
transformation of an NVDL script in the full syntax to a script in the simple syntax

**3.19**
**slot node**
either an attribute slot node or element slot node

**3.20**
**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.

# 4 Notation

The following notations have been adopted in the formal definitios provided in this part of ISO/IEC 19757.

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

< {..} >: the namespace name 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* <- [*sn*, *x*]: an element constructed from *e* by replacing a slot node *sn* with an element sequence or attribute section *x*

|*e*|: the number of descendant elements in an element *e*

createPlaceholder(*s*): a placeholder element created from an element section *s* as specified in 7.6.

virtualElement(*s*): an empty element created from an attribute section *s* as specified in 7.7

path(*s*): the path from the parent element section of *s* to *s* as defined in 8.2

# 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 RELAX NG data model. Names, contexts, attributes, and strings in the NVDL data model are identical to those in the RELAX NG data model defined in ISO/IEC 19757-2. Elements in the NVDL data model are extended as described 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.

NOTE 1 Since contexts include mappings from prefixes to namespace names, validators invoked by NVDL dispatchers can handle qualified names occurring in attribute values or element contents.

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

An attribute slot node is a slot 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 either *e* or some descendant element of *e*.

An element slot node is a slot 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 either *e* or some descendant element of *e*.

NOTE 2 An element (say *e*) is not a descendant of another element (say *e′*) if *e′* has an element slot node for *e*.

NOTE 3 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 ISO/IEC 19757-2.

EXAMPLE 1 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 namespace name, 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.

EXAMPLE 2 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 namespace name 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 described in 6.2. Meanwhile, the simple syntax described in 6.3 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 following RELAX NG schema in the compact syntax. The same schema in the XML syntax is shown in Annex A.

```
namespace local = ""
default namespace nvdl = "http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"

start =
  element rules {
    (schemaType?,
     trigger*,
     (rule* | (attribute startMode { xsd:NCName }, mode+)))
    & foreign
  }

trigger =
  element trigger {
    (attribute ns { xsd:string },
    attribute nameList { list { xsd:NCName } })
    & foreign
  }

mode =
  element mode {
    (attribute name { xsd:NCName },
    includedMode*,
    rule*)
    & foreign
  }

includedMode =
  element mode {
    (attribute name {xsd:NCName}?,
    includedMode*,
    rule*)
    & foreign
  }
```

```
rule =
  element namespace {
    (attribute ns { xsd:string },
    attribute wildCard {xsd:string{maxLength = "1"}}?,
    ruleModel)
    & foreign
  }
  | element anyNamespace { ruleModel  & foreign}

ruleModel = attribute match { elementsOrAttributes }?, actions

elementsOrAttributes =
  list {
    ("elements", "attributes")
     | ("attributes", "elements")
     | "elements"
     | "attributes"
  }

actions =
  cancelAction |
  (noResultAction*, (noResultAction|resultAction), noResultAction*)

cancelAction =
  element cancelNestedActions {foreign}

noResultAction =
  element validate {
    (schemaType?,
    (message | option)*,
    schema,
    modeUsage)  & foreign
  }
  | element allow|reject { (message*, modeUsage) & foreign }

schema =
  attribute schema { xsd:anyURI } |
  element schema {(text | foreignElement), foreignAttribute*}

message =
    attribute message {text}
    | element message {(text, xmlAttribute*), nonXMLForeignAttribute*}

resultAction =
  element attach|attachPlaceholder|unwrap { (message*, modeUsage) & foreign }

option =
  element option {
    (attribute name { xsd:anyURI },
    attribute arg { text }?,
    attribute mustSupport { xsd:boolean }?)
    & foreign
  }

modeUsage =
  (attribute useMode { xsd:NCName }
   | nestedMode)?,
  element context {
    (attribute path { path },
     (attribute useMode { xsd:NCName }
      | nestedMode)?
    ) & foreign
  }*

nestedMode =
  element mode {
```

```
    (includedMode*,
    rule*)
    & foreign
  }

schemaType = attribute schemaType { mediaType }

## 5.1 of RFC 2045 allows <any (US-ASCII) CHAR except SPACE, CTLs,
## or tspecials>, where
##
##     tspecials :=  "(" / ")" / "<" / ">" / "@" /
##                   "," / ";" / ":" / "\" / <">
##                   "/" / "[" / "]" / "?" / "="
##

mediaType =
  xsd:string {
    pattern = "\s*"
            ~ "[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*"
            ~ "\\"
            ~ "[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*"
            ~ "\s*"}

path =
  xsd:string {
    pattern = "\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*"
            ~ "(\|\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*)*"
  }

foreignElement =
  element (* - nvdl:*) {attribute * {text}*, mixed{anyElement*}}

anyElement =
  element * {attribute * {text}*, mixed{anyElement*}}

foreignAttribute = attribute * - (nvdl:* | local:*) {text}

nonXMLForeignAttribute = attribute * - (xml:* | nvdl:* | local:*) {text}

xmlAttribute =
  attribute xml:lang {text}
  | attribute xml:space {"default" | "preserve"}
  | attribute xml:base {anyURI}

foreign = foreignAttribute*, foreignElement*
```

## 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 following RELAX NG schema in the compact syntax. The same schema in the XML syntax is shown in Annex B.

```
namespace local = ""
default namespace nvdl = "http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"

start =
  element rules {
    attribute startMode { xsd:NCName }, trigger*, mode+
  }

trigger =
  element trigger {
    attribute ns { xsd:string },
```

```
    attribute nameList { list { xsd:NCName } }
  }

mode =
  element mode {
    attribute name { xsd:NCName },
    rule*
  }

rule =
  element namespace {
    attribute ns { xsd:string },
    attribute wildCard {xsd:string{maxLength = "1"}},
    ruleModel
  }
  | element anyNamespace { ruleModel }

ruleModel = attribute match { elementsOrAttributes }, actions

elementsOrAttributes =
  "elements" | "attributes"

actions =
  (noResultAction*, (noResultAction|resultAction), noResultAction*)

noResultAction =
  element validate {
    schemaType?,
    (message | option)*,
    schema,
    modeUsage
  }

schema =
  attribute schema { xsd:anyURI } |
  element schema {(text | foreignElement)}

message =
    element message {text & attribute xml:lang {text}}

resultAction =
  element attach|attachPlaceholder|unwrap { message*, modeUsage }

option =
  element option {
    attribute name { xsd:anyURI },
    attribute arg { text }?,
    attribute mustSupport { xsd:boolean }
  }

modeUsage =
  attribute useMode { xsd:NCName },
  element context {
    attribute path { path },
    attribute useMode { xsd:NCName }
  }*

schemaType = attribute schemaType { mediaType }

## 5.1 of RFC 2045 allows <any (US-ASCII) CHAR except SPACE, CTLs,
## or tspecials>, where
##
##     tspecials :=  "(" / ")" / "<" / ">" / "@" /
##                   "," / ";" / ":" / "\" / <">
##                   "/" / "[" / "]" / "?" / "="
##
```

```
mediaType =
  xsd:string {
    pattern = "\s*"
            ~ "[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*"
            ~ "\\"
            ~ "[0-9A-Za-z!#$%&'\*\+\-\.\^_`\{\|\}~]*"
            ~ "\s*"}

path =
  xsd:string {
    pattern = "\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*"
            ~ "(\|\s*(/\s*)?\i\c*(\s*/\s*\i\c*)*\s*)*"
  }

foreignElement =
  element (* - nvdl:*) {attribute * {text}*, mixed{anyElement*}}

anyElement =
  element * {attribute * {text}*, mixed{anyElement*}}
```

## 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

Attributes having qualified names 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] property 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 message attribute

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

### 6.4.5 mustSupport attribute

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

### 6.4.6 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.

### 6.4.7 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.

### 6.4.8 child mode elements of validate, allow, reject, attach, unwrap, attachPlaceholder, or context elements

In this transformation rule, validate, allow, reject, attach, unwrap, attachPlaceholder, or context elements are transformed so that they do not have mode elements as children.

If a mode element occurs as a child of a validate, allow, reject, attach, unwrap, attachPlaceholder, or context element that has one and only one ancestor mode element, this mode element becomes 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, unwrap, or attachPlaceholder 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 within validate, allow, reject, attach, unwrap, attachPlaceholder, or context elements.

### 6.4.9 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 the element. The match attribute of the first copy is transformed into match="elements", while that of the second is transformed into match="attributes".

If a namespace element does not specify the wildCard attribute, wildCard="*" is added.

### 6.4.10 mode inclusion

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

NOTE It is intended that external parsed entities or XInclude[9] are used for syntactically including external NVDL script fragments. The transformation defined here allows post-processing after such syntactical inclusion.

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

An anyNamespace element within a lowest-level mode is said to be overridden when a mode directly containing this mode has another anyNamespace element and these anyNamespace elements specify the same value for the match attribute.

An namespace element within a lowest-level mode is said to be overridden when a mode directly containing this mode has another namespace element and these namespace elements specify the same value for the match attribute, the same value for the ns attribute, and the same value for the wildCard attribute.

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

— Each namespace or anyNamespace element within *md* is removed if it has a cancelNestedActions element as a child.

— Each namespace or anyNamespace element within *md* is removed if it is overridden.

— *md* is replaced by its children

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

### 6.4.11 competition within 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 of namespace elements formally. Let $w_1$ be the value of the wildCard attribute specified by the first namespace element and let $ns_1$ be the value of the ns attribute specified by the same element. Also let $w_2$ and $ns_2$ be those specified by the second namespace element. These namespace elements compete when two pairs ($ns_1$, $w_1$) and ($ns_2$, $w_2$) compete. These two pairs compete if and only if:

— Case 1: both $ns_1$ and $ns_2$ are empty strings,

— Case 2: either (1) $ns_1$="" and $ns_2$="$w_2$" or (2) $ns_2$="" and $ns_1$="$w_1$",

— Case 3: both $ns_1$ and $ns_2$ begin with the same character that is neither $w_1$ nor $w_2$, and a pair ($nsTail_1$, $w_1$) competes with another pair ($nsTail_2$, $w_2$), where $nsTail_1$ and $nsTail_2$ are obtained by stripping the first characters of $ns_1$ and $ns_2$, respectively,

NOTE 1 Case 3 implies that ("a", "*") and ("a", "?") compete.

— Case 4: $w_1$ is not "", $ns_1$ begins with $w_1$, and a pair ($ns_1$, $w_1$) competes with another pair ($nsTail_2$, $w_2$), where $nsTail_2$ is obtained by stripping the first character of $ns_2$, or

NOTE 2 Case 4 implies that ("*", "*") and ("a", "") compete.

— Case 5: $w_2$ is not "", $ns_2$ begins with $w_2$, and a pair ($ns_2$, $w_2$) competes with another pair ($nsTail_1$, $w_1$), where $nsTail_1$ is obtained by stripping the first character of $ns_1$

NOTE 3 The restriction in this clause ensures that only one namespace or anyNamespace in a mode is triggered by a namespace name, so that at most one unwrap or attach is invoked.

### 6.4.12 default anyNamespace

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.

### 6.4.13 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/predefinedSchema/1.0"/></schema>,
```

where allow in the namespace "http://purl.oclc.org/dsdl/nvdl/ns/predefinedSchema/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 reject element followed by

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

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

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

#### 6.4.14 useMode attribute

For each validate, attach, attachPlaceholder, unwrap, or context element without a useMode attribute, a useMode attribute is added. The value shall be copied from the name attribute of the ancestor mode 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 operation 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 name of some attribute of *e*. This attribute section contains those attributes of *e* which belong to *n*, and the attribute slot node is the slot 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 non-root element (say *e*), if *e* and its parent element belong to different namespaces. The element section is represented by *e*, and the element slot node is the slot for the element section. Then, the child sequence of the parent element of *e* is modified by replacing *e* with the element slot node.

EXAMPLE 1 Five element sections are generated from the data model in Example 1 of 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]></>
```

The first element section is the parent element section of the second and third ones, while the third one is the parent element section of the fourth and fifth ones.

EXAMPLE 2 Five element sections and five attribute sections are generated from the data model in Example 2 of 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]></>
```

The first element section is the parent element section of the second and third ones, while the third one is the parent element section of the fourth and fifth ones.

The first element section is the parent element section of the first and second attribute sections, while the third element section is the parent element section of the third, fourth, and fifth attribute sections.

## 7.3 Decomposition of element sections by `trigger` elements

Controlled by `trigger` elements, this operation further decomposes element sections to element sections. Decomposition is achieved using the following rules:

First, select an element *e* occurring in some element section (say *es*) such that, for some `trigger` element *t*, (1) *e* is located by *t*, (2) *e* is not the root of *es*, and (3) the parent element of *e* is not located by *t*. Here an element is said to be located by *t* if (1) the namespace name of this element is character-by-character identical to the value of the `ns` attribute of *t* and (2) the local name of this element is one of the tokens specified by the `nameList` attribute of *t*.

Second, decompose the element section *es* into two element sections. The first element section is represented by *e*, and an element slot node is created as the slot for this element section. The second element section is *es*, but the child sequence of the parent element of *e* is modified by replacing *e* with the element slot node.

The selection and decomposition rules are repeated until no elements can be selected.

NOTE This decomposition operation is useful when namespaces do not provide enough information for creating element sections from XML instances.

EXAMPLE 1 Consider an XML document as follows. This document conforms to DocBook 4.2[6]. Since the first version of Docbook precedes XML namespaces, DocBook 4.2 does not use namespaces. Note that DocBook 4.2 borrows table structures from the CALS DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<book>
  <article>
    <title>Docbook containing a CALS table</title>
    <para>para</para>
    <table>
      <title/>
      <tgroup cols="2">
        <tbody>
          <row><entry>1</entry><entry>A</entry></row>
          <row><entry>2</entry><entry>B</entry></row>
        </tbody>
      </tgroup>
    </table>
  </article>
</book>
```

The following trigger element can be used to extract a table from the example Docbook document .

```
<trigger ns="" nameList="table"/>
```

Two element sections are constructed. The first element section does not have any local names borrowed from the CALS table.

```
<{}book [cx1]>
  <{}article [cx1]>
    <{}title [cx1]>Docbook containing a CALS table</>
    <{}para [cx1]>para</>
    $esn1
  </>
</>
```

The second element section has local names borrowed from the CALS table and has no other local names.

```
<{}table [cx1]>
  <{}title [cx1]></>
  <{}tgroup [cx1] $asn1>
    <{}tbody>
      <{}row><{}entry>1</><{}entry>A</></>
      <{}row><{}entry>2</><{}entry>B</></>
    </>
  </>
</>
```

EXAMPLE 2 Consider an XML document as follows. This document conforms to XHTML 2.0[8]. Note that switch and case elements are borrowed from XForms 1.0[7], but they are in the XHTML 2.0 namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/2002/06/xhtml2">
  <head>
    <title xml:lang="en">XHTML 2.0 containing XForms</title>
  </head>
  <body>
    <p>para</p>
    <switch>
      <case>
        <p>embedded para</p>
      </case>
    </switch>
  </body>
</html>
```

The following trigger elements can be used to extract the switch and case elements.

```
<trigger ns="http://www.w3.org/2002/06/xhtml2" nameList="html head title body p"/>
<trigger ns="http://www.w3.org/2002/06/xhtml2" nameList="switch"/>
```

Three element sections are constructed. The first element section uses XHTML 2.0.

```
<{http://www.w3.org/2002/06/xhtml2}html [cx1]>
   <{http://www.w3.org/2002/06/xhtml2}head [cx1]>
      <{http://www.w3.org/2002/06/xhtml2 [cx1]}title  $asn1>XHTML 2.0 containing XForms</>
   </>
   <{http://www.w3.org/2002/06/xhtml2}body [cx1]>
      <{http://www.w3.org/2002/06/xhtml2}p [cx1]>para</>
      $esn1
   </>
</>
```

The second element section uses XForms, although the namespace is XHTML 2.0.

```
<{http://www.w3.org/2002/06/xhtml2}switch [cx1]>
   <{http://www.w3.org/2002/06/xhtml2}case [cx1]>
      $esn2
   </>
</>
```

The third element section uses XHTML 2.0.

```
<{http://www.w3.org/2002/06/xhtml2}p [cx1]>embedded para</>
```

## 7.4 Attaching attribute sections to elements

An attribute section (say *x*) can be attached to an element (say *e*) at an attribute slot node (say asn). This is done by replacing *asn* with those attributes in *x*. The result is denoted *e* <- [*asn*, *x*].

EXAMPLE If the third attribute section is attached to the third element section in Example 2 of 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
</>
```

## 7.5 Attaching element sequences to elements

A sequence of elements (say *x*) can be attached to an element (say *e*) at an element slot node *esn*. This is done by replacing *esn* with *x*. If two text nodes become adjacent (this happens only when *x* is an empty sequence), they are concatenated to form a single text node. The result is denoted *e* <- [*esn*, *x*].

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

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

EXAMPLE 2 If the fourth element section followed by the fifth element section is attached to the first element section in Example 1 of 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]></>
  </>
</>
```

### 7.6 Creating placeholder elements from element sections

A placeholder element can be created from a given element section (say *s*). The placeholder is an empty element such that the namespace name is "http://purl.oclc.org/dsdl/nvdl/ns/instance/1.0", the local name is placeholder, and the ns and localName attributes specify the namespace name and local name of the root element of *s*, respectively. The result is denoted createPlaceholder(*s*).

EXAMPLE The placeholder element created from the third element section shown in Example 1 of 7.2 is:

```
<{http://purl.oclc.org/dsdl/nvdl/ns/instance/1.0}placeholder ns="http://www.example.com/one" localName="foo2"></>.
```

### 7.7 Converting attribute sections to empty elements

An attribute section (say *s*) can be converted to an empty element. This is done as follows. An empty element is created so that the namespace name is "http://purl.oclc.org/dsdl/nvdl/ns/instance/1.0", the local name is virtualElement, the child sequence is empty, and the attribute set contains those attributes in *s*. The result is denoted virtualElement(*s*).

EXAMPLE The empty element created from the second attribute section shown in Example 2 of 7.2 is:

```
<{http://purl.oclc.org/dsdl/nvdl/ns/instance/1.0}virtualElement
  {http://www.example.com/two}bar1="_"
  {http://www.example.com/two}bar2="_"></>.
```

## 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: Creating element and attribute sections

— Stage 2: Constructing interpretations

— Stage 3: Combining sections

— Stage 4: Filtering of the combined sections

— 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

The following functions and predicates are used later in this clause.

NOTE 1 Note that paths contain local names but do not contain prefixes or namespace names.

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 may be replaced with different strings. When *wc* is "", match(*nsValue*, *wc*, *uri*) holds if and only if *nsValue* is identical to *uri*.

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*.

path(*s*) is a function that constructs a path from a given element section *s*. Let *s'* be the parent element section of *s*. Then, path(*s*) is a path that begins with the local name of the root of element section *s'* and ends with the local name of the parent element of the element slot node for *s*.

matchPathExp(*pathExp*, *p*) holds when a path *p* matches *pathExp*. Path expression *pathExp* is a list of one or more choices separated by '|', where each choice is a list of one or more unqualified names separated by '/', optionally preceded by '/'. Path *p* matches *pathExp* when:

— Case 1: some choice in *pathExp* begins with '/' and the sequence of NCNames in *p* is identical to that in this choice, or

— Case 2: some choice in *pathExp* does not begin with '/' and some ending sequence of NCNames in *p* is identical to the sequence of NCNames in this choice.

NOTE 2 matchPathExp is similar to XPath, but does not use namespace names or namespace prefixes.

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 (say *c*) of *action*, *mode* is specified by the useMode attribute of *c*, matchPathExp(*pathExp*, *p*) holds where *pathExp* is the value of the path attribute of *c*, and, for any preceding sibling context element (say *c'*) of *c*, matchPathExp(*pathExp'*, *p*) does not hold where *pathExp'* is the value of the path attribute of *c'*.

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: Creating element and attribute sections

This stage first creates element sections and attribute sections from a given instance, as defined in 7.2. Then, the element sections are further decomposed by the trigger elements in a given NVDL script, as defined in 7.3.

### 8.4 Stage 2: Constructing interpretations

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

An interpretation *I* is a function that maps an element section (*s*) in the given instance to a pair of an action ($I_A(s)$) and a mode ($I_M(s)$) and maps an attribute section (*s'*) in the given instance to an action ($I_A(s')$) 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 that contains no NCNames.

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

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

More than one interpretation may exist for a single instance when a single namespace or anyNamespace element has multiple actions. All possible interpretations are constructed.

### 8.5 Stage 3: 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:

— Case 1: $I_A(s)$ is an attach action

attBubble[*I*](*s*) = *s*

— Case 2: Otherwise

attBubble[*I*](*s*) = {}

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:

— Case 1: $I_A(s)$ is a validate action

elemBubble[*I*](*s*) = ()

— Case 2: $I_A(s)$ is an attach action

elemBubble[$I$]($s$) = ($X_m$), where

$X_0 = Y_n$,

$X_1 = X_0$ <- [$esn_1$, elemBubble[$I$]($s_1$)],

$X_2 = X_1$ <- [$esn_2$, elemBubble[$I$]($s_2$)],

...

$X_m = X_{m-1}$ <- [$esn_m$, elemBubble[$I$]($s_m$)],

$s_1$, $s_2$, ..., $s_m$ are the child element sections of $s$,

$esn_1$, $esn_2$, ..., $esn_m$ are the element slot nodes for $s_1$, $s_2$, ..., $s$, respectively, where

$Y_0 = s$,

$Y_1 = Y_0$ <- [$asn_1$, attBubble[$I$]($a_1$)],

$Y_2 = Y_1$ <- [$asn_2$, attBubble[$I$]($a_2$)],

...

$Y_n = Y_{n-1}$ <- [$asn_n$, attBubble[$I$]($a_n$)],

$a_1$, $a_2$, ..., $a_n$ are the child attribute sections of $s$, and

$asn_1$, $asn_2$, ..., $asn_n$ are the attribute slot nodes for $a_1$, $a_2$, ..., $a_n$, respectively.

NOTE 1 The number (m) of element sections and that (n) of attribute sections may be different.

— Case 3: $I_A(s)$ is an attachPlaceholder action

elemBubble[$I$]($s$) = (createPlaceholder(s))

— Case 4: $I_A(s)$ is an unwrap action

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 $s$.

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.

— Case 1: $I_A(s)$ is a validate action

syn[$I$]($s$) = ($X_m$), where

$X_0 = Y_n$,

$X_1 = X_0$ <- [$esn_1$, elemBubble[$I$]($s_1$)],

$X_2 = X_1$ <- [$esn_2$, elemBubble[$I$]($s_2$)],

...

$X_m = X_{m-1}$ <- [$esn_m$, elemBubble[$I$]($s_m$)],

$s_1$, $s_2$, ..., $s_m$ are the child element sections of $s$,

$esn_1$, $esn_2$, ..., $esn_m$ are the element slot nodes for $s_1$, $s_2$, ..., $s$, respectively, where

$Y_0 = s$,

$Y_1 = Y_0$ <- [$asn_1$, attBubble[$I$]($a_1$)],

$Y_2 = Y_1$ <- [$asn_2$, attBubble[$I$]($a_2$)],

...

$Y_n = Y_{n-1}$ <- [$asn_n$, attBubble[$I$]($a_n$)],

$a_1$, $a_2$, ..., $a_n$ are the child attribute sections of $s$, and

$asn_1$, $asn_2$, ..., $asn_n$ are attribute slot nodes for $a_1$, $a_2$, ..., $a_n$, respectively.

NOTE 2 The number (m) of element sections and that (n) of attribute sections may be different.

## 8.6 Stage 4: Filtering of the combined sections

This stage constructs two sets *PlanElem* and *PlanAtt* of ($v$, $e$) , where $v$ is a validate element in the NVDL script and $e$ is an element.

— *PlanElem* = { ($v$, $e$) | ($s$, $v$, $e$) in $Z$, and |$e$| >= |$e'$| for any ($s$, $v$, $e'$) in $Z$}, where $Z$ is the set of triplets ($s$, $v$, $e$) such that $s$ is an element section, $v$ is a validate action, $e$ is an element, $v = I_A(s)$, and $e$ = syn[$I$]($s$), and

NOTE |$e$| >= |$e'$| selects the "biggest" validation candidate among those having the same validate action applied to the same element section.

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

## 8.7 Stage 5: Validation

### 8.7.1 General

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

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

### 8.7.2 Determining schemas and schema languages

If $v$ contains a schema element as a child, its content is used as the 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 a foreign element, its namespace is used for determining the schema language.

NOTE 1 A media type for the RELAX NG compact syntax application/relax-ng-compact-syntax is expected to be registered at IANA.

If $v$ references to a schema by the schema attribute, the schema is constructed by dereferencing its value, which is

 

an IRI (see IETF RFC 3987), and the schema language is determined from the schema and the schemaType attribute of *v*.

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

— When the media type is an XML media type (i.e., application/xml, text/xml, or */*+xml), the MIME entity shall be parsed as an XML document in accordance with the applicable RFC (at the time 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 2 IETF RFC 3023 does not define the interpretation of fragment identifiers for application/xml, text/xml, or */*+xml.

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

### 8.7.3 Schema rewriting for attribute sections

For each (*v*, *e*) in *PlanAtt*, the NVDL dispatcher writes the schema *s* in *v*. If *s* is written in RELAX NG, *s* is rewritten as <element><anyName/>*s*</element>. If *s* is written in some other schema language and that schema language provides a schema rewriting mechanism for NVDL, *s* is rewritten by applying that mechanism. Otherwise, *s* is not rewritten.

NOTE 1 While attribute sections are converted to elements at Stage 4 by virtualElement, schemas for attribute sets are converted to schemas for elements by this rewriting.

NOTE 2 Schema languages should provide schema rewriting mechanisms for NVDL. If they do not, schema authors have to specify virtualElement explicitly.

### 8.7.4 Options

When a validator is invoked for a pair (*v*, *e*), 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 This part of ISO/IEC 19757 does not introduce any standard options. Schema languages or their implementations are recommended to introduce options so that they can interwork with NVDL.

## 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/predefinedSchema/1.0".

# Annex A
(normative)

# Full syntax in RELAX NG

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

```
<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>
     <zeroOrMore>
      <ref name="trigger"/>
     </zeroOrMore>
     <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="trigger">
  <element name="trigger">
   <interleave>
    <group>
     <attribute name="ns">
      <data type="string"/>
     </attribute>
     <attribute name="nameList">
      <list>
       <data type="NCName"/>
      </list>
     </attribute>
    </group>
    <ref name="foreign"/>
   </interleave>
  </element>
 </define>
```

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

<define name="includedMode">
 <element name="mode">
  <interleave>
   <group>
    <optional>
     <attribute name="name">
      <data type="NCName"/>
     </attribute>
    </optional>
    <zeroOrMore>
     <ref name="includedMode"/>
    </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="maxLength">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="cancelNestedActions">
    <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>attachPlaceholder</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="includedMode"/>
      </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>
   <zeroOrMore>
    <ref name="trigger"/>
   </zeroOrMore>
   <oneOrMore>
    <ref name="mode"/>
   </oneOrMore>
  </element>
 </start>

 <define name="trigger">
  <element name="trigger">
   <attribute name="ns">
    <data type="string"/>
   </attribute>
   <attribute name="nameList">
    <list>
     <data type="NCName"/>
    </list>
   </attribute>
  </element>
 </define>

 <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>
    <attribute name="wildCard">
     <data type="string">
      <param name="maxLength">1</param>
     </data>
    </attribute>
    <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>attachPlaceholder</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 in 6.2 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>
   <zeroOrMore>
    <ref name="trigger"/>
   </zeroOrMore>
   <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="trigger">
  <element name="trigger">
   <attribute name="ns">
    <data type="string"/>
   </attribute>
   <attribute name="nameList">
    <list>
     <data type="NCName"/>
    </list>
   </attribute>
  </element>
 </define>

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

```
    </attribute>
    <zeroOrMore>
      <ref name="includedMode"/>
    </zeroOrMore>
    <zeroOrMore>
      <ref name="rule"/>
    </zeroOrMore>
  </element>
</define>

<define name="includedMode">
  <element name="mode">
    <optional>
      <attribute name="name">
        <data type="NCName"/>
      </attribute>
    </optional>
    <zeroOrMore>
      <ref name="includedMode"/>
    </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="maxLength">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="cancelNestedActions">
    <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>attachPlaceholder</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="includedMode"/>
     </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 XHTML 2.0 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"
   match="elements"
   wildCard="*">
   <validate schema="xhtml.rng"
    useMode="body"/>
  </namespace>
  <anyNamespace match="elements">
   <validate useMode="root">
    <schema>
     <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/predefinedSchema/1.0" />
    </schema>
   </validate>
  </anyNamespace>
  <anyNamespace match="attributes">
   <attach useMode="root"/>
  </anyNamespace>
```

```
  </mode>

  <mode name="body">
   <namespace ns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    match="elements"
    wildCard="*">
    <validate schema="rdfxml.rng"
     useMode="inRDF"/>
   </namespace>
   <anyNamespace match="elements">
    <validate useMode="root">
     <schema>
      <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/predefinedSchema/1.0" />
     </schema>
    </validate>
   </anyNamespace>
   <anyNamespace match="attributes">
    <attach useMode="root"/>
   </anyNamespace>
  </mode>

  <mode name="inRDF">
   <anyNamespace match="elements">
    <attach useMode="inRDF"/>
   </anyNamespace>
   <anyNamespace match="attributes">
    <attach useMode="inRDF"/>
   </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>
```

#### D.2.2.2 Stage 1

Two element sections, xhtml and rdf, are created from this document. 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#". The element slot node for rdf is *esnRdf* and occurs in xhtml. rdf has two attribute slot nodes, *asnRdf* and *asnDc*. The attribute section for *asnRdf* is @rdf, which is for the namespace "http://www.w3.org/1999/xhtml". The attribute section for *asnDc* is @dc, which is for the namespace "http://purl.org/dc/elements/1.1/".

```
xhtml [esnRdf]
  rdf [asnRdf, asnDc]
    @rdf
```

@dc

#### D.2.2.3 Stage 2

There is one interpretation.

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

#### D.2.2.4 Stage 3

xhtml <- [*esnRdf*, ()] is created. The value is shown below:

```
<{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]>
  </>
</>
```

rdf <- [*asnRdf*, @rdf] <- [*asnDc*, @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">
  </>
</>
```

Here 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.2.2.5 Stage 4

PlanElem has two pairs, namely:

— (validate against xhtml.rng, xhtml <- [*esnRdf*, ()])

— (validate against rdfxml.rng, rdf <- [*asnRdf*, @rdf] <- [*asnDc*, @dc])

PlanAtt is empty.

#### D.2.2.6 Stage 5

The RELAX NG validator validates xhtml <- [*esnRdf*, ()] against xhtml.rng.

The RDF validator validates rdf <- [*asnRdf*, @rdf] <- [*asnDc*, @dc] against rdfxml.rng.

### D.3 XHTML 2.0 and XForms

#### D.3.1 Normalization

The following NVDL script combines schemas for XHTML 2.0 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 XForms in *XHTML2*</a:documentation>
        <unwrap message="Skipping a descendant XForms section."/>
       </namespace>
       <namespace ns="http://www.w3.org/2002/06/xhtml2">
        <a:documentation>XHTML2 in ... in XForms 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"
   wildCard="*">
   <validate schema="xhtml2.rng"
    useMode="InXhtml2"/>
  </namespace>
  <anyNamespace match="elements">
   <validate useMode="root">
    <schema>
     <reject xmlns="http://purl.oclc.org/dsdl/nvdl/ns/predefinedSchema/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"
    wildCard="*">
    <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/predefinedSchema/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"
    wildCard="*">
    <attach useMode="InXformsInXhtml2">
      <message>Attaching a descendant XForms section.</message>
    </attach>
  </namespace>
  <namespace ns="http://www.w3.org/2002/06/xhtml2"
    match="elements"
    wildCard="*">
    <unwrap useMode="InXformsInXhtml2">
      <message>Skipping a descedant XHTML2 section.</message>
    </unwrap>
  </namespace>
  <anyNamespace match="elements">
    <validate useMode="InXformsInXhtml2">
      <schema>
      </schema>
    </validate>
  </anyNamespace>
  <anyNamespace match="attributes">
    <attach/>
  </anyNamespace>
</mode>

<mode name="SkippingXFormsInXhtml2">
  <namespace ns="http://www.w3.org/2002/xforms"
    match="elements"
    wildCard="*">
    <unwrap useMode="SkippingXFormsInXhtml2">
      <message>Skipping a descendant XForms section.</message>
    </unwrap>
  </namespace>
  <namespace ns="http://www.w3.org/2002/06/xhtml2"
    match="elements"
    wildCard="*">
    <attach useMode="SkippingXFormsInXhtml2">
      <message>Attaching a descendant XHTML2 section.</message>
    </attach>
  </namespace>
  <anyNamespace match="elements">
    <validate useMode="SkippingXFormsInXhtml2">
```

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

#### D.3.2.2 Stage 1

Six element sections are created from this document. Three of them (say, xhtml1, xhtml2, and xhtml3) belong to the XHTML 2.0 namespace and the others (say xforms1, xforms2, and xforms3) belong to the XForms namespace. The element slot nodes for xhtml2 and xhtml3 are esnXhtml2 and esnXhtml3, respectively, and occur in xforms1 and xforms2, respectively. The element slot nodes for xforms1, xforms2, and xforms3 are esnXforms1, esnXforms2, and esnXforms3, respectively, and occur in xhtml1, xhtml2, and xhtml3, respectively.

```
xhtml1 [esnXforms1]
  xforms1 [esnXhtml2]
    xhtml2 [esnXforms2]
      xforms2 [esnXhtml3]
        xhtml3 [esnXforms3]
          xforms3
```

#### D.3.2.3 Stage 2

There are two interpretations:

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

and

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

#### D.3.2.4 Stage 3

For the first interpretation, xhtml1 <- [esnXforms1, ()] and xforms1 <- [esnXhtml2, (xforms2 <- [esnXhtml3, xforms3]] are created. They are:

```
<{http://www.w3.org/2002/06/xhtml2}table [cx1]>
</>
```

and

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

respectively, 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 <- [esnXforms1, xhtml2 <- [esnXforms2, xhtml3]] is created. It is shown below:

```
<{http://www.w3.org/2002/06/xhtml2}table [cx1]>
     <{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 adjusted for readability.

#### D.3.2.5 Stage 4

PlanElem has two pairs, namely:

— (validate against xforms.rng, xforms1 <- [esnXhtml2, xforms2 <- [esnXhtml3, xforms3]])

— (validate against xhtml2.rng, xhtml1 <- [esnXforms1, xhtml2 <- [esnXforms2, xhtml3]])

Note that X has a triplet (xhtml1, validate against xhtml2.rng, xhtml1 <- [esnXforms1, ()]). However, it is not used for constructing PlanElem, since xhtml1 <- [esnXforms1, xhtml2 <- [esnXforms2, xhtml3]] is "bigger".

PlanAtt is empty.

#### D.3.2.6 Stage 5

The RELAX NG validator validates xforms1 <- [esnXhtml2, xforms2 <- [esnXhtml3, xforms3]] against xforms.rng, and validates xhtml1 <- [esnXforms1, xhtml2 <- [esnXforms2, xhtml3]] against xhtml2.rng.

# Bibliography

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

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

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

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

[5] *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

[6] *The DocBook Document Type*, OASIS Committee Specification 4.2, 16 July 2002, available at http://www.oasis-open.org/docbook/specs/cs-docbook-docbook-4.2.html

[7] *XForms 1.0*, W3C Recommendation, 14 October 2003, available at http://www.w3.org/TR/2003/REC-xforms-20031014/

[8] *XHTML 2.0*, W3C Working Draft, 27 May 2005, available at http://www.w3.org/TR/2005/WD-xhtml2-20050527

[9] *XML Inclusions(XInclude) Version 1.0*, W3C Recommendation, 20 December 2004, available at http://www.w3.org/TR/2004/REC-xinclude-20041220/