Internet Engineering Task Force (IETF)
Request for Comments: 6110
Category: Standards Track
ISSN: 2070-1721

L. Lhotka, Ed.
CESNET

February 2011

Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content

Abstract

This document specifies the mapping rules for translating YANG data models into Document Schema Definition Languages (DSDL), a coordinated set of XML schema languages standardized as ISO/IEC 19757. The following DSDL schema languages are addressed by the mapping: Regular Language for XML Next Generation (RELAX NG), Schematron, and Schematron and Document Schema Renaming Language (DSRL). The mapping takes one or more YANG modules and produces a set of DSDL schemas for a selected target document type -- datastore content, Network Configuration Protocol (NETCONF) messages, etc. Procedures for schema-based validation of such documents are also discussed.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6110.

Table of Contents

1. Introduction

The NETCONF Working Group has completed a base protocol used for configuration management [RFC4741]. This base specification defines protocol bindings and an XML container syntax for configuration and management operations, but does not include a data modeling language or accompanying rules for how to model configuration and state information carried by NETCONF. The IETF Operations Area has a long tradition of defining data for Simple Network Management Protocol (SNMP) Management Information Bases (MIB) modules [RFC1157] using the Structure of Management Information (SMI) language [RFC2578] to model its data. While this specific modeling approach has a number of well-understood problems, most of the data modeling features provided by SMI are still considered extremely important. Simply modeling the valid syntax without the additional semantic relationships has caused significant interoperability problems in the past.

The NETCONF community concluded that a data modeling framework is needed to support ongoing development of IETF and vendor-defined management information modules. The NETMOD Working Group was chartered to design a modeling language defining the semantics of operational data, configuration data, event notifications, and operations, with focus on "human-friendliness", i.e., readability and ease of use. The result is the YANG data modeling language [RFC6020], which now serves for the normative description of NETCONF data models.

Since NETCONF uses XML for encoding its messages, it is natural to express the constraints on NETCONF content using standard XML schema languages. For this purpose, the NETMOD WG selected the Document Schema Definition Languages (DSDL) that is being standardized as ISO/IEC 19757 [DSDL]. The DSDL framework comprises a set of XML

schema languages that address grammar rules, semantic constraints, and other data modeling aspects, but also, and more importantly, do it in a coordinated and consistent way. While it is true that some DSDL parts have not been standardized yet and are still work in progress, the three parts that the YANG-to-DSDL mapping relies upon -- Regular Language for XML Next Generation (RELAX NG), Schematron and Document Schema Renaming Language (DSRL) -- already have the status of an ISO/ IEC International Standard and are supported in a number of software tools.

This document contains a specification of a mapping that translates YANG data models to XML schemas utilizing a subset of the DSDL schema languages. The mapping procedure is divided into two steps: In the first step, the structure of the data tree, signatures of remote procedure call (RPC) operations, and notifications are expressed as the so-called "hybrid schema" -- a single RELAX NG schema with annotations representing additional data model information (metadata, documentation, semantic constraints, default values, etc.). The second step then generates a coordinated set of DSDL schemas that can be used for validating specific XML documents such as client requests, server responses or notifications, perhaps also taking into account additional context such as active capabilities or features.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [RFC4741]:

- client
- datastore
- message
- operation
- server

The following terms are defined in [RFC6020]:

- augment
- base type
- built-in type

- configuration data
- container
- data model
- data node
- data tree
- derived type
- device deviation
- extension
- feature
- grouping
- instance identifier
- leaf-list
- list
- mandatory node
- module
- RPC
- RPC operation
- schema node
- schema tree
- state data
- submodule
- top-level data node
- uses

The following terms are defined in [XML-INFOSET]:

- attribute
- document
- document element
- document type declaration (DTD)
- element
- information set
- namespace

In the text, the following typographic conventions are used:

- YANG statement keywords are delimited by single quotes.
- XML element names are delimited by "<" and ">" characters.
- Names of XML attributes are prefixed by the "@" character.
- Other literal values are delimited by double quotes.

XML element names are always written with explicit namespace prefixes corresponding to the following XML vocabularies:

"a" DTD compatibility annotations [RNG-DTD];

"dc" Dublin Core metadata elements [RFC5013];

"dsrl" Document Semantics Renaming Language [DSRL];

"en" NETCONF event notifications [RFC5277];

"nc" NETCONF protocol [RFC4741];

"nma" NETMOD-specific schema annotations (see Section 5.3);

"nmf" NETMOD-specific XML Path Language (XPath) extension functions (see Section 12.7);

"rng" RELAX NG [RNG];

"sch" ISO Schematron [Schematron];

"xsd" W3C XML Schema [XSD].

The following table shows the mapping of these prefixes to namespace URIs.

Prefix	Namespace URI
a	http://relaxng.org/ns/compatibility/annotations/1.0
dc	http://purl.org/dc/terms
dsrl	http://purl.oclc.org/dsdl/dsrl
en	urn:ietf:params:xml:ns:netconf:notification:1.0
nc	urn:ietf:params:xml:ns:netconf:base:1.0
nma	urn:ietf:params:xml:ns:netmod:dsdl-annotations:1
nmf	urn:ietf:params:xml:ns:netmod:xpath-extensions:1
rng	http://relaxng.org/ns/structure/1.0
sch	http://purl.oclc.org/dsdl/schematron
xsd	http://www.w3.org/2001/XMLSchema

Table 1: Used namespace prefixes and corresponding URIs

2.1. Glossary of New Terms

- ancestor data type: Any data type from which a given data type is (transitively) derived.
- ancestor built-in data type: The built-in data type that is at the start of the type derivation chain for a given data type.
- hybrid schema: A RELAX NG schema with annotations, which embodies the same information as the source YANG module(s). See Section 8.1 for details.
- implicit node: A data node that, if it is not instantiated in a data tree, may be added to the information set of that data tree (configuration, RPC input or output, notification) without changing the semantics of the data tree.

3. Objectives and Motivation

The main objective of this work is to complement YANG as a data modeling language with validation capabilities of DSDL schema languages, namely RELAX NG, Schematron, and DSRL. This document describes the correspondence between grammatical, semantic, and data type constraints expressed in YANG and equivalent DSDL patterns and rules. The ultimate goal is to be able to capture all substantial information contained in YANG modules and express it in DSDL schemas. While the mapping from YANG to DSDL described in this document may in principle be invertible, the inverse mapping from DSDL to YANG is beyond the scope of this document.

XML-based information models and XML-encoded data appear in several different forms in various phases of YANG data modeling and NETCONF workflow -- configuration datastore contents, RPC requests and replies, and notifications. Moreover, RPC operations are characterized by an inherent diversity resulting from selective availability of capabilities and features. YANG modules can also define new RPC operations. The mapping should be able to accommodate this variability and generate schemas that are specifically tailored to a particular situation and thus considerably more effective for validation than generic all-encompassing schemas.

In order to cope with this variability, we assume that the DSDL schemas will be generated on demand for a particular purpose from the available collection of YANG modules and their lifetime will be relatively short. In other words, we don't envision that any collection of DSDL schemas will be created and maintained over an extended period of time in parallel to YANG modules.

The generated schemas are primarily intended as input to existing XML schema validators and other off-the-shelf tools. However, the schemas may also be perused by developers and users as a formal representation of constraints on a particular XML-encoded data object. Consequently, our secondary goal is to keep the schemas as readable as possible. To this end, the complexity of the mapping is distributed into two steps:

1. The first step maps one or more YANG modules to the so-called hybrid schema, which is a single RELAX NG schema that describes grammatical constraints for the main data tree as well as for RPC operations and notifications. Semantic constraints and other information appearing in the input YANG modules is recorded in the hybrid schema in the form of foreign namespace annotations. The output of the first step can thus be considered a virtually complete equivalent of the input YANG modules. It cannot, however, be directly used for any validation.

2. In the second step, the hybrid schema from step 1 is transformed further to a coordinated set of fully conformant DSDL schemas containing constraints for a particular data object and a specific situation. The DSDL schemas are intended mainly for machine validation using off-the-shelf tools.

4. DSDL Schema Languages

Document Schema Definition Languages (DSDL) is a framework of schema languages that is being developed as the International Standard ISO/IEC 19757 [DSDL]. Unlike other approaches to XML document validation, most notably W3C XML Schema Definition (XSD) [XSD], the DSDL framework adheres to the principle of "small languages": each of the DSDL constituents is a stand-alone schema language with a relatively narrow purpose and focus. Together, these schema languages may be used in a coordinated way to accomplish various validation tasks.

The mapping described in this document uses three of the DSDL schema languages, namely RELAX NG [RNG], Schematron [Schematron], and DSRL [DSRL].

4.1. RELAX NG

RELAX NG (pronounced "relaxing") is an XML schema language for grammar-based validation and Part 2 of the ISO/IEC DSDL family of standards [RNG]. Like XSD, it is able to describe constraints on the structure and contents of XML documents. However, unlike the DTD [XML] and XSD schema languages, RELAX NG intentionally avoids any infoset augmentation such as defining default values. In the DSDL architecture, the particular task of defining and applying default values is delegated to another schema language, DSRL (see Section 4.3).

As its base data type library, RELAX NG uses the W3C XML Schema Datatypes [XSD-D]; but unlike XSD, other data type libraries may be used along with it or even replace it if necessary.

RELAX NG is very liberal in accepting annotations from other namespaces. With a few exceptions, such annotations may be placed anywhere in the schema and need no encapsulating elements such as <xsd:annotation> in XSD.

RELAX NG schemas can be represented in two equivalent syntaxes: XML and compact. The compact syntax is described in Annex C of the RELAX NG specification [RNG-CS], which was added to the standard in 2006 (Amendment 1). Automatic bidirectional conversions between the two syntaxes can be accomplished using several tools, for example, Trang [Trang].

For its terseness and readability, the compact syntax is often the preferred form for publishing RELAX NG schemas, whereas validators and other software tools usually work with the XML syntax. However, the compact syntax has two drawbacks:

- External annotations make the compact syntax schema considerably less readable. While in the XML syntax the annotating elements and attributes are represented in a simple and uniform way (XML elements and attributes from foreign namespaces), the compact syntax uses as many as four different syntactic constructs: documentation, grammar, initial, and following annotations. Therefore, the impact of annotations on readability is often much stronger for the compact syntax than it is for the XML syntax.

- In a computer program, it is more difficult to generate the compact syntax than the XML syntax. While a number of software libraries exist that make it easy to create an XML tree in the memory and then serialize it, no such aid is available for the compact syntax.

For these reasons, the mapping specification in this document uses exclusively the XML syntax. Where appropriate, though, the schemas resulting from the translation MAY be presented in the equivalent compact syntax.

RELAX NG elements are qualified with the namespace URI "http://relaxng.org/ns/structure/1.0". The namespace of the XSD data type library is "http://www.w3.org/2001/XMLSchema-datatypes".

4.2. Schematron

Schematron is Part 3 of DSDL that reached the status of a full ISO/IEC standard in 2006 [Schematron]. In contrast to the traditional schema languages such as DTD, XSD, or RELAX NG, which are based on the concept of a formal grammar, Schematron utilizes a rule-based approach. Its rules may specify arbitrary conditions involving data from different parts of an XML document. Each rule consists of three essential components:

- context - an XPath expression that defines the set of locations where the rule is to be applied;

- assert or report condition - another XPath expression that is evaluated relative to the location matched by the context expression;

- human-readable message that is displayed when the assert condition is false or report condition is true.

The difference between the assert and report condition is that the former is positive in that it states a condition that a valid document has to satisfy, whereas the latter specifies an error condition.

Schematron draws most of its expressive power from XPath [XPath] and Extensible Stylesheet Language Transformations (XSLT) [XSLT]. ISO Schematron allows for dynamic query language binding so that the following XML query languages can be used: STX, XSLT 1.0, XSLT 1.1, EXSLT, XSLT 2.0, XPath 1.0, XPath 2.0, and XQuery 1.0 (this list may be extended in the future).

Human-readable error messages are another feature that sets Schematron apart from other common schema languages. The messages may even contain XPath expressions that are evaluated in the actual context and thus refer to information items in the XML document being validated.

Another feature of Schematron that is used by the mapping are abstract patterns. These work essentially as macros and may also contain parameters which are supplied when the abstract pattern is used.

Schematron elements are qualified with namespace URI "http://purl.oclc.org/dsdl/schematron".

4.3. Document Semantics Renaming Language (DSRL)

DSRL (pronounced "disrule") is Part 8 of DSDL that reached the status of a full ISO/IEC standard in 2008 [DSRL]. Unlike RELAX NG and Schematron, DSRL is allowed to modify XML information set of the validated document. While DSRL is primarily intended for renaming XML elements and attributes, it can also define default values for XML attributes and default contents for XML elements or subtrees so that the default contents are inserted if they are missing in the validated documents. The latter feature is used by the YANG-to-DSDL mapping for representing YANG default contents consisting of leaf nodes with default values and their ancestor non-presence containers.

DSRL elements are qualified with namespace URI "http://purl.oclc.org/dsdl/dsrl".

5. Additional Annotations

Besides the DSDL schema languages, the mapping also uses three sets of annotations that are added as foreign-namespace attributes and elements to RELAX NG schemas.

Two of the annotation sets -- Dublin Core elements and DTD compatibility annotations -- are standard vocabularies for representing metadata and documentation, respectively. Although these data model items are not used for formal validation, they quite often carry important information for data model implementers. Therefore, they SHOULD be included in the hybrid schema and MAY also appear in the final validation schemas.

The third set are NETMOD-specific annotations. They are specifically designed for the hybrid schema and convey semantic constraints and other information that cannot be expressed directly in RELAX NG. In the second mapping step, these annotations are converted to Schematron and DSRL rules.

5.1. Dublin Core Metadata Elements

Dublin Core is a system of metadata elements that was originally created for describing metadata of World Wide Web resources in order to facilitate their automated lookup. Later it was accepted as a standard for describing metadata of arbitrary resources. This specification uses the definition from [RFC5013].

Dublin Core elements are qualified with namespace URI "http://purl.org/dc/terms".

5.2. RELAX NG DTD Compatibility Annotations

DTD compatibility annotations are a part of the RELAX NG DTD Compatibility specification [RNG-DTD]. YANG-to-DSDL mapping uses only the <a:documentation> annotation for representing YANG 'description' and 'reference' texts.

Note that there is no intention to make the resulting schemas DTD-compatible, the main reason for using these annotations is technical: they are well supported and adequately formatted by several RELAX NG tools.

DTD compatibility annotations are qualified with namespace URI "http://relaxng.org/ns/compatibility/annotations/1.0".

5.3. NETMOD-Specific Annotations

NETMOD-specific annotations are XML elements and attributes that are qualified with the namespace URI "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1" and that appear in various locations of the hybrid schema. YANG statements are mapped to these annotations in a straightforward way. In most cases, the annotation attributes and elements have the same name as the corresponding YANG statement.

Table 2 lists, alphabetically, the names of NETMOD-specific annotation attributes (prefixed with "@") and elements (in angle brackets) along with a reference to the section where their use is described. Appendix A contains a RELAX NG schema for this annotation vocabulary.

annotation	section	note
@nma:config	10.9	
<nma:data>	8.1	4
@nma:default	10.12	
<nma:error-app-tag>	10.16	1
<nma:error-message>	10.17	1
@nma:if-feature	10.22	
@nma:implicit	10.11, 10.7, 10.12	
<nma:input>	8.1	4
<nma:instance-identifier>	10.53.7	2
@nma:key	10.26	
@nma:leaf-list	10.28	
@nma:leafref	10.53.8	
@nma:mandatory	10.8	
@nma:max-elements	10.28	
@nma:min-elements	10.28	

@nma:module	10.34	
<nma:must>	10.35	3
<nma:notification>	8.1	4
<nma:notifications>	8.1	4
@nma:ordered-by	10.38	
<nma:output>	8.1	4
<nma:rpc>	8.1	4
<nma:rpcs>	8.1	4
@nma:status	10.51	
@nma:unique	10.55	
@nma:units	10.56	
@nma:when	10.59	

Table 2: NETMOD-specific annotations

Notes:

1. Appears only as a subelement of <nma:must>.

2. Has an optional attribute @require-instance.

3. Has a mandatory attribute @assert and two optional subelements <nma:error-app-tag> and <nma:error-message>.

4. Marker element in the hybrid schema.

6. Overview of the Mapping

This section gives an overview of the YANG-to-DSDL mapping, its inputs and outputs. Figure 1 presents an overall structure of the mapping:

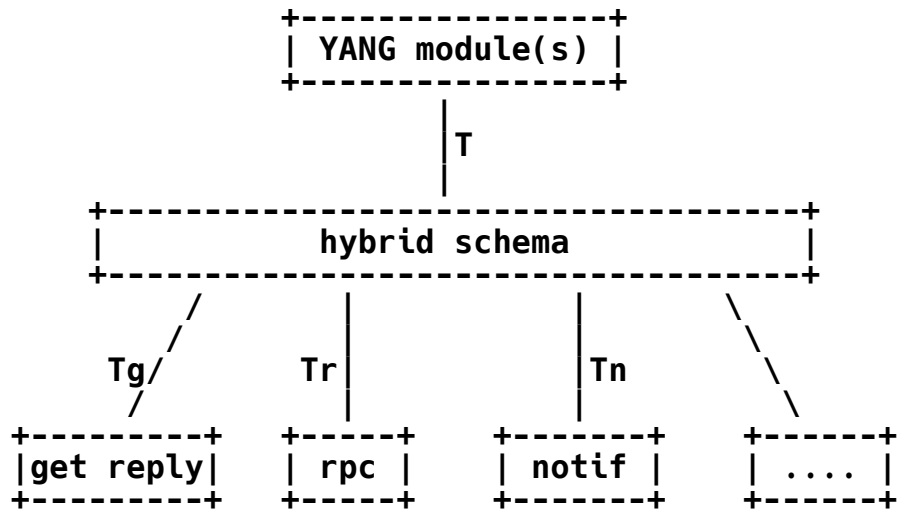

Figure 1: Structure of the mapping

The mapping procedure is divided into two steps:

1. Transformation T in the first step maps one or more YANG modules to the hybrid schema (see Section 8.1). Constraints that cannot be expressed directly in RELAX NG (list key definitions, 'must' statements, etc.) and various documentation texts are recorded in the schema as foreign-namespace annotations.

2. In the second step, the hybrid schema may be transformed in multiple ways to a coordinated set of DSDL schemas that can be used for validating a particular data object in a specific context. Figure 1 shows three simple possibilities as examples. In the process, appropriate parts of the hybrid schema are extracted and specific annotations transformed to equivalent, but usually more complex, Schematron patterns, DSRL element maps, etc.

An implementation of the mapping algorithm MUST accept one or more valid YANG modules as its input. It is important to be able to process multiple YANG modules together since multiple modules may be negotiated for a NETCONF session and the contents of the configuration datastore is then obtained as the union of data trees specified by the individual modules, which may also lead to multiple root nodes of the datastore hierarchy. In addition, the input modules may be further coupled by the 'augment' statement in which one module augments the data tree of another module.

It is also assumed that the algorithm has access, perhaps on demand, to all YANG modules that the input modules import (directly or transitively).

Other information contained in input YANG modules, such as semantic constraints and default values, is recorded in the hybrid schema as annotations -- XML attributes or elements qualified with the namespace URI "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1". Metadata describing the YANG modules are mapped to Dublin Core annotations elements (Section 5.1). Finally, documentation strings are mapped to <a:documentation> elements belonging to the DTD compatibility vocabulary (Section 5.2).

The output of the second step is a coordinated set of three DSDL schemas corresponding to a specific data object and context:

- RELAX NG schema describing the grammatical and data type constraints;
- Schematron schema expressing other constraints such as uniqueness of list keys or user-specified semantic rules;
- DSRL schema containing the specification of default contents.

7. NETCONF Content Validation

This section describes how the schemas generated by the YANG-to-DSDL mapping are supposed to be applied for validating XML instance documents such as the contents of a datastore or various NETCONF messages.

The validation proceeds in the following steps, which are also illustrated in Figure 2:

1. The XML instance document is checked for grammatical and data type validity using the RELAX NG schema.
2. Default values for leaf nodes have to be applied and their ancestor containers added where necessary. It is important to add the implicit nodes before the next validation step because YANG specification [RFC6020] requires that the data tree against which XPath expressions are evaluated already has all defaults filled-in. Note that this step modifies the information set of the validated XML document.
3. The semantic constraints are checked using the Schematron schema.

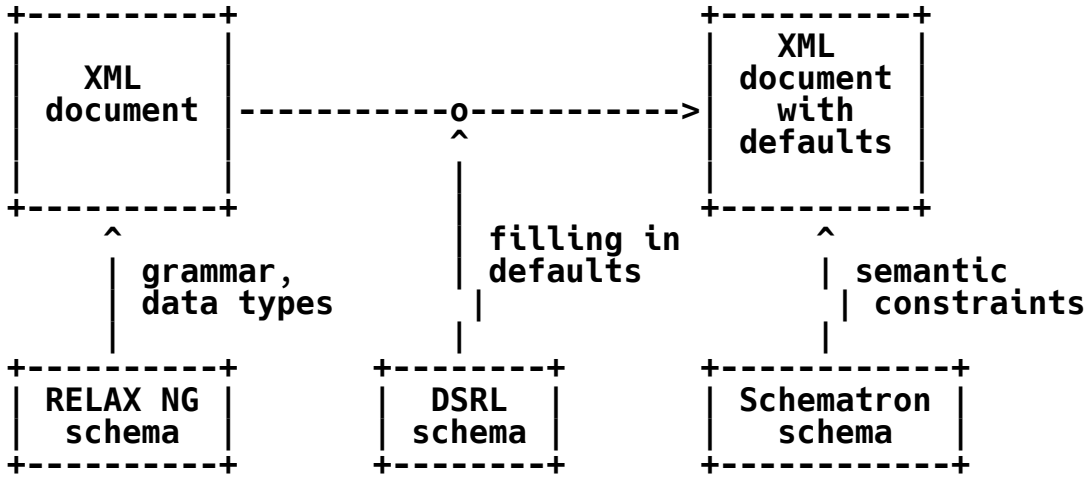

Figure 2: Outline of the validation procedure

8. Design Considerations

YANG data models could, in principle, be mapped to the DSDL schemas in a number of ways. The mapping procedure described in this document uses several specific design decisions that are discussed in the following subsections.

8.1. Hybrid Schema

As was explained in Section 6, the first step of the mapping produces an intermediate document -- the hybrid schema, which specifies all constraints for the entire data model using the RELAX NG syntax and additional annotations. In cannot be directly used for validation -- as a matter of fact, it is not even a valid RELAX NG schema because it contains multiple schemas demarcated by special annotation elements.

Every input YANG module corresponds to exactly one embedded grammar in the hybrid schema. This separation of input YANG modules allows each embedded grammar to include named pattern definitions into its own namespace, which is important for mapping YANG groupings (see Section 9.2 for additional details).

In addition to grammatical and data type constraints, YANG modules provide other important information that cannot be expressed in a RELAX NG schema: semantic constraints, default values, metadata, documentation, and so on. Such information items are represented in

the hybrid schema as XML attributes and elements belonging to the namespace with the following URI: "urn:ietf:params:xml:ns:netmod:dsdl-annotations:1". A complete list of these annotations is given in Section 5.3, detailed rules about their use are then contained in the following sections.

YANG modules define data models not only for configuration and state data but also for (multiple) RPC operations [RFC4741] and/or event notifications [RFC5277]. In order to be able to capture all three types of data models in one schema document, the hybrid schema uses special markers that enclose sub-schemas for configuration and state data, individual RPC operations (both input and output part) and individual notifications.

The markers are the following XML elements in the namespace of NETMOD-specific annotations (URI urn:ietf:params:xml:ns:netmod:dsdl-annotations:1):

Element name	Role
nma:data	encloses configuration and state data
nma:rpcs	encloses all RPC operations
nma:rpc	encloses an individual RPC operation
nma:input	encloses an RPC request
nma:output	encloses an RPC reply
nma:notifications	encloses all notifications
nma:notification	encloses an individual notification

Table 3: Marker elements in the hybrid schema

For example, consider a data model formed by two YANG modules "example-a" and "example-b" that define nodes in the namespaces "http://example.com/ns/example-a" and "http://example.com/ns/example-b". Module "example-a" defines configuration/state data, RPC methods and notifications, whereas "example-b" defines only configuration/state data. The hybrid schema can then be schematically represented as follows:

```
  <grammar xmlns="http://relaxng.org/ns/structure/1.0"
           xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
           xmlns:exa="http://example.com/ns/example-a"
           xmlns:exb="http://example.com/ns/example-b"
           datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
    <start>
      <grammar nma:module="example-a"
               ns="http://example.com/ns/example-a">
        <start>
          <nma:data>
            ...configuration and state data defined in "example-a"...
          </nma:data>
          <nma:rpcs>
            <nma:rpc>
              <nma:input>
                <element name="exa:myrpc">
                  ...
                </element>
              </nma:input>
              <nma:output>
                ...
              </nma:output>
            </nma:rpc>
            ...
          </nma:rpcs>
          <nma:notifications>
            <nma:notification>
              <element name="exa:mynotif">
                ...
              </element>
            </nma:notification>
            ...
          </nma:notifications>
        </start>
        ...local named pattern definitions of example-a...
      </grammar>
      <grammar nma:module="example-b"
               ns="http://example.com/ns/example-a">
        <start>
          <nma:data>
            ...configuration and state data defined in "example-b"...
          </nma:data>
          <nma:rpcs/>
          <nma:notifications/>
        </start>
        ...local named pattern definitions of example-b...
      </grammar>
    </start>
```

```
    ...global named pattern definitions...
  </grammar>
```

A complete hybrid schema for the data model of a DHCP server is given in Appendix C.2.

8.2. Modularity

Both YANG and RELAX NG offer means for modularity, i.e., for splitting the contents of a full schema into separate modules and combining or reusing them in various ways. However, the approaches taken by YANG and RELAX NG differ. Modularity in RELAX NG is suitable for ad hoc combinations of a small number of schemas whereas YANG assumes a large set of modules similar to SNMP MIB modules. The following differences are important:

- In YANG, whenever module A imports module B, it gets access to the definitions (groupings and typedefs) appearing at the top level of module B. However, no part of data tree from module B is imported along with it. In contrast, the <rng:include> pattern in RELAX NG imports both definitions of named patterns and the entire schema tree from the included schema.

- The names of imported YANG groupings and typedefs are qualified with the namespace of the imported module. On the other hand, the names of data nodes contained inside the imported groupings, when used within the importing module, become part of the importing module's namespace. In RELAX NG, the names of patterns are unqualified and so named patterns defined in both the importing and imported module share the same flat namespace. The contents of RELAX NG named patterns may either keep the namespace of the schema where they are defined or inherit the namespace of the importing module, analogically to YANG. However, in order to achieve the latter behavior, the definitions of named patterns must be included from an external schema, which has to be prepared in a special way (see [Vli04], Chapter 11).

In order to map, as much as possible, the modularity of YANG to RELAX NG, a validating RELAX NG schema (the result of the second mapping step) has to be split into two files, one of them containing all global definitions that are mapped from top-level YANG groupings appearing in all input YANG module. This RELAX NG schema MUST NOT define any namespace via the @ns attribute.

The other RELAX NG schema file then defines actual data trees mapped from input YANG modules, each of them enclosed in an own embedded grammar. Those embedded grammars, in which at least one of the global definitions is used, MUST include the first schema with

definitions and also MUST define the local namespace using the @ns attribute. This way, the global definitions can be used inside different embedded grammar, each time accepting a different local namespace.

Named pattern definitions that are mapped from non-top-level YANG groupings MUST be placed inside the embedded grammar corresponding to the YANG module where the grouping is defined.

In the hybrid schema, we need to distinguish the global and non-global named pattern definitions while still keeping the hybrid schema in one file. This is accomplished in the following way:

- Every global definition MUST be placed as a child of the outer <rng:grammar> element (the document root of the hybrid schema).
- Every non-global definitions MUST be placed as a child of the corresponding embedded <rng:grammar> element.

YANG also allows for splitting a module into a number of submodules. However, as submodules have no impact on the scope of identifiers and namespaces, the modularity based on submodules is not mapped in any way. The contents of submodules is therefore handled as if the submodule text appeared directly in the main module.

8.3. Granularity

RELAX NG supports different styles of schema structuring: one extreme, often called "Russian Doll", specifies the structure of an XML instance document in a single hierarchy. The other extreme, the flat style, uses a similar approach as the Data Type Definition (DTD) schema language -- every XML element corresponds to a named pattern definition. In practice, some compromise between the two extremes is usually chosen.

YANG supports both styles in principle, too, but in most cases the modules are organized in a way closer to the "Russian Doll" style, which provides a better insight into the structure of the configuration data. Groupings are usually defined only for contents that are prepared for reuse in multiple places via the 'uses' statement. In contrast, RELAX NG schemas tend to be much flatter, because finer granularity is also needed in RELAX NG for extensibility of the schemas -- it is only possible to replace or modify schema fragments that are factored out as named patterns. For YANG, this is not an issue since its 'augment' and 'refine' statements can delve, by using path expressions, into arbitrary depths of existing structures.

In general, it is not feasible to map YANG's powerful extension mechanisms to those available in RELAX NG. For this reason, the mapping essentially keeps the granularity of the original YANG data model: YANG groupings and definitions of derived types usually have direct counterparts in definitions of named patterns in the resulting RELAX NG schema.

8.4. Handling of XML Namespaces

Most modern XML schema languages, including RELAX NG, Schematron, and DSRL, support schemas for so-called compound XML documents that contain elements from multiple namespaces. This is useful for our purpose since the YANG-to-DSDL mapping allows for multiple input YANG modules, which naturally leads to compound document schemas.

RELAX NG offers two alternatives for defining the target namespaces in the schema:

1. First possibility is the traditional XML way via the @xmlns:xxx attribute.

2. One of the target namespace URIs may be declared using the @ns attribute.

In both the hybrid schema and validation RELAX NG schemas generated in the second step, the namespaces MUST be declared as follows:

1. The root <rng:grammar> MUST have @xmlns:xxx attributes declaring prefixes of all namespaces that are used in the data model. The prefixes SHOULD be identical to those defined in the 'prefix' statements. An implementation of the mapping MUST resolve all collisions in the prefixes defined by different input modules, if there are any.

2. Each embedded <rng:grammar> element MUST declare the namespace of the corresponding module using the @ns attribute. This way, the names of nodes defined by global named patterns are able to adopt the local namespace of each embedded grammar, as explained in Section 8.2.

This setup is illustrated by the example at the end of Section 8.1.

DSRL schemas may declare any number of target namespaces via the standard XML attributes xmlns:xxx.

In contrast, Schematron requires all used namespaces to be defined in the <sch:ns> subelements of the document element <sch:schema>.

9. Mapping YANG Data Models to the Hybrid Schema

This section explains the main principles governing the first step of the mapping. Its result is the hybrid schema that is described in Section 8.1.

A detailed specification of the mapping of individual YANG statements is contained in Section 10.

9.1. Occurrence Rules for Data Nodes

In DSDL schema languages, occurrence constraints for a node are always localized together with that node. In a RELAX NG schema, for example, the <rng:optional> pattern appears as the parent element of the pattern defining a leaf or non-leaf element. Similarly, DSRL specifies default contents separately for every single node, be it a leaf or non-leaf element.

For leaf nodes in YANG modules, the occurrence constraints are also easily inferred from the substatements of 'leaf'. On the other hand, for a YANG container, it is often necessary to examine its entire subtree in order to determine the container's occurrence constraints.

Therefore, one of the goals of the first mapping step is to infer the occurrence constraints for all data nodes and mark, accordingly, the corresponding <rng:element> patterns in the hybrid schema so that any transformation procedure in the second mapping step can simply use this information and need not examine the subtree again.

First, it has to be decided whether a given data node must always be present in a valid configuration. If so, such a node is called mandatory, otherwise it is called optional. This constraint is closely related to the notion of mandatory nodes in Section 3.1 in [RFC6020]. The only difference is that this document also considers list keys to be mandatory.

The other occurrence constraint has to do with the semantics of the 'default' statement and the possibility of removing empty non-presence containers. As a result, the information set of a valid configuration may be modified by adding or removing certain leaf or container elements without changing the meaning of the configuration. In this document, such elements are called implicit. In the hybrid schema, they can be identified as RELAX NG patterns having either the @nma:default or the @nma:implicit attribute.

Note that both occurrence constraints apply to containers at the top level of the data tree, and then also to other containers under the additional condition that their parent node exists in the instance document. For example, consider the following YANG fragment:

```
container outer {
    presence 'Presence of "outer" means something.';
    container c1 {
        leaf foo {
            type uint8;
            default 1;
        }
    }
    container c2 {
        leaf-list bar {
            type uint8;
            min-elements 0;
        }
    }
    container c3 {
        leaf baz {
            type uint8;
            mandatory true;
        }
    }
}
```

Here, container "outer" has the 'presence' substatement, which means that it is optional and not implicit. If "outer" is not present in a configuration, its child containers are not present as well. However, if "outer" does exist, it makes sense to ask which of its child containers are optional and which are implicit. In this case, "c1" is optional and implicit, "c2" is optional but not implicit, and "c3" is mandatory (and therefore not implicit).

The following subsections give precise rules for determining whether a container is optional or mandatory and whether it is implicit. In order to simplify the recursive definition of these occurrence characteristics, it is useful to define them also for other types of YANG schema nodes, i.e., leaf, list, leaf-list, anyxml, and choice.

9.1.1. Optional and Mandatory Nodes

The decision whether a given node is mandatory or optional is governed by the following rules:

- Leaf, anyxml, and choice nodes are mandatory if they contain the substatement "mandatory true;". For a choice node, this means that at least one node from exactly one case branch must exist.
- In addition, a leaf node is mandatory if it is declared as a list key.
- A list or leaf-list node is mandatory if it contains the 'min-elements' substatement with an argument value greater than zero.
- A container node is mandatory if its definition does not contain the 'presence' substatement and at least one of its child nodes is mandatory.

A node that is not mandatory is said to be optional.

In RELAX NG, definitions of nodes that are optional must be explicitly wrapped in the <rng:optional> element. The mapping MUST use the above rules to determine whether a YANG node is optional, and if so, insert the <rng:optional> element in the hybrid schema.

However, alternatives in <rng:choice> MUST NOT be defined as optional in the hybrid schema. If a choice in YANG is not mandatory, <rng:optional> MUST be used to wrap the entire <rng:choice> pattern.

9.1.2. Implicit Nodes

The following rules are used to determine whether a given data node is implicit:

- List, leaf-list, and anyxml nodes are never implicit.
- A leaf node is implicit if and only if it has a default value, defined either directly or via its data type.
- A container node is implicit if and only if it does not have the 'presence' substatement, none of its children are mandatory, and at least one child is implicit.

In the hybrid schema, all implicit containers, as well as leafs that obtain their default value from a typedef and don't have the @nma:default attribute, MUST be marked with @nma:implicit attribute having the value of "true".

Note that Section 7.9.3 in [RFC6020] specifies other rules that must be taken into account when deciding whether or not a given container or leaf appearing inside a case of a choice is ultimately implicit. Specifically, a leaf or container under a case can be implicit only

if the case appears in the argument of the choice's 'default' statement. However, this is not sufficient by itself but also depends on the particular instance XML document, namely on the presence or absence of nodes from other (non-default) cases. The details are explained in Section 11.3.

9.2. Mapping YANG Groupings and Typedefs

YANG groupings and typedefs are generally mapped to RELAX NG named patterns. There are, however, several caveats that the mapping has to take into account.

First of all, YANG typedefs and groupings may appear at all levels of the module hierarchy and are subject to lexical scoping, see Section 5.5 in [RFC6020]. Second, top-level symbols from external modules may be imported as qualified names represented using the external module namespace prefix and the name of the symbol. In contrast, named patterns in RELAX NG (both local and imported via the <rng:include> pattern) share the same namespace and within a grammar they are always global -- their definitions may only appear at the top level as children of the <rng:grammar> element. Consequently, whenever YANG groupings and typedefs are mapped to RELAX NG named pattern definitions, their names MUST be disambiguated in order to avoid naming conflicts. The mapping uses the following procedure for mangling the names of groupings and type definitions:

- Names of groupings and typedefs appearing at the top level of the YANG module hierarchy are prefixed with the module name and two underscore characters ("__").

- Names of other groupings and typedefs, i.e., those that do not appear at the top level of a YANG module, are prefixed with the module name, double underscore, and then the names of all ancestor data nodes separated by double underscore.

- Finally, since the names of groupings and typedefs in YANG have different namespaces, an additional underscore character is added to the beginning of the mangled names of all groupings.

An additional complication is caused by the YANG rules for subelement ordering (see, e.g., Section 7.5.7 in [RFC6020]): in RPC input and output parameters, subelements must follow the order specified in the data model; otherwise, the order is arbitrary. Consequently, if a grouping is used both in RPC input/output parameters and elsewhere, it MUST be mapped to two different named pattern definitions -- one with fixed order and the other with arbitrary order. To distinguish them, the "__rpc" suffix MUST be appended to the version with fixed order.

EXAMPLE. Consider the following YANG module that imports the standard module "ietf-inet-types" [RFC6021]:

```
module example1 {
    namespace "http://example.com/ns/example1";
    prefix ex1;
    typedef vowels {
        type string {
            pattern "[aeiouy]*";
        }
    }
    grouping "grp1" {
        leaf "void" {
            type "empty";
        }
    }
    container "cont" {
        leaf foo {
            type vowels;
        }
        uses "grp1";
    }
}
```

The hybrid schema generated by the first mapping step will then contain the following two (global) named pattern definitions:

```
<rng:define name="example1__vowels">
  <rng:data type="string">
    <rng:param name="pattern">[aeiouy]*</rng:param>
  </rng:data>
</rng:define>

<rng:define name="_example1__grp1">
  <rng:optional>
    <rng:element name="void">
      <rng:empty/>
    </rng:element>
  </rng:optional>
</rng:define>
```

9.2.1. YANG Refinements and Augments

YANG groupings represent a similar concept as named pattern definitions in RELAX NG, and both languages also offer mechanisms for their subsequent modification. However, in RELAX NG, the definitions themselves are modified, whereas YANG provides two substatements of 'uses', which modify expansions of groupings:

- The 'refine' statement allows for changing parameters of a schema node inside the grouping referenced by the parent 'uses' statement;

- The 'augment' statement can be used for adding new schema nodes to the grouping contents.

Both 'refine' and 'augment' statements are quite powerful in that they can address, using XPath-like expressions as their arguments, schema nodes that are arbitrarily deep inside the grouping contents. In contrast, modifications of named pattern definitions in RELAX NG are applied exclusively at the topmost level of the named pattern contents. In order to achieve a modifiability of named patterns comparable to YANG, a RELAX NG schema would have to be extremely flat (cf. Section 8.3) and very difficult to read.

Since the goal of the mapping described in this document is to generate ad hoc DSDL schemas, we decided to avoid these complications and instead expand the grouping and refine and/or augment it "in place". In other words, every 'uses' statement that has 'refine' and/or 'augment' substatements is replaced by the contents of the corresponding grouping, the changes specified in the 'refine' and 'augment' statements are applied, and the resulting YANG schema fragment is mapped as if the 'uses'/'grouping' indirection wasn't there.

If there are further 'uses' statements inside the grouping contents, they may require expansion, too: it is necessary if the contained 'uses'/'grouping' pair lies on the "modification path" specified in the argument of a 'refine' or 'augment' statement.

EXAMPLE. Consider the following YANG module:

```
module example2 {
    namespace "http://example.com/ns/example2";
    prefix ex2;
    grouping leaves {
        uses fr;
        uses es;
    }
    grouping fr {
        leaf feuille {
            type string;
        }
    }
    grouping es {
        leaf hoja {
            type string;
        }
    }
    uses leaves;
}
```

The resulting hybrid schema contains three global named pattern definitions corresponding to the three groupings, namely:

```
<rng:define name="_example2__leaves">
  <rng:interleave>
    <rng:ref name="_example2__fr"/>
    <rng:ref name="_example2__es"/>
  </rng:interleave>
</rng:define>

<rng:define name="_example2__fr">
  <rng:optional>
    <rng:element name="feuille">
      <rng:data type="string"/>
    </rng:element>
  </rng:optional>
</rng:define>

<rng:define name="_example2__es">
  <rng:optional>
    <rng:element name="hoja">
      <rng:data type="string"/>
    </rng:element>
  </rng:optional>
</rng:define>
```

and the configuration data part of the hybrid schema is a single named pattern reference:

```
<nma:data>
  <rng:ref name="_example2__leaves"/>
</nma:data>
```

Now assume that the "uses leaves" statement contains a 'refine' substatement, for example:

```
uses leaves {
    refine "hoja" {
        default "alamo";
    }
}
```

The resulting hybrid schema now contains just one named pattern definition - "_example2__fr". The other two groupings "leaves" and "es" have to be expanded because they both lie on the "modification path", i.e., contain the leaf "hoja" that is being refined. The configuration data part of the hybrid schema now looks like this:

```
<nma:data>
  <rng:interleave>
    <rng:ref name="_example2__fr"/>
    <rng:optional>
      <rng:element name="ex2:hoja" nma:default="alamo">
        <rng:data type="string"/>
      </rng:element>
    </rng:optional>
  </rng:interleave>
</nma:data>
```

9.2.2. Type Derivation Chains

RELAX NG has no equivalent of the type derivation mechanism in YANG that allows one to restrict a built-in type (perhaps in multiple steps) by adding new constraints. Whenever a derived YANG type is used without restrictions -- as a substatement of either 'leaf' or another 'typedef' -- then the 'type' statement is mapped simply to a named pattern reference <rng:ref>, and the type definition is mapped to a RELAX NG named pattern definition <rng:define>. However, if any restrictions are specified as substatements of the 'type' statement, the type definition MUST be expanded at that point so that only the ancestor built-in type appears in the hybrid schema, restricted with facets that correspond to the combination of all restrictions found along the type derivation chain and also in the 'type' statement.

EXAMPLE. Consider this YANG module:

```
module example3 {
    namespace "http://example.com/ns/example3";
    prefix ex3;
    typedef dozen {
        type uint8 {
            range 1..12;
        }
    }
    leaf month {
        type dozen;
    }
}
```

The 'type' statement in "leaf month" has no restrictions and is therefore mapped simply to the reference <rng:ref name="example3__dozen"/> and the corresponding named pattern is defined as follows:

```
<rng:define name="example3__dozen">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">1</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:define>
```

Assume now that the definition of leaf "month" is changed to:

```
leaf month {
    type dozen {
        range 7..max;
    }
}
```

The output RELAX NG schema then will not contain any named pattern definition and the leaf "month" will be mapped directly to:

```
<rng:element name="ex3:month">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">7</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:element>
```

The mapping of type derivation chains may be further complicated by the presence of the 'default' statement in type definitions. In the simple case, when a type definition containing the 'default'

statement is used without restrictions, the 'default' statement is mapped to the @nma:default attribute attached to the <rng:define> element.

However, if that type definition has to be expanded due to restrictions, the @nma:default attribute arising from the expanded type or ancestor types in the type derivation chain MUST be attached to the pattern where the expansion occurs. If there are multiple 'default' statements in consecutive steps of the type derivation, only the 'default' statement that is closest to the expanded type is used.

EXAMPLE. Consider this variation of the last example:

```
module example3bis {
    namespace "http://example.com/ns/example3bis";
    prefix ex3bis;
    typedef dozen {
        type uint8 {
            range 1..12;
        }
        default 7;
    }
    leaf month {
        type dozen;
    }
}
```

The 'typedef' statement in this module is mapped to the following named pattern definition:

```
<rng:define name="example3bis__dozen" @nma:default="7">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">1</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:define>
```

If the "dozen" type is restricted when used in the leaf "month" definition, as in the previous example, the "dozen" type has to be expanded and @nma:default becomes an attribute of the <ex3bis:month> element definition:

```
<rng:element name="ex3bis:month" @nma:default="7">
  <rng:data type="unsignedByte">
    <rng:param name="minInclusive">7</rng:param>
    <rng:param name="maxInclusive">12</rng:param>
  </rng:data>
</rng:element>
```

However, if the definition of the leaf "month" itself contained the 'default' substatement, the default specified for the "dozen" type would be ignored.

9.3. Translation of XPath Expressions

YANG uses full XPath 1.0 syntax [XPath] for the arguments of 'must', 'when', and 'path' statements. As the names of data nodes defined in a YANG module always belong to the namespace of that YANG module, YANG adopted a simplification similar to the concept of default namespace in XPath 2.0: node names in XPath expressions needn't carry a namespace prefix inside the module where they are defined and the local module's namespace is assumed.

Consequently, all XPath expressions MUST be translated into a fully conformant XPath 1.0 expression: every unprefixed node name MUST be prepended with the local module's namespace prefix as declared by the 'prefix' statement.

XPath expressions appearing inside top-level groupings require special attention because all unprefixed node names contained in them must adopt the namespace of each module where the grouping is used (cf. Section 8.2). In order to achieve this, the local prefix MUST be represented using the variable "$pref" in the hybrid schema. A Schematron schema which encounters such an XPath expression then supplies an appropriate value for this variable via a parameter to an abstract pattern to which the YANG grouping is mapped (see Section 11.2).

For example, XPath expression "/dhcp/max-lease-time" appearing in a YANG module with the "dhcp" prefix will be translated to:

- "$pref:dhcp/$pref:max-lease-time", if the expression is inside a top-level grouping;
- "dhcp:dhcp/dhcp:max-lease-time", otherwise.

YANG also uses other XPath-like expressions, namely key identifiers and "descendant schema node identifiers" (see the ABNF production for and "descendant-schema-nodeid" in Section 12 of [RFC6020]). These expressions MUST be translated by adding local module prefixes as well.

9.4. YANG Language Extensions

YANG allows for extending its own language in-line by adding new statements with keywords from special namespaces. Such extensions first have to be declared using the 'extension' statement, and then they can be used as the standard YANG statements, from which they are distinguished by a namespace prefix qualifying the extension keyword. RELAX NG has a similar extension mechanism -- XML elements and attributes with names from foreign namespaces may be inserted at almost any place of a RELAX NG schema.

YANG language extensions may or may not have a meaning in the context of DSDL schemas. Therefore, an implementation MAY ignore any or all of the extensions. However, an extension that is not ignored MUST be mapped to XML element(s) and/or attribute(s) that exactly match the YIN form of the extension, see Section 11.1 in [RFC6020].

EXAMPLE. Consider the following extension defined by the "acme" module:

```
extension documentation-flag {
    argument number;
}
```

This extension can then be used in the same or another module, for instance like this:

```
leaf folio {
    acme:documentation-flag 42;
    type string;
}
```

If this extension is honored by the mapping, it will be mapped to:

```
<rng:element name="acme:folio">
   <acme:documentation-flag number="42"/>
   <rng:data type="string"/>
</rng:element>
```

Note that the 'extension' statement itself is not mapped in any way.

10. Mapping YANG Statements to the Hybrid Schema

Each subsection in this section is devoted to one YANG statement and provides the specification of how the statement is mapped to the hybrid schema. The subsections are sorted alphabetically by the statement keyword.

Each YANG statement is mapped to an XML fragment, typically a single element or attribute, but it may also be a larger structure. The mapping procedure is inherently recursive, which means that after finishing a statement the mapping continues with its substatements, if there are any, and a certain element of the resulting fragment becomes the parent of other fragments resulting from the mapping of substatements. Any changes to this default recursive procedure are explicitly specified.

YANG XML encoding rules translate to the following rules for ordering multiple subelements:

1. Within the <nma:rpcs> subtree (i.e., for input and output parameters of an RPC operation) the order of subelements is fixed and their definitions in the hybrid schema MUST follow the order specified in the source YANG module.

2. When mapping the 'list' statement, all keys MUST come before any other subelements and in the same order as they are declared in the 'key' statement. The order of the remaining (non-key) subelements is not specified, so their definitions in the hybrid schema MUST be enclosed in the <rng:interleave> element.

3. Otherwise, the order of subelements is arbitrary and, consequently, all definitions of subelements in the hybrid schema MUST be enclosed in the <rng:interleave> element.

The following conventions are used in this section:

- The argument of the statement being mapped is denoted by ARGUMENT.

- The element in the RELAX NG schema that becomes the parent of the resulting XML fragment is denoted by PARENT.

10.1. The 'anyxml' Statement

This statement is mapped to the <rng:element> element and ARGUMENT with prepended local namespace prefix becomes the value of its @name attribute. The contents of <rng:element> are:

```
<rng:ref name="__anyxml__"/>
```

Substatements of the 'anyxml' statement, if any, MAY be mapped to additional children of the <rng:element> element.

If at least one 'anyxml' statement occurs in any of the input YANG modules, the following pattern definition MUST be added exactly once to the RELAX NG schema as a child of the root <rng:grammar> element (cf. [Vli04], p. 172):

```
<rng:define name="__anyxml__">
  <rng:zeroOrMore>
    <rng:choice>
      <rng:attribute>
        <rng:anyName/>
      </rng:attribute>
      <rng:element>
        <rng:anyName/>
        <rng:ref name="__anyxml__"/>
      </rng:element>
      <rng:text/>
    </rng:choice>
  </rng:zeroOrMore>
</rng:define>
```

EXAMPLE: YANG statement in a module with namespace prefix "yam"

```
anyxml data {
    description "Any XML content allowed here.";
}
```

is mapped to the following fragment:

```
<rng:element name="yam:data">
    <a:documentation>Any XML content allowed here</a:documentation>
    <rng:ref name="__anyxml__"/>
</rng:element>
```

An anyxml node is optional if there is no "mandatory true;" substatement. The <rng:element> element then MUST be wrapped in <rng:optional>, except when the 'anyxml' statement is a child of the 'choice' statement and thus forms a shorthand case for that choice (see Section 9.1.1 for details).

10.2. The 'argument' Statement

This statement is not mapped to the output schema, but see the rules for handling extensions in Section 9.4.

10.3. The 'augment' Statement

As a substatement of 'uses', this statement is handled as a part of 'uses' mapping, see Section 10.57.

At the top level of a module or submodule, the 'augment' statement is used for augmenting the schema tree of another YANG module. If the augmented module is not processed within the same mapping session, the top-level 'augment' statement MUST be ignored. Otherwise, the contents of the statement are added to the foreign module with the namespace of the module where the 'augment' statement appears.

10.4. The 'base' Statement

This statement is ignored as a substatement of 'identity' and handled within the 'identityref' type if it appears as a substatement of that type definition, see Section 10.53.6.

10.5. The 'belongs-to' Statement

This statement is not used since the processing of submodules is always initiated from the main module, see Section 10.24.

10.6. The 'bit' Statement

This statement is handled within the "bits" type, see Section 10.53.4.

10.7. The 'case' Statement

This statement is mapped to the <rng:group> or <rng:interleave> element, depending on whether or not the statement belongs to an definition of an RPC operation. If the argument of a sibling 'default' statement equals to ARGUMENT, the @nma:implicit attribute with the value of "true" MUST be added to that <rng:group> or <rng:interleave> element. The @nma:implicit attribute MUST NOT be used for nodes at the top-level of a non-default case (see Section 7.9.3 in [RFC6020]).

10.8. The 'choice' Statement

This statement is mapped to the <rng:choice> element.

If 'choice' has the 'mandatory' substatement with the value of "true", the attribute @nma:mandatory MUST be added to the <rng:choice> element with the value of ARGUMENT. This case may require

additional handling, see Section 11.2.1. Otherwise, if "mandatory true;" is not present, the <rng:choice> element MUST be wrapped in <rng:optional>.

The alternatives in <rng:choice> -- mapped from either the 'case' statement or a shorthand case -- MUST NOT be defined as optional.

10.9. The 'config' Statement

This statement is mapped to the @nma:config attribute, and ARGUMENT becomes its value.

10.10. The 'contact' Statement

This statement SHOULD NOT be used by the mapping since the hybrid schema may be mapped from multiple YANG modules created by different authors. The hybrid schema contains references to all input modules in the Dublin Core elements <dc:source>, see Section 10.34. The original YANG modules are the authoritative sources of the authorship information.

10.11. The 'container' Statement

Using the rules specified in Section 9.1.1, the mapping algorithm MUST determine whether the statement defines an optional container, and if so, insert the <rng:optional> element and make it the new PARENT.

The container defined by this statement is then mapped to the <rng:element> element, which becomes a child of PARENT and uses ARGUMENT with prepended local namespace prefix as the value of its @name attribute.

Finally, using the rules specified in Section 9.1.2, the mapping algorithm MUST determine whether the container is implicit, and if so, add the attribute @nma:implicit with the value of "true" to the <rng:element> element.

10.12. The 'default' Statement

If this statement is a substatement of 'leaf', it is mapped to the @nma:default attribute of PARENT and ARGUMENT becomes its value.

As a substatement of 'typedef', the 'default' statement is also mapped to the @nma:default attribute with the value of ARGUMENT. The placement of this attribute depends on whether or not the type definition has to be expanded when it is used:

- If the type definition is not expanded, @nma:default becomes an attribute of the <rng:define> pattern resulting from the parent 'typedef' mapping.

- Otherwise, @nma:default becomes an attribute of the ancestor RELAX NG pattern inside which the expansion takes place.

Details and an example are given in Section 9.2.2.

Finally, as a substatement of 'choice', the 'default' statement identifies the default case and is handled within the 'case' statement, see Section 10.7. If the default case uses the shorthand notation where the 'case' statement is omitted, the @nma:implicit attribute with the value of "true" is either attached to the node representing the default case in the shorthand notation or, alternatively, an extra <rng:group> element MAY be inserted and the @nma:implicit attribute attached to it. In the latter case, the net result is the same as if the 'case' statement wasn't omitted for the default case.

EXAMPLE. The following 'choice' statement in a module with namespace prefix "yam"

```
choice leaves {
    default feuille;
    leaf feuille { type empty; }
    leaf hoja { type empty; }
}
```

is either mapped directly to:

```
<rng:choice>
  <rng:element name="yam:feuille" nma:implicit="true">
    <rng:empty/>
  </rng:element>
  <rng:element name="yam:hoja">
    <rng:empty/>
  </rng:element/>
</rng:choice>
```

or the default case may be wrapped in an extra <rng:group>:

```
<rng:choice>
  <rng:group nma:implicit="true">
    <rng:element name="yam:feuille">
      <rng:empty/>
    </rng:element>
  </rng:group>
  <rng:element name="yam:hoja">
    <rng:empty/>
  </rng:element/>
</rng:choice>
```

10.13. The 'description' Statement

This statement is mapped to the DTD compatibility element <a:documentation> and ARGUMENT becomes its text.

In order to get properly formatted in the RELAX NG compact syntax, this element SHOULD be inserted as the first child of PARENT.

10.14. The 'deviation' Statement

This statement is ignored. However, it is assumed that all deviations are known beforehand and the corresponding changes have already been applied to the input YANG modules.

10.15. The 'enum' Statement

This statement is mapped to the <rng:value> element, and ARGUMENT becomes its text. All substatements except 'status' are ignored because the <rng:value> element cannot contain annotation elements, see [RNG], Section 6.

10.16. The 'error-app-tag' Statement

This statement is ignored unless it is a substatement of 'must'. In the latter case, it is mapped to the <nma:error-app-tag> element. See also Section 10.35.

10.17. The 'error-message' Statement

This statement is ignored unless it is a substatement of 'must'. In the latter case, it is mapped to the <nma:error-message> element. See also Section 10.35.

10.18. The 'extension' Statement

This statement is ignored. However, extensions to the YANG language MAY be mapped as described in Section 9.4.

10.19. The 'feature' Statement

This statement is ignored.

10.20. The 'grouping' Statement

This statement is mapped to a RELAX NG named pattern definition <rng:define>, but only if the grouping defined by this statement is used without refinements and augments in at least one of the input modules. In this case, the named pattern definition becomes a child of the <rng:grammar> element and its name is ARGUMENT mangled according to the rules specified in Section 9.2.

As explained in Section 8.2, a named pattern definition MUST be placed:

- as a child of the root <rng:grammar> element if the corresponding grouping is defined at the top level of an input YANG module;
- otherwise as a child of the embedded <rng:grammar> element corresponding to the module in which the grouping is defined.

Whenever a grouping is used with refinements and/or augments, it is expanded so that the refinements and augments may be applied in place to the prescribed schema nodes. See Section 9.2.1 for further details and an example.

An implementation MAY offer the option of mapping all 'grouping' statements as named pattern definitions in the output RELAX NG schema even if they are not referenced. This is useful for mapping YANG "library" modules that typically contain only 'typedef' and/or 'grouping' statements.

10.21. The 'identity' Statement

This statement is mapped to the following named pattern definition which is placed as a child of the root <rng:grammar> element:

```
<rng:define name="__PREFIX_ARGUMENT">
  <rng:choice>
    <rng:value type="QName">PREFIX:ARGUMENT</rng:value>
    <rng:ref name="IDENTITY1"/>
    ...
  </rng:choice>
</rng:define>
```

where:

PREFIX is the prefix used in the hybrid schema for the namespace of the module where the current identity is defined.

IDENTITY1 is the name of the named pattern corresponding to an identity that is derived from the current identity. Exactly one <rng:ref> element MUST be present for every such identity.

EXAMPLE ([RFC6020], Section 7.16.3). Consider the following identities defined in two input YANG modules:

```
module crypto-base {
  namespace "http://example.com/crypto-base";
  prefix "crypto";
  identity crypto-alg {
    description
      "Base identity from which all crypto algorithms
       are derived.";
    }
}

module des {
  namespace "http://example.com/des";
  prefix "des";
  import "crypto-base" {
    prefix "crypto";
  }
  identity des {
    base "crypto:crypto-alg";
    description "DES crypto algorithm";
  }
  identity des3 {
    base "crypto:crypto-alg";
    description "Triple DES crypto algorithm";
  }
}
```

The identities will be mapped to the following named pattern definitions:

```
<define name="__crypto_crypto-alg">
  <choice>
    <value type="QName">crypto:crypto-alg</value>
    <ref name="__des_des"/>
    <ref name="__des_des3"/>
  </choice>
</define>
<define name="__des_des">
  <value type="QName">des:des</value>
</define>
<define name="__des_des3">
  <value type="QName">des:des3</value>
</define>
```

10.22. The 'if-feature' Statement

ARGUMENT together with arguments of all sibling 'if-feature' statements (with added prefixes, if missing) MUST be collected in a space-separated list that becomes the value of the @nma:if-feature attribute. This attribute is attached to PARENT.

10.23. The 'import' Statement

This statement is not specifically mapped. The module whose name is in ARGUMENT has to be parsed so that the importing module is able to use its top-level groupings, typedefs and identities, and also augment the data tree of the imported module.

If the 'import' statement has the 'revision' substatement, the corresponding revision of the imported module MUST be used. The mechanism for finding a given module revision is outside the scope of this document.

10.24. The 'include' Statement

This statement is not specifically mapped. The submodule whose name is in ARGUMENT has to be parsed and its contents mapped exactly as if the submodule text appeared directly in the main module text.

If the 'include' statement has the 'revision' substatement, the corresponding revision of the submodule MUST be used. The mechanism for finding a given submodule revision is outside the scope of this document.

10.25. The 'input' Statement

This statement is handled within 'rpc' statement, see Section 10.50.

10.26. The 'key' Statement

This statement is mapped to @nma:key attribute. ARGUMENT MUST be translated so that every key is prefixed with the namespace prefix of the local module. The result of this translation then becomes the value of the @nma:key attribute.

10.27. The 'leaf' Statement

This statement is mapped to the <rng:element> element and ARGUMENT with prepended local namespace prefix becomes the value of its @name attribute.

If the leaf is optional, i.e., if there is no "mandatory true;" substatement and the leaf is not declared among the keys of an enclosing list, then the <rng:element> element MUST be enclosed in <rng:optional>, except when the 'leaf' statement is a child of the 'choice' statement and thus represents a shorthand case for that choice (see Section 9.1.1 for details).

10.28. The 'leaf-list' Statement

This statement is mapped to a block enclosed by either the <rng:zeroOrMore> or the <rng:oneOrMore> element depending on whether the argument of 'min-elements' substatement is "0" or positive, respectively (it is zero by default). This <rng:zeroOrMore> or <rng:oneOrMore> element becomes the PARENT.

<rng:element> is then added as a child element of PARENT and ARGUMENT with prepended local namespace prefix becomes the value of its @name attribute. Another attribute, @nma:leaf-list, MUST also be added to this <rng:element> element with the value of "true". If the 'leaf-list' statement has the 'min-elements' substatement and its argument is greater than one, additional attribute @nma:min-elements is attached to <rng:element> and the argument of 'min-elements' becomes the value of this attribute. Similarly, if there is the 'max-elements' substatement and its argument value is not "unbounded", attribute @nma:max-elements is attached to this element and the argument of 'max-elements' becomes the value of this attribute.

EXAMPLE. Consider the following 'leaf-list' appearing in a module with the namespace prefix "yam":

```
leaf-list foliage {
    min-elements 3;
    max-elements 6378;
    ordered-by user;
    type string;
}
```

It is mapped to the following RELAX NG fragment:

```
<rng:oneOrMore>
  <rng:element name="yam:foliage" nma:leaf-list="true"
               nma:ordered-by="user"
               nma:min-elements="3" nma:max-elements="6378">
    <rng:data type="string"/>
  </rng:element>
</rng:oneOrMore>
```

10.29. The 'length' Statement

This statement is handled within the "string" type, see Section 10.53.10.

10.30. The 'list' Statement

This statement is mapped exactly as the 'leaf-list' statement, see Section 10.28. The only difference is that the @nma:leaf-list annotation either MUST NOT be present or MUST have the value of "false".

When mapping the substatements of 'list', the order of children of the list element MUST be specified so that list keys, if there are any, always appear in the same order as they are defined in the 'key' substatement and before other children, see [RFC6020], Section 7.8.5. In particular, if a list key is defined in a grouping but the list node itself is not a part of the same grouping, and the position of the 'uses' statement would violate the above ordering requirement, the grouping MUST be expanded, i.e., the 'uses' statement replaced by the grouping contents.

For example, consider the following YANG fragment of a module with the prefix "yam":

```
grouping keygrp {
  leaf clef {
    type uint8;
  }
}
list foo {
  key clef;
  leaf bar {
    type string;
  }
  leaf baz {
    type string;
  }
  uses keygrp;
}
```

It is mapped to the following RELAX NG fragment:

```
<rng:zeroOrMore>
  <rng:element name="yam:foo" nma:key="yam:clef">
    <rng:element name="yam:clef">
      <rng:data type="unsignedByte"/>
    </rng:element>
    <rng:interleave>
      <rng:element name="yam:bar">
        <rng:data type="string"/>
      </rng:element>
      <rng:element name="yam:baz">
        <rng:data type="string"/>
      </rng:element>
    </rng:interleave>
  </rng:element>
</rng:zeroOrMore>
```

Note that the "keygrp" grouping is expanded and the definition of "yam:clef" is moved before the <rng:interleave> pattern.

10.31. The 'mandatory' Statement

This statement may appear as a substatement of 'leaf', 'choice', or 'anyxml' statement. If ARGUMENT is "true", the parent data node is mapped as mandatory, see Section 9.1.1.

As a substatement of 'choice', this statement is also mapped to the @nma:mandatory attribute, which is added to PARENT. The value of this attribute is the argument of the parent 'choice' statement.

10.32. The 'max-elements' Statement

This statement is handled within 'leaf-list' or 'list' statements, see Section 10.28.

10.33. The 'min-elements' Statement

This statement is handled within 'leaf-list' or 'list' statements, see Section 10.28.

10.34. The 'module' Statement

This statement is mapped to an embedded <rng:grammar> pattern having the @nma:module attribute with the value of ARGUMENT. In addition, a <dc:source> element SHOULD be created as a child of this <rng:grammar> element and contain ARGUMENT as a metadata reference to the input YANG module. See also Section 10.49.

Substatements of the 'module' statement MUST be mapped so that:

- statements representing configuration/state data are mapped to descendants of the <nma:data> element;
- statements representing the contents of RPC requests or replies are mapped to descendants of the <nma:rpcs> element;
- statements representing the contents of event notifications are mapped to descendants of the <nma:notifications> element.

10.35. The 'must' Statement

This statement is mapped to the <nma:must> element. It has one mandatory attribute @assert (with no namespace) that contains ARGUMENT transformed into a valid XPath expression (see Section 9.3). The <nma:must> element may have other subelements resulting from mapping the 'error-app-tag' and 'error-message' substatements. Other substatements of 'must', i.e., 'description' and 'reference', are ignored.

EXAMPLE. YANG statement in the "dhcp" module

```
must 'current() <= ../max-lease-time' {
    error-message
        "The default-lease-time must be less than max-lease-time";
}
```

is mapped to:

```
<nma:must assert="current()&lt;=../dhcp:max-lease-time">
  <nma:error-message>
    The default-lease-time must be less than max-lease-time
  </nma:error-message>
</nma:must>
```

10.36. The 'namespace' Statement

This statement is mapped simultaneously in two ways:

1. to the @xmlns:PREFIX attribute of the root <rng:grammar> element where PREFIX is the namespace prefix specified by the sibling 'prefix' statement. ARGUMENT becomes the value of this attribute;

2. to the @ns attribute of PARENT, which is an embedded <rng: grammar> pattern. ARGUMENT becomes the value of this attribute.

10.37. The 'notification' Statement

This statement is mapped to the following subtree of the <nma: notifications> element in the hybrid schema (where PREFIX is the prefix of the local YANG module):

```
<nma:notification>
  <rng:element name="PREFIX:ARGUMENT">
    ...
  </rng:element>
</nma:notification>
```

Substatements of 'notification' are mapped under <rng:element name="PREFIX:ARGUMENT">.

10.38. The 'ordered-by' Statement

This statement is mapped to @nma:ordered-by attribute and ARGUMENT becomes the value of this attribute. See Section 10.28 for an example.

10.39. The 'organization' Statement

This statement is ignored by the mapping because the hybrid schema may be mapped from multiple YANG modules authored by different parties. The hybrid schema SHOULD contain references to all input modules in the Dublin Core <dc:source> elements, see Section 10.34.

The original YANG modules are the authoritative sources of the authorship information.

10.40. The 'output' Statement

This statement is handled within the 'rpc' statement, see Section 10.50.

10.41. The 'path' Statement

This statement is handled within the "leafref" type, see Section 10.53.8.

10.42. The 'pattern' Statement

This statement is handled within the "string" type, see Section 10.53.10.

10.43. The 'position' Statement

This statement is ignored.

10.44. The 'prefix' Statement

This statement is handled within the sibling 'namespace' statement, see Section 10.36, or within the parent 'import' statement, see Section 10.23. As a substatement of 'belongs-to' (in submodules), the 'prefix' statement is ignored.

10.45. The 'presence' Statement

This statement influences the mapping of the parent container (Section 10.11): the parent container definition MUST be wrapped in <rng:optional>, regardless of its contents. See also Section 9.1.1.

10.46. The 'range' Statement

This statement is handled within numeric types, see Section 10.53.9.

10.47. The 'reference' Statement

This statement is mapped to <a:documentation> element and its text is set to ARGUMENT prefixed with "See: ".

10.48. The 'require-instance' Statement

This statement is handled within "instance-identifier" type (Section 10.53.7).

10.49. The 'revision' Statement

The mapping uses only the most recent instance of the 'revision' statement, i.e., one with the latest date in ARGUMENT, which specifies the current revision of the input YANG module [RFC6020]. This date SHOULD be recorded, together with the name of the YANG module, in the corresponding Dublin Core <dc:source> element (see Section 10.34), for example in this form:

```
<dc:source>YANG module 'foo', revision 2010-03-02</dc:source>
```

The 'description' substatement of 'revision' is ignored.

10.50. The 'rpc' Statement

This statement is mapped to the following subtree in the RELAX NG schema (where PREFIX is the prefix of the local YANG module):

```
<nma:rpc>
  <nma:input>
    <rng:element name="PREFIX:ARGUMENT">
      ... mapped contents of 'input' ...
    </rng:element>
  </nma:input>
  <nma:output">
    ... mapped contents of 'output' ...
  </nma:output>
</nma:rpc>
```

As indicated in the schema fragment, contents of the 'input' substatement (if any) are mapped under <rng:element name="PREFIX:ARGUMENT">. Similarly, contents of the 'output' substatement are mapped under <nma:output>. If there is no 'output' substatement, the <nma:output> element MUST NOT be present.

The <nma:rpc> element is a child of <nma:rpcs>.

10.51. The 'status' Statement

This statement MAY be ignored. Otherwise, it is mapped to @nma:status attribute and ARGUMENT becomes its value.

10.52. The 'submodule' Statement

This statement is not specifically mapped. Its substatements are mapped as if they appeared directly in the module to which the submodule belongs.

10.53. The 'type' Statement

Most YANG built-in data types have an equivalent in the XSD data type library [XSD-D] as shown in Table 4.

YANG type	XSD type	Meaning
int8	byte	8-bit integer value
int16	short	16-bit integer value
int32	int	32-bit integer value
int64	long	64-bit integer value
uint8	unsignedByte	8-bit unsigned integer value
uint16	unsignedShort	16-bit unsigned integer value
uint32	unsignedInt	32-bit unsigned integer value
uint64	unsignedLong	64-bit unsigned integer value
string	string	character string
binary	base64Binary	binary data in base64 encoding

Table 4: YANG built-in data types with equivalents in the W3C XML Schema Type Library

Two important data types of the XSD data type library -- "dateTime" and "anyURI" -- are not built-in types in YANG but instead are defined as derived types in the standard modules [RFC6021]: "date-and-time" in the "ietf-yang-types" module and "uri" in the "ietf-inet-types" module. However, the formal restrictions in the YANG type definitions are rather weak. Therefore, implementations of the YANG-to-DSDL mapping SHOULD detect these derived types in source YANG modules and map them to "dateType" and "anyURI", respectively.

Details about the mapping of individual YANG built-in types are given in the following subsections.

10.53.1. The "empty" Type

This type is mapped to <rng:empty/>.

10.53.2. The "boolean" Type

This built-in type does not allow any restrictions and is mapped to the following XML fragment:

```
<rng:choice>
  <rng:value>true</rng:value>
  <rng:value>false</rng:value>
</rng:choice>
```

Note that the XSD "boolean" type cannot be used here because it allows, unlike YANG, an alternative numeric representation of boolean values: 0 for "false" and 1 for "true".

10.53.3. The "binary" Type

This built-in type does not allow any restrictions and is mapped simply by inserting an <rng:data> element whose @type attribute value is set to "base64Binary" (see also Table 4).

10.53.4. The "bits" Type

This type is mapped to the <rng:list> and for each 'bit' substatement the following XML fragment is inserted as a child of <rng:list>:

```
<rng:optional>
  <rng:value>bit_name</rng:value>
</rng:optional>
```

where bit_name is the name of the bit as found in the argument of a 'bit' substatement.

10.53.5. The "enumeration" and "union" Types

These types are mapped to the <rng:choice> element.

10.53.6. The "identityref" Type

This type is mapped to the following named pattern reference:

```
<rng:ref name="__PREFIX_BASE"/>
```

where PREFIX:BASE is the qualified name of the identity appearing in the argument of the 'base' substatement.

For example, assume that module "des" in Section 10.21 contains the following leaf definition:

```
leaf foo {
  type identityref {
    base crypto:crypto-alg;
  }
}
```

This leaf would then be mapped to the following element pattern:

```
<element name="des:foo">
  <ref name="__crypto_crypto-alg"/>
</element>
```

10.53.7. The "instance-identifier" Type

This type is mapped to <rng:data> element with @type attribute set to "string". In addition, an empty <nma:instance-identifier> element MUST be inserted as a child of PARENT.

The argument of the 'require-instance' substatement, if it exists, becomes the value of the @require-instance attribute of the <nma: instance-identifier> element.

10.53.8. The "leafref" Type

This type is mapped exactly as the type of the leaf given in the argument of 'path' substatement. However, if the type of the referred leaf defines a default value, this default value MUST be ignored by the mapping.

In addition, @nma:leafref attribute MUST be added to PARENT. The argument of the 'path' substatement, translated according to Section 9.3, is set as the value of this attribute.

10.53.9. The Numeric Types

YANG built-in numeric types are "int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64", and "decimal64". They are mapped to the <rng:data> element with the @type attribute set to ARGUMENT translated according to Table 4 above.

An exception is the "decimal64" type, which is mapped to the "decimal" type of the XSD data type library. Its precision and number of fractional digits are controlled with the following facets, which MUST always be present:

- "totalDigits" facet set to the value of 19.

- "fractionDigits" facet set to the argument of the 'fraction-digits' substatement.

The fixed value of "totalDigits" corresponds to the maximum of 19 decimal digits for 64-bit integers.

For example, the statement:

```
type decimal64 {
    fraction-digits 2;
}
```

is mapped to the following RELAX NG data type:

```
<rng:data type="decimal">
  <rng:param name="totalDigits">19</rng:param>
  <rng:param name="fractionDigits">2</rng:param>
</rng:data>
```

All numeric types support the 'range' restriction, which is mapped as follows:

If the range expression consists of just a single range LO..HI, then it is mapped to a pair of data type facets:

```
      <rng:param name="minInclusive">LO</rng:param>
```

and

```
       <rng:param name="maxInclusive">HI</rng:param>
```

If the range consists of a single number, the values of both facets are set to this value. If LO is equal to the string "min", the "minInclusive" facet is omitted. If HI is equal to the string "max", the "maxInclusive" facet is omitted.

If the range expression has multiple parts separated by "|", then the parent <rng:data> element must be repeated once for every range part and all such <rng:data> elements are wrapped in <rng:choice> element. Each <rng:data> element contains the "minInclusive" and "maxInclusive" facets for one part of the range expression as described in the previous paragraph.

For the "decimal64" type, the "totalDigits" and "fractionDigits" must be repeated inside each of the <rng:data> elements.

For example,

```
type int32 {
    range "-6378..0|42|100..max";
}
```

is mapped to the following RELAX NG fragment:

```
<rng:choice>
  <rng:data type="int">
    <rng:param name="minInclusive">-6378</rng:param>
    <rng:param name="maxInclusive">0</rng:param>
  </rng:data>
  <rng:data type="int">
    <rng:param name="minInclusive">42</rng:param>
    <rng:param name="maxInclusive">42</rng:param>
  </rng:data>
  <rng:data type="int">
    <rng:param name="minInclusive">100</rng:param>
  </rng:data>
</rng:choice>
```

See Section 9.2.2 for further details on mapping the restrictions.

10.53.10. The "string" Type

This type is mapped to the <rng:data> element with the @type attribute set to "string".

The 'length' restriction is handled analogically to the 'range' restriction for the numeric types (Section 10.53.9):

If the length expression has just a single range:

- and if the length range consists of a single number LENGTH, the following data type facet is inserted:

  ```
  <rng:param name="length">LENGTH</rng:param>.
  ```

- if the length range is of the form LO..HI, i.e., it consists of both the lower and upper bound. The following two data type facets are then inserted:

  ```
  <rng:param name="minLength">LO</rng:param>
  ```

  and

  ```
  <rng:param name="maxLength">HI</rng:param>
  ```

If LO is equal to the string "min", the "minLength" facet is omitted. If HI is equal to the string "max", the "maxLength" facet is omitted.

If the length expression has of multiple parts separated by "|", then the parent <rng:data> element must be repeated once for every range part and all such <rng:data> elements are wrapped in <rng:choice> element. Each <rng:data> element contains the "length" or "minLength" and "maxLength" facets for one part of the length expression as described in the previous paragraph.

Every 'pattern' restriction of the "string" data type is mapped to the "pattern" facet:

```
<rng:param name="pattern">...</rng:param>
```

with text equal to the argument of the 'pattern' statement. All such "pattern" facets must be repeated inside each copy of the <rng:data> element, i.e., once for each length range.

For example,

```
type string {
    length "1|3..8";
    pattern "[A-Z][a-z]*";
}
```

is mapped to the following RELAX NG fragment:

```
<rng:choice>
  <rng:data type="string">
    <rng:param name="length">1</rng:param>
    <rng:param name="pattern">[A-Z][a-z]*</rng:param>
  </rng:data>
  <rng:data type="string">
    <rng:param name="minLength">3</rng:param>
    <rng:param name="maxLength">8</rng:param>
    <rng:param name="pattern">[A-Z][a-z]*</rng:param>
  </rng:data>
</rng:choice>
```

10.53.11. Derived Types

If the 'type' statement refers to a derived type, it is mapped in one of the following ways depending on whether it contains any restrictions as its substatements:

1. Without restrictions, the 'type' statement is mapped simply to the <rng:ref> element, i.e., a reference to a named pattern. If the RELAX NG definition of this named pattern has not been added to the hybrid schema yet, the corresponding type definition MUST be found and its mapping installed as a subelement of either the root or an embedded <rng:grammar> element, see Section 10.54. Even if a given derived type is used more than once in the input YANG modules, the mapping of the corresponding 'typedef' MUST be installed only once.

2. If any restrictions are present, the ancestor built-in type for the given derived type must be determined and the mapping of this base type MUST be used. Restrictions appearing at all stages of the type derivation chain MUST be taken into account and their conjunction added to the <rng:data> element that defines the basic type.

See Section 9.2.2 for more details and an example.

10.54. The 'typedef' Statement

This statement is mapped to a RELAX NG named pattern definition <rng:define>, but only if the type defined by this statement is used without restrictions in at least one of the input modules. In this case, the named pattern definition becomes a child of either the root or an embedded <rng:grammar> element, depending on whether or not the 'typedef' statement appears at the top level of a YANG module. The name of this named pattern definition is set to ARGUMENT mangled according to the rules specified in Section 9.2.

Whenever a derived type is used with additional restrictions, the ancestor built-in type for the derived type is used instead with restrictions (facets) that are a combination of all restrictions specified along the type derivation chain. See Section 10.53.11 for further details and an example.

An implementation MAY offer the option of recording all 'typedef' statements as named patterns in the output RELAX NG schema even if they are not referenced. This is useful for mapping YANG "library" modules containing only 'typedef' and/or 'grouping' statements.

10.55. The 'unique' Statement

This statement is mapped to the @nma:unique attribute. ARGUMENT MUST be translated so that every node identifier in each of its components is prefixed with the namespace prefix of the local module, unless the prefix is already present. The result of this translation then becomes the value of the @nma:unique attribute.

For example, assuming that the local module prefix is "ex",

unique "foo ex:bar/baz"

is mapped to the following attribute/value pair:

nma:unique="ex:foo ex:bar/ex:baz"

10.56. The 'units' Statement

This statement is mapped to the @nma:units attribute and ARGUMENT becomes its value.

10.57. The 'uses' Statement

If this statement has neither 'refine' nor 'augment' substatements, it is mapped to the <rng:ref> element, i.e., a reference to a named pattern, and the value of its @name attribute is set to ARGUMENT mangled according to Section 9.2. If the RELAX NG definition of the referenced named pattern has not been added to the hybrid schema yet, the corresponding grouping MUST be found and its mapping installed as a subelement of <rng:grammar>, see Section 10.20.

Otherwise, if the 'uses' statement has any 'refine' or 'augment' substatements, the corresponding grouping must be looked up and its contents inserted under PARENT. See Section 9.2.1 for further details and an example.

10.58. The 'value' Statement

This statement is ignored.

10.59. The 'when' Statement

This statement is mapped to the @nma:when attribute and ARGUMENT, translated according to Section 9.3, becomes it value.

10.60. The 'yang-version' Statement

This statement is not mapped to the output schema. However, an implementation SHOULD check that it is compatible with the YANG version declared by the statement (currently version 1). In the case of a mismatch, the implementation SHOULD report an error and terminate.

10.61. The 'yin-element' Statement

This statement is not mapped to the output schema, but see the rules for extension handling in Section 9.4.

11. Mapping the Hybrid Schema to DSDL

As explained in Section 6, the second step of the YANG-to-DSDL mapping takes the hybrid schema and transforms it to various DSDL schemas capable of validating instance XML documents. As an input parameter, this step takes, in the simplest case, just a specification of the NETCONF XML document type that is to be validated. These document types can be, for example, the contents of a datastore, a reply to <nc:get> or <nc:get-config>, contents of other RPC requests/replies and event notifications, and so on.

The second mapping step has to accomplish the following three general tasks:

1. Extract the parts of the hybrid schema that are appropriate for the requested document type. For example, if a <nc:get> reply is to be validated, the subtree under <nma:data> has to be selected.

2. The schema must be adapted to the specific encapsulating XML elements mandated by the RPC layer. These are, for example, <nc:rpc> and <nc:data> elements in the case of a <nc:get> reply or <en:notification> for a notification.

3. Finally, NETMOD-specific annotations that are relevant for the schema language of the generated schema must be mapped to the corresponding patterns or rules.

These three tasks are together much simpler than the first mapping step and can be effectively implemented using XSL transformations [XSLT].

The following subsections describe the details of the second mapping step for the individual DSDL schema languages. Section 12 then contains a detailed specification for the mapping of all NETMOD-specific annotations.

11.1. Generating RELAX NG Schemas for Various Document Types

With one minor exception, obtaining a validating RELAX NG schema from the hybrid schema only means taking appropriate parts of the hybrid schema and assembling them in a new RELAX NG grammar, perhaps after removing all unwanted annotations.

The structure of the resulting RELAX NG schema is similar to that of the hybrid schema: the root grammar contains embedded grammars, one for each input YANG module. However, as explained in Section 8.2, global named pattern definitions (children of the root <rng:grammar> element) MUST be moved to a separate schema file.

Depending on the XML document type that is the target for validation, such as <nc:get> or <nc:get-config> reply, RPC operations or notifications, patterns defining corresponding top-level information items MUST be added, such as <nc:rpc-reply> with the @message-id attribute and so on.

In order to avoid copying common named pattern definitions for common NETCONF elements and attributes to every single output RELAX NG file, such schema-independent definitions SHOULD be collected in a library file that is then included by the validating RELAX NG schemas. Appendix B has the listing of such a library file.

The minor exception mentioned above is the annotation @nma:config, which must be observed if the target document type is a reply to <nc:get-config>. In this case, each element definition that has this attribute with the value of "false" MUST be removed from the schema together with its descendants. See Section 12.1 for more details.

11.2. Mapping Semantic Constraints to Schematron

Schematron schemas tend to be much flatter and more uniform compared to RELAX NG. They have exactly four levels of XML hierarchy: <sch:schema>, <sch:pattern>, <sch:rule>, and <sch:assert> or <sch:report>.

In a Schematron schema generated by the second mapping step, the basic unit of organization is a rule represented by the <sch:rule> element. The following NETMOD-specific annotations from the hybrid schema (henceforth called "semantic annotations") are mapped to corresponding Schematron rules: <nma:must>, @nma:key, @nma:unique, @nma:max-elements, @nma:min-elements, @nma:when, @nma:leafref, @nma:leaf-list, and also @nma:mandatory appearing as an attribute of <rng:choice> (see Section 11.2.1).

Each input YANG module is mapped to a Schematron pattern whose @id attribute is set to the module name. Every <rng:element> pattern containing at least one of the above-mentioned semantic annotations is then mapped to a Schematron rule:

```
<sch:rule context="XELEM">
  ...
</sch:rule>
```

The value of the mandatory @context attribute of <sch:rule> (denoted as XELEM) MUST be set to the absolute path of the context element in the data tree. The <sch:rule> element contains the mappings of all contained semantic annotations in the form of Schematron asserts or reports.

Semantic annotations appearing inside a named pattern definition (i.e., having <rng:define> among its ancestors) require special treatment because they may be potentially used in different contexts. This is accomplished by using Schematron abstract patterns that use the "$pref" variable in place of the local namespace prefix. The value of the @id attribute of such an abstract pattern MUST be set to the name of the named pattern definition that is being mapped (i.e., the mangled name of the original YANG grouping).

When the abstract pattern is instantiated, the values of the following two parameters MUST be provided:

- pref: the actual namespace prefix,

- start: XPath expression defining the context in which the grouping is used.

EXAMPLE. Consider the following YANG module:

```
module example4 {
  namespace "http://example.com/ns/example4";
  prefix ex4;
  uses sorted-leaf-list;
  grouping sorted-leaf-list {
    leaf-list sorted-entry {
      must "not(preceding-sibling::sorted-entry > .)" {
        error-message "Entries must appear in ascending order.";
      }
      type uint8;
    }
  }
}
```

The resulting Schematron schema for a reply to <nc:get> is then as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:ns uri="http://example.com/ns/example4" prefix="ex4"/>
  <sch:ns uri="urn:ietf:params:xml:ns:netconf:base:1.0"
          prefix="nc"/>
  <sch:pattern abstract="true"
               id="_example4__sorted-leaf-list">
    <sch:rule context="$start/$pref:sorted-entry">
      <sch:report
          test=". = preceding-sibling::$pref:sorted-entry">
        Duplicate leaf-list entry "<sch:value-of select="."/>".
      </sch:report>
      <sch:assert
          test="not(preceding-sibling::$pref:sorted-entry &gt; .)">
        Entries must appear in ascending order.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
  <sch:pattern id="example4"/>
  <sch:pattern id="id2573371" is-a="_example4__sorted-leaf-list">
    <sch:param name="start" value="/nc:rpc-reply/nc:data"/>
    <sch:param name="pref" value="ex4"/>
  </sch:pattern>
</sch:schema>
```

The "sorted-leaf-list" grouping from the input module is mapped to an abstract pattern with an @id value of "_example4__sorted-leaf-list" in which the 'must' statement corresponds to the <sch:assert> element. The abstract pattern is the instantiated by the pattern with an @id value of "id2573371", which sets the "start" and "pref" parameters to appropriate values.

Note that another Schematron element, <sch:report>, was automatically added, checking for duplicate leaf-list entries.

The mapping from the hybrid schema to Schematron proceeds in the following steps:

1. First, the active subtree(s) of the hybrid schema must be selected depending on the requested target document type. This procedure is identical to the RELAX NG case, including the handling of @nma:config if the target document type is <nc:get-config> reply.

2. Namespaces of all input YANG modules, together with the namespaces of base NETCONF ("nc" prefix) or notifications ("en" prefix) MUST be declared using the <sch:ns> element, for example:

```
<sch:ns uri="http://example.com/ns/example4" prefix="ex4"/>
```

3. One pattern is created for every input module. In addition, an abstract pattern is created for every named pattern definition containing one or more semantic annotations.

4. A <sch:rule> element is created for each element pattern containing semantic annotations.

5. Every such annotation is then mapped to an <sch:assert> or <sch:report> element, which is installed as a child of the <sch:rule> element.

11.2.1. Constraints on Mandatory Choice

In order to fully represent the semantics of YANG's 'choice' statement with the "mandatory true;" substatement, the RELAX NG grammar has to be combined with a special Schematron rule.

EXAMPLE. Consider the following module:

```
module example5 {
    namespace "http://example.com/ns/example5";
    prefix ex5;
    choice foobar {
        mandatory true;
        case foo {
            leaf foo1 {
                type uint8;
            }
            leaf foo2 {
                type uint8;
            }
        }
        leaf bar {
            type uint8;
        }
    }
}
```

In this module, all three leaf nodes in both case branches are optional but because of the "mandatory true;" statement, at least one of them must be present in a valid configuration. The 'choice' statement from this module is mapped to the following fragment of the RELAX NG schema:

```
<rng:choice>
  <rng:interleave>
    <rng:optional>
      <rng:element name="ex5:foo1">
        <rng:data type="unsignedByte"/>
      </rng:element>
    </rng:optional>
    <rng:optional>
      <rng:element name="ex5:foo2">
        <rng:data type="unsignedByte"/>
      </rng:element>
    </rng:optional>
  </rng:interleave>
  <rng:element name="ex5:bar">
    <rng:data type="unsignedByte"/>
  </rng:element>
</rng:choice>
```

In the second case branch, the "ex5:bar" element is defined as mandatory so that this element must be present in a valid configuration if this branch is selected. However, the two elements in the first branch "foo" cannot be both declared as mandatory since each of them alone suffices for a valid configuration. As a result, the above RELAX NG fragment would successfully validate configurations where none of the three leaf elements are present.

Therefore, mandatory choices, which can be recognized in the hybrid schema as <rng:choice> elements with the @nma:mandatory annotation, have to be handled in a special way: for each mandatory choice where at least one of the cases contains more than one node, a Schematron rule MUST be added enforcing the presence of at least one element from any of the cases. (RELAX NG schema guarantees that elements from different cases cannot be mixed together, that all mandatory nodes are present, etc.).

For the example module above, the Schematron rule will be as follows:

```
<sch:rule context="/nc:rpc-reply/nc:data">
  <sch:assert test="ex5:foo1 or ex5:foo2 or ex5:bar">
    Node(s) from at least one case of choice "foobar" must exist.
  </sch:assert>
</sch:rule>
```

11.3. Mapping Default Values to DSRL

DSRL is the only component of DSDL that is allowed to change the information set of the validated XML document. While DSRL also has other functions, YANG-to-DSDL mapping uses it only for specifying and applying default contents. For XML instance documents based on YANG data models, insertion of default contents may potentially take place for all implicit nodes identified by the rules in Section 9.1.2.

In DSRL, the default contents of an element are specified using the <dsrl:default-content> element, which is a child of <dsrl:element-map>. Two sibling elements of <dsrl:default-content> determine the context for the application of the default contents, see [DSRL]:

- the <dsrl:parent> element contains an XSLT pattern specifying the parent element; the default contents are applied only if the parent element exists in the instance document;

- the <dsrl:name> contains the XML name of the element that, if missing or empty, is inserted together with the contents of <dsrl:default-content>.

The <dsrl:parent> element is optional in a general DSRL schema but, for the purpose of the YANG-to-DSDL mapping, this element MUST be always present, in order to guarantee a proper application of default contents.

DSRL mapping only deals with <rng:element> patterns in the hybrid schema that define implicit nodes (see Section 9.1.2). Such element patterns are distinguished by having NETMOD-specific annotation attributes @nma:default or @nma:implicit, i.e., either:

```
<rng:element name="ELEM" nma:default="DEFVALUE">
  ...
</rng:element>
```

or

```
<rng:element name="ELEM" nma:implicit="true">
  ...
</rng:element>
```

The former case applies to leaf nodes having the 'default' substatement, but also to leaf nodes that obtain their default value from a typedef, if this typedef is expanded according to the rules in Section 9.2.2 so that the @nma:default annotation is attached directly to the leaf's element pattern.

The latter case is used for all implicit containers (see Section 9.1) and for leafs that obtain the default value from a typedef and don't have the @nma:default annotation.

In the simplest case, both element patterns are mapped to the following DSRL element map:

```
<dsrl:element-map>
  <dsrl:parent>XPARENT</dsrl:parent>
  <dsrl:name>ELEM</dsrl:name>
  <dsrl:default-content>DEFCONT</dsrl:default-content>
</dsrl:element-map>
```

where XPARENT is the absolute XPath of ELEM's parent element in the data tree and DEFCONT is constructed as follows:

- If the implicit node ELEM is a leaf and has the @nma:default attribute, DEFCONT is set to the value of this attribute (denoted above as DEFVALUE).

- If the implicit node ELEM is a leaf and has the @nma:implicit attribute with the value of "true", the default value has to be determined from the @nma:default attribute of the definition of ELEM's type (perhaps recursively) and used in place of DEFCONT in the above DSRL element map. See also Section 9.2.2.

- Otherwise, the implicit node ELEM is a container and DEFCONT is constructed as an XML fragment containing all descendant elements of ELEM that have either the @nma:implicit or the @nma:default attribute.

In addition, when mapping the default case of a choice, it has to be guaranteed that the default contents are not applied if any node from any non-default case is present. This is accomplished by setting <dsrl:parent> in a special way:

```
<dsrl:parent>XPARENT[not (ELEM1|ELEM2|...|ELEMn)]</dsrl:parent>
```

where ELEM1, ELEM2, ... ELEMn are the names of all top-level nodes from all non-default cases. The rest of the element map is exactly as before.

EXAMPLE. Consider the following YANG module:

```
module example6 {
  namespace "http://example.com/ns/example6";
  prefix ex6;
  container outer {
    leaf leaf1 {
      type uint8;
      default 1;
    }
    choice one-or-two {
      default "one";
      container one {
        leaf leaf2 {
          type uint8;
          default 2;
        }
      }
      leaf leaf3 {
        type uint8;
        default 3;
      }
    }
  }
}
```

The DSRL schema generated for the "get-reply" target document type will be:

```
<?xml version="1.0" encoding="utf-8"?>
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
           xmlns:ex6="http://example.com/ns/example6"
           xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data</dsrl:parent>
    <dsrl:name>ex6:outer</dsrl:name>
    <dsrl:default-content>
      <ex6:leaf1>1</ex6:leaf1>
      <ex6:one>
        <ex6:leaf2>2</ex6:leaf2>
      </ex6:one>
    </dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>/nc:rpc-reply/nc:data/ex6:outer</dsrl:parent>
    <dsrl:name>ex6:leaf1</dsrl:name>
    <dsrl:default-content>1</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>
      /nc:rpc-reply/nc:data/ex6:outer[not(ex6:leaf3)]
    </dsrl:parent>
    <dsrl:name>ex6:one</dsrl:name>
    <dsrl:default-content>
      <ex6:leaf2>2</ex6:leaf2>
    </dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:parent>
      /nc:rpc-reply/nc:data/ex6:outer/ex6:one
    </dsrl:parent>
    <dsrl:name>ex6:leaf2</dsrl:name>
    <dsrl:default-content>2</dsrl:default-content>
  </dsrl:element-map>
</dsrl:maps>
```

Note that the default value for "leaf3" defined in the YANG module is ignored because "leaf3" represents a non-default alternative of a choice and as such never becomes an implicit element.

12. Mapping NETMOD-Specific Annotations to DSDL Schema Languages

This section contains the mapping specification for the individual NETMOD-specific annotations. In each case, the result of the mapping must be inserted into an appropriate context of the target DSDL schema as described in Section 11. The context is determined by the element pattern in the hybrid schema to which the annotation is attached. In the rest of this section, CONTELEM will denote the name of this context element properly qualified with its namespace prefix.

12.1. The @nma:config Annotation

If this annotation is present with the value of "false", the following rules MUST be observed for DSDL schemas of <nc:get-config> reply:

- When generating RELAX NG, the contents of the CONTELEM definition MUST be changed to <rng:notAllowed>.
- When generating Schematron or DSRL, the CONTELEM definition and all its descendants in the hybrid schema MUST be ignored.

12.2. The @nma:default Annotation

This annotation is used for generating the DSRL schema as described in Section 11.3.

12.3. The <nma:error-app-tag> Annotation

This annotation currently has no mapping defined.

12.4. The <nma:error-message> Annotation

This annotation is handled within <nma:must>, see Section 12.13.

12.5. The @if-feature Annotation

The information about available features MAY be supplied as an input parameter to an implementation. In this case, the following changes MUST be performed for all features that are considered unavailable:

- When generating RELAX NG, the contents of the CONTELEM definition MUST be changed to <rng:notAllowed>.
- When generating Schematron or DSRL, the CONTELEM definition and all its descendants in the hybrid schema MUST be ignored.

12.6. The @nma:implicit Annotation

This annotation is used for generating the DSRL schema as described in Section 11.3.

12.7. The <nma:instance-identifier> Annotation

If this annotation element has the @require-instance attribute with the value of "false", it is ignored. Otherwise, it is mapped to the following Schematron assert:

```
<sch:assert test="nmf:evaluate(.)">
  The element pointed to by "CONTELEM" must exist.
</sch:assert>
```

The nmf:evaluate() function is an XSLT extension function (see Extension Functions in [XSLT]) that evaluates an XPath expression at run time. Such an extension function is available in Extended XSLT (EXSLT) or provided as a proprietary extension by some XSLT processors, for example Saxon.

12.8. The @nma:key Annotation

Assume this annotation attribute contains "k_1 k_2 ... k_n", i.e., specifies n children of CONTELEM as list keys. The annotation is then mapped to the following Schematron report:

```
<sch:report test="CONDITION">
  Duplicate key of list "CONTELEM"
</sch:report>
```

where CONDITION has this form:
preceding-sibling::CONTELEM[C_1 and C_2 and ... and C_n]

Each sub-expression C_i, for i=1,2,...,n, specifies the condition for violated uniqueness of the key k_i, namely

k_i=current()/k_i

12.9. The @nma:leaf-list Annotation

This annotation is mapped to the following Schematron rule, which detects duplicate entries of a leaf-list:

```
<sch:report
    test=". = preceding-sibling::PREFIX:sorted-entry">
  Duplicate leaf-list entry "<sch:value-of select="."/>".
</sch:report>
```

See Section 11.2 for a complete example.

12.10. The @nma:leafref Annotation

This annotation is mapped to the following assert:

```
<sch:assert test="PATH=.">
  Leaf "PATH" does not exist for leafref value "VALUE"
</sch:assert>
```

where PATH is the value of @nma:leafref and VALUE is the value of the context element in the instance document for which the referred leaf doesn't exist.

12.11. The @nma:min-elements Annotation

This annotation is mapped to the following Schematron assert:

```
<sch:assert test="count(../CONTELEM)&gt;=MIN">
  List "CONTELEM" - item count must be at least MIN
</sch:assert>
```

where MIN is the value of @nma:min-elements.

12.12. The @nma:max-elements Annotation

This annotation is mapped to the following Schematron assert:

```
<sch:assert
    test="count(../CONTELEM)&lt;=MAX or preceding-sibling::../CONTELEM">
  Number of list items must be at most MAX
</sch:assert>
```

where MAX is the value of @nma:min-elements.

12.13. The <nma:must> Annotation

This annotation is mapped to the following Schematron assert:

```
<sch:assert test="EXPRESSION">
  MESSAGE
</sch:assert>
```

where EXPRESSION is the value of the mandatory @assert attribute of <nma:must>. If the <nma:error-message> subelement exists, MESSAGE is set to its contents; otherwise, it is set to the default message "Condition EXPRESSION must be true".

12.14. The <nma:ordered-by> Annotation

This annotation currently has no mapping defined.

12.15. The <nma:status> Annotation

This annotation currently has no mapping defined.

12.16. The @nma:unique Annotation

The mapping of this annotation is almost identical as for @nma:key, see Section 12.8, with two small differences:

- The value of @nma:unique is a list of descendant schema node identifiers rather than simple leaf names. However, the XPath expressions specified in Section 12.8 work without any modifications if the descendant schema node identifiers are substituted for k_1, k_2, ..., k_n.

- The message appearing as the text of <sch:report> is different: "Violated uniqueness for list CONTELEM".

12.17. The @nma:when Annotation

This annotation is mapped to the following Schematron assert:

```
<sch:assert test="EXPRESSION">
  Node "CONTELEM" is only valid when "EXPRESSION" is true.
</sch:assert>
```

where EXPRESSION is the value of @nma:when.

13. IANA Considerations

This document requests the following two registrations of namespace URIs in the IETF XML registry [RFC3688]:

```
-------------------------------------------------------
URI: urn:ietf:params:xml:ns:netmod:dsdl-annotations:1

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.
-------------------------------------------------------

-------------------------------------------------------
URI: urn:ietf:params:xml:ns:netmod:xpath-extensions:1

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.
-------------------------------------------------------
```

14. Security Considerations

This document defines a procedure that maps data models expressed in the YANG language to a coordinated set of DSDL schemas. The procedure itself has no security impact on the Internet.

DSDL schemas obtained by the mapping procedure may be used for validating the contents of NETCONF messages or entire datastores, and thus they provide additional validity checks above those performed by NETCONF server and client implementations supporting YANG data models. The strictness of this validation is directly derived from the source YANG modules that the validated XML data adhere to.

15. Contributors

The following people contributed significantly to the initial version of this document:

- Rohan Mahy
- Sharon Chisholm (Ciena)

16. Acknowledgments

The editor wishes to thank the following individuals who provided helpful suggestions and/or comments on various versions of this document: Andy Bierman, Martin Bjorklund, Jirka Kosek, Juergen Schoenwaelder, and Phil Shafer.

17. References

17.1. Normative References

[DSDL] ISO/IEC, "Document Schema Definition Languages (DSDL) - Part 1: Overview", ISO/IEC 19757-1, November 2004.

[DSRL] ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 8: Document Semantics Renaming Language - DSRL", ISO/IEC 19757-8:2008(E), December 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, October 2010.

[RNG] ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. Second Edition.", ISO/IEC 19757-2:2008(E), December 2008.

[RNG-CS] ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 2: Regular-Grammar-Based Validation - RELAX NG. AMENDMENT 1: Compact Syntax", ISO/IEC 19757-2:2003/Amd. 1:2006(E), January 2006.

[RNG-DTD] Clark, J., Ed. and M. Murata, Ed., "RELAX NG DTD Compatibility", OASIS Committee Specification, 3 December 2001.

[Schematron] ISO/IEC, "Information Technology - Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron", ISO/IEC 19757-3:2006(E), June 2006.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2006/REC-xml-20060816>.

[XML-INFOSET] Tobin, R. and J. Cowan, "XML Information Set (Second Edition)", World Wide Web Consortium Recommendation REC-xml-infoset-20040204, February 2004, <http://www.w3.org/TR/2004/REC-xml-infoset-20040204>.

[XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

[XSD-D] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[XSLT] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999.

17.2. Informative References

[DHCPtut] Bjorklund, M., "DHCP Tutorial", November 2007, <http://www.yang-central.org/twiki/bin/view/Main/DhcpTutorial>.

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC5013] Kunze, J., "The Dublin Core Metadata Element Set", RFC 5013, August 2007.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[Trang] Clark, J., "Trang: Multiformat schema converter based on RELAX NG", 2008, <http://www.thaiopensource.com/relaxng/trang.html>.

[Vli04] van der Vlist, E., "RELAX NG", O'Reilly, 2004.

[XSD] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[pyang] Bjorklund, M. and L. Lhotka, "pyang: An extensible YANG validator and converter in Python", 2010, <http://code.google.com/p/pyang/>.

Appendix A. RELAX NG Schema for NETMOD-Specific Annotations

This appendix defines the content model for all NETMOD-specific annotations in the form of RELAX NG named pattern definitions.

<CODE BEGINS> file "nmannot.rng"

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <define name="config-attribute">
    <attribute name="nma:config">
      <data type="boolean"/>
    </attribute>
  </define>

  <define name="data-element">
    <element name="nma:data">
      <ref name="__anyxml__"/>
    </element>
  </define>

  <define name="default-attribute">
    <attribute name="nma:default">
      <data type="string"/>
    </attribute>
  </define>

  <define name="error-app-tag-element">
    <element name="nma:error-app-tag">
      <text/>
    </element>
  </define>

  <define name="error-message-element">
    <element name="nma:error-message">
      <text/>
    </element>
  </define>
```

```
  <define name="if-feature-attribute">
    <attribute name="nma:if-feature">
      <list>
        <data type="QName"/>
      </list>
    </attribute>
  </define>

  <define name="implicit-attribute">
    <attribute name="nma:implicit">
      <data type="boolean"/>
    </attribute>
  </define>

  <define name="instance-identifier-element">
    <element name="nma:instance-identifier">
      <optional>
        <attribute name="nma:require-instance">
          <data type="boolean"/>
        </attribute>
      </optional>
    </element>
  </define>

  <define name="key-attribute">
    <attribute name="nma:key">
      <list>
        <data type="QName"/>
      </list>
    </attribute>
  </define>

  <define name="leaf-list-attribute">
    <attribute name="nma:leaf-list">
      <data type="boolean"/>
    </attribute>
  </define>

  <define name="leafref-attribute">
    <attribute name="nma:leafref">
      <data type="string"/>
    </attribute>
  </define>
```

```
  <define name="mandatory-attribute">
    <attribute name="nma:mandatory">
      <data type="Name"/>
    </attribute>
  </define>

  <define name="max-elements-attribute">
    <attribute name="nma:max-elements">
      <data type="nonNegativeInteger"/>
    </attribute>
  </define>

  <define name="min-elements-attribute">
    <attribute name="nma:min-elements">
      <data type="nonNegativeInteger"/>
    </attribute>
  </define>

  <define name="module-attribute">
    <attribute name="nma:module">
      <data type="Name"/>
    </attribute>
  </define>

  <define name="must-element">
    <element name="nma:must">
      <attribute name="assert">
        <data type="string"/>
      </attribute>
      <interleave>
        <optional>
          <ref name="error-app-tag-element"/>
        </optional>
        <optional>
          <ref name="error-message-element"/>
        </optional>
      </interleave>
    </element>
  </define>
```

```
  <define name="notifications-element">
    <element name="nma:notifications">
      <zeroOrMore>
        <element name="nma:notification">
          <ref name="__anyxml__"/>
        </element>
      </zeroOrMore>
    </element>
  </define>

  <define name="rpcs-element">
    <element name="nma:rpcs">
      <zeroOrMore>
        <element name="nma:rpc">
          <interleave>
            <element name="nma:input">
              <ref name="__anyxml__"/>
            </element>
            <optional>
              <element name="nma:output">
                <ref name="__anyxml__"/>
              </element>
            </optional>
          </interleave>
        </element>
      </zeroOrMore>
    </element>
  </define>

  <define name="ordered-by-attribute">
    <attribute name="nma:ordered-by">
      <choice>
        <value>user</value>
        <value>system</value>
      </choice>
    </attribute>
  </define>
```

```
  <define name="status-attribute">
    <optional>
      <attribute name="nma:status">
        <choice>
          <value>current</value>
          <value>deprecated</value>
          <value>obsolete</value>
        </choice>
      </attribute>
    </optional>
  </define>

  <define name="unique-attribute">
    <optional>
      <attribute name="nma:unique">
        <list>
          <data type="token"/>
        </list>
      </attribute>
    </optional>
  </define>

  <define name="units-attribute">
    <optional>
      <attribute name="nma:units">
        <data type="string"/>
      </attribute>
    </optional>
  </define>

  <define name="when-attribute">
    <optional>
      <attribute name="nma:when">
        <data type="string"/>
      </attribute>
    </optional>
  </define>
```

```
  <define name="__anyxml__">
    <zeroOrMore>
      <choice>
        <attribute>
          <anyName/>
        </attribute>
        <element>
          <anyName/>
          <ref name="__anyxml__"/>
        </element>
        <text/>
      </choice>
    </zeroOrMore>
  </define>

  </grammar>

  <CODE ENDS>
```

Appendix B. Schema-Independent Library

In order to avoid copying the common named pattern definitions to every RELAX NG schema generated in the second mapping step, the definitions are collected in a library file -- schema-independent library -- which is included by the validating schemas under the file name "relaxng-lib.rng" (XML syntax) and "relaxng-lib.rnc" (compact syntax). The included definitions cover patterns for common elements from base NETCONF [RFC4741] and event notifications [RFC5277].

```
  <CODE BEGINS> file "relaxng-lib.rng"

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

  <grammar xmlns="http://relaxng.org/ns/structure/1.0"
           xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:en="urn:ietf:params:xml:ns:netconf:notification:1.0"
           datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

    <define name="message-id-attribute">
      <attribute name="message-id">
        <data type="string">
          <param name="maxLength">4095</param>
        </data>
      </attribute>
    </define>
```

```
    <define name="ok-element">
      <element name="nc:ok">
        <empty/>
      </element>
    </define>

    <define name="eventTime-element">
      <element name="en:eventTime">
        <data type="dateTime"/>
      </element>
    </define>
  </grammar>

  <CODE ENDS>
```

Appendix C. Mapping DHCP Data Model - A Complete Example

This appendix demonstrates both steps of the YANG-to-DSDL mapping applied to the "canonical" DHCP tutorial [DHCPtut] data model. The input YANG module is shown in Appendix C.1 and the output schemas in the following two subsections.

The hybrid schema was obtained by the "dsdl" plugin of the pyang tool [pyang] and the validating DSDL schemas were obtained by XSLT stylesheets that are also part of pyang distribution.

Due to the limit of 72 characters per line, a few long strings required manual editing, in particular the regular expression patterns for IP addresses, etc. These were replaced by the placeholder string "... regex pattern ...". Also, line breaks were added to several documentation strings and Schematron messages.

Other than that, the results of the automatic translations were not changed.

C.1. Input YANG Module

```
module dhcp {
  namespace "http://example.com/ns/dhcp";
  prefix dhcp;

  import ietf-yang-types { prefix yang; }
  import ietf-inet-types { prefix inet; }
```

```
     organization
       "yang-central.org";
     description
       "Partial data model for DHCP, based on the config of
        the ISC DHCP reference implementation.";

     container dhcp {
       description
         "configuration and operational parameters for a DHCP server.";

       leaf max-lease-time {
         type uint32;
         units seconds;
         default 7200;
       }

       leaf default-lease-time {
         type uint32;
         units seconds;
         must '. <= ../max-lease-time' {
           error-message
             "The default-lease-time must be less than max-lease-time";
         }
         default 600;
       }

       uses subnet-list;

       container shared-networks {
         list shared-network {
           key name;

           leaf name {
             type string;
           }
           uses subnet-list;
         }
       }

       container status {
         config false;
         list leases {
           key address;
```

```
        leaf address {
          type inet:ip-address;
        }
        leaf starts {
          type yang:date-and-time;
        }
        leaf ends {
          type yang:date-and-time;
        }
        container hardware {
          leaf type {
            type enumeration {
              enum "ethernet";
              enum "token-ring";
              enum "fddi";
            }
          }
          leaf address {
            type yang:phys-address;
          }
        }
      }
    }
  }

  grouping subnet-list {
    description "A reusable list of subnets";
    list subnet {
      key net;
      leaf net {
        type inet:ip-prefix;
      }
      container range {
        presence "enables dynamic address assignment";
        leaf dynamic-bootp {
          type empty;
          description
            "Allows BOOTP clients to get addresses in this range";
        }
```

```
          leaf low {
            type inet:ip-address;
            mandatory true;
          }
          leaf high {
            type inet:ip-address;
            mandatory true;
          }
        }

        container dhcp-options {
          description "Options in the DHCP protocol";
          leaf-list router {
            type inet:host;
            ordered-by user;
            reference "RFC 2132, sec. 3.8";
          }
          leaf domain-name {
            type inet:domain-name;
            reference "RFC 2132, sec. 3.17";
          }
        }

        leaf max-lease-time {
          type uint32;
          units seconds;
          default 7200;
        }
      }
    }
  }
```

C.2. Hybrid Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
    xmlns="http://relaxng.org/ns/structure/1.0"
    xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
    xmlns:dc="http://purl.org/dc/terms"
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    xmlns:dhcp="http://example.com/ns/dhcp"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
 <dc:creator>Pyang 1.0a, DSDL plugin</dc:creator>
 <dc:date>2010-06-17</dc:date>
 <start>
  <grammar nma:module="dhcp" ns="http://example.com/ns/dhcp">
   <dc:source>YANG module 'dhcp'</dc:source>
   <start>
```

```
      <nma:data>
       <optional>
        <element nma:implicit="true" name="dhcp:dhcp">
         <interleave>
          <a:documentation>
           configuration and operational parameters for a DHCP server.
          </a:documentation>
          <optional>
           <element nma:default="7200"
                    name="dhcp:max-lease-time"
                    nma:units="seconds">
            <data type="unsignedInt"/>
           </element>
          </optional>
          <optional>
           <element nma:default="600"
                    name="dhcp:default-lease-time"
                    nma:units="seconds">
            <data type="unsignedInt"/>
            <nma:must assert=". &lt;= ../dhcp:max-lease-time">
             <nma:error-message>
              The default-lease-time must be less than max-lease-time
             </nma:error-message>
            </nma:must>
           </element>
          </optional>
          <ref name="_dhcp__subnet-list"/>
          <optional>
           <element name="dhcp:shared-networks">
            <zeroOrMore>
             <element nma:key="dhcp:name"
                      name="dhcp:shared-network">
              <element name="dhcp:name">
               <data type="string"/>
              </element>
              <ref name="_dhcp__subnet-list"/>
             </element>
            </zeroOrMore>
           </element>
          </optional>
          <optional>
           <element name="dhcp:status" nma:config="false">
            <zeroOrMore>
             <element nma:key="dhcp:address"
                      name="dhcp:leases">
              <element name="dhcp:address">
               <ref name="ietf-inet-types__ip-address"/>
              </element>
```

```
              <interleave>
               <optional>
                <element name="dhcp:starts">
                 <ref name="ietf-yang-types__date-and-time"/>
                </element>
               </optional>
               <optional>
                <element name="dhcp:ends">
                 <ref name="ietf-yang-types__date-and-time"/>
                </element>
               </optional>
               <optional>
                <element name="dhcp:hardware">
                 <interleave>
                  <optional>
                   <element name="dhcp:type">
                    <choice>
                     <value>ethernet</value>
                     <value>token-ring</value>
                     <value>fddi</value>
                    </choice>
                   </element>
                  </optional>
                  <optional>
                   <element name="dhcp:address">
                    <ref name="ietf-yang-types__phys-address"/>
                   </element>
                  </optional>
                 </interleave>
                </element>
               </optional>
              </interleave>
             </element>
            </zeroOrMore>
           </element>
          </optional>
         </interleave>
        </element>
       </optional>
      </nma:data>
      <nma:rpcs/>
      <nma:notifications/>
     </start>
    </grammar>
   </start>
   <define name="ietf-yang-types__phys-address">
    <data type="string">
     <param name="pattern">([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?</param>
```

```
     </data>
    </define>
    <define name="ietf-inet-types__ipv6-address">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ip-prefix">
     <choice>
      <ref name="ietf-inet-types__ipv4-prefix"/>
      <ref name="ietf-inet-types__ipv6-prefix"/>
     </choice>
    </define>
    <define name="ietf-inet-types__host">
     <choice>
      <ref name="ietf-inet-types__ip-address"/>
      <ref name="ietf-inet-types__domain-name"/>
     </choice>
    </define>
    <define name="ietf-yang-types__date-and-time">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="_dhcp__subnet-list">
     <a:documentation>A reusable list of subnets</a:documentation>
     <zeroOrMore>
      <element nma:key="net" name="subnet">
       <element name="net">
        <ref name="ietf-inet-types__ip-prefix"/>
       </element>
       <interleave>
        <optional>
         <element name="range">
          <interleave>
           <optional>
            <element name="dynamic-bootp">
             <a:documentation>
              Allows BOOTP clients to get addresses in this range
             </a:documentation>
             <empty/>
            </element>
           </optional>
           <element name="low">
            <ref name="ietf-inet-types__ip-address"/>
           </element>
           <element name="high">
```

```
          <ref name="ietf-inet-types__ip-address"/>
         </element>
        </interleave>
       </element>
      </optional>
      <optional>
       <element name="dhcp-options">
        <interleave>
         <a:documentation>
          Options in the DHCP protocol
         </a:documentation>
         <zeroOrMore>
          <element nma:leaf-list="true" name="router"
                   nma:ordered-by="user">
           <a:documentation>
            See: RFC 2132, sec. 3.8
           </a:documentation>
           <ref name="ietf-inet-types__host"/>
          </element>
         </zeroOrMore>
         <optional>
          <element name="domain-name">
           <a:documentation>
            See: RFC 2132, sec. 3.17
           </a:documentation>
           <ref name="ietf-inet-types__domain-name"/>
          </element>
         </optional>
        </interleave>
       </element>
      </optional>
      <optional>
       <element nma:default="7200" name="max-lease-time"
                nma:units="seconds">
        <data type="unsignedInt"/>
       </element>
      </optional>
     </interleave>
    </element>
   </zeroOrMore>
  </define>
  <define name="ietf-inet-types__domain-name">
   <data type="string">
    <param name="pattern">... regex pattern ...</param>
    <param name="minLength">1</param>
    <param name="maxLength">253</param>
   </data>
  </define>
```

```
    <define name="ietf-inet-types__ipv4-prefix">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ipv4-address">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ipv6-prefix">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ip-address">
     <choice>
      <ref name="ietf-inet-types__ipv4-address"/>
      <ref name="ietf-inet-types__ipv6-address"/>
     </choice>
    </define>
   </grammar>
```

C.3. Final DSDL Schemas

This appendix contains DSDL schemas that were obtained from the hybrid schema in Appendix C.2 by XSL transformations. These schemas can be directly used for validating a reply to unfiltered <nc:get> with the contents corresponding to the DHCP data model.

The RELAX NG schema (in two parts, as explained in Section 8.2) also includes the schema-independent library from Appendix B.

C.3.1. Main RELAX NG Schema for <nc:get> Reply

```
   <?xml version="1.0" encoding="utf-8"?>
   <grammar
       xmlns="http://relaxng.org/ns/structure/1.0"
       xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
       xmlns:dhcp="http://example.com/ns/dhcp"
       datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
       ns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <include href="relaxng-lib.rng"/>
    <start>
     <element name="rpc-reply">
      <ref name="message-id-attribute"/>
      <element name="data">
```

```
      <interleave>
       <grammar ns="http://example.com/ns/dhcp">
        <include href="dhcp-gdefs.rng"/>
        <start>
         <optional>
          <element name="dhcp:dhcp">
           <interleave>
            <optional>
             <element name="dhcp:max-lease-time">
              <data type="unsignedInt"/>
             </element>
            </optional>
            <optional>
             <element name="dhcp:default-lease-time">
              <data type="unsignedInt"/>
             </element>
            </optional>
            <ref name="_dhcp__subnet-list"/>
            <optional>
             <element name="dhcp:shared-networks">
              <zeroOrMore>
               <element name="dhcp:shared-network">
                <element name="dhcp:name">
                 <data type="string"/>
                </element>
                <ref name="_dhcp__subnet-list"/>
               </element>
              </zeroOrMore>
             </element>
            </optional>
            <optional>
             <element name="dhcp:status">
              <zeroOrMore>
               <element name="dhcp:leases">
                <element name="dhcp:address">
                 <ref name="ietf-inet-types__ip-address"/>
                </element>
                <interleave>
                 <optional>
                  <element name="dhcp:starts">
                   <ref name="ietf-yang-types__date-and-time"/>
                  </element>
                 </optional>
                 <optional>
                  <element name="dhcp:ends">
                   <ref name="ietf-yang-types__date-and-time"/>
                  </element>
                 </optional>
```

```
                  <optional>
                   <element name="dhcp:hardware">
                    <interleave>
                     <optional>
                      <element name="dhcp:type">
                       <choice>
                        <value>ethernet</value>
                        <value>token-ring</value>
                        <value>fddi</value>
                       </choice>
                      </element>
                     </optional>
                     <optional>
                      <element name="dhcp:address">
                       <ref name="ietf-yang-types__phys-address"/>
                      </element>
                     </optional>
                    </interleave>
                   </element>
                  </optional>
                 </interleave>
                </element>
               </zeroOrMore>
              </element>
             </optional>
            </interleave>
           </element>
          </optional>
         </start>
        </grammar>
       </interleave>
      </element>
     </element>
    </start>
   </grammar>
```

C.3.2. RELAX NG Schema - Global Named Pattern Definitions

```
   <?xml version="1.0" encoding="utf-8"?>
   <grammar
       xmlns="http://relaxng.org/ns/structure/1.0"
       xmlns:nma="urn:ietf:params:xml:ns:netmod:dsdl-annotations:1"
       xmlns:dhcp="http://example.com/ns/dhcp"
       datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
    <define name="ietf-yang-types__phys-address">
     <data type="string">
      <param name="pattern">
       ([0-9a-fA-F]{2}(:[0-9a-fA-F]{2})*)?
```

```
      </param>
     </data>
    </define>
    <define name="ietf-inet-types__ipv6-address">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ip-prefix">
     <choice>
      <ref name="ietf-inet-types__ipv4-prefix"/>
      <ref name="ietf-inet-types__ipv6-prefix"/>
     </choice>
    </define>
    <define name="ietf-inet-types__host">
     <choice>
      <ref name="ietf-inet-types__ip-address"/>
      <ref name="ietf-inet-types__domain-name"/>
     </choice>
    </define>
    <define name="ietf-yang-types__date-and-time">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="_dhcp__subnet-list">
     <zeroOrMore>
      <element name="subnet">
       <element name="net">
        <ref name="ietf-inet-types__ip-prefix"/>
       </element>
       <interleave>
        <optional>
         <element name="range">
          <interleave>
           <optional>
            <element name="dynamic-bootp">
             <empty/>
            </element>
           </optional>
           <element name="low">
            <ref name="ietf-inet-types__ip-address"/>
           </element>
           <element name="high">
            <ref name="ietf-inet-types__ip-address"/>
           </element>
          </interleave>
         </element>
```

```
        </optional>
        <optional>
         <element name="dhcp-options">
          <interleave>
           <zeroOrMore>
            <element name="router">
             <ref name="ietf-inet-types__host"/>
            </element>
           </zeroOrMore>
           <optional>
            <element name="domain-name">
             <ref name="ietf-inet-types__domain-name"/>
            </element>
           </optional>
          </interleave>
         </element>
        </optional>
        <optional>
         <element name="max-lease-time">
          <data type="unsignedInt"/>
         </element>
        </optional>
       </interleave>
      </element>
     </zeroOrMore>
    </define>
    <define name="ietf-inet-types__domain-name">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
      <param name="minLength">1</param>
      <param name="maxLength">253</param>
     </data>
    </define>
    <define name="ietf-inet-types__ipv4-prefix">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ipv4-address">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
     </data>
    </define>
    <define name="ietf-inet-types__ipv6-prefix">
     <data type="string">
      <param name="pattern">... regex pattern ...</param>
      <param name="pattern">... regex pattern ...</param>
     </data>
```

```
    </define>
    <define name="ietf-inet-types__ip-address">
     <choice>
      <ref name="ietf-inet-types__ipv4-address"/>
      <ref name="ietf-inet-types__ipv6-address"/>
     </choice>
    </define>
   </grammar>
```

C.3.3. Schematron Schema for <nc:get> Reply

```
  <?xml version="1.0" encoding="utf-8"?>
  <sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
   <sch:ns uri="http://example.com/ns/dhcp" prefix="dhcp"/>
   <sch:ns uri="urn:ietf:params:xml:ns:netconf:base:1.0" prefix="nc"/>
   <sch:pattern abstract="true" id="_dhcp__subnet-list">
    <sch:rule context="$start/$pref:subnet">
     <sch:report test="preceding-sibling::$pref:subnet
                       [$pref:net=current()/$pref:net]">
      Duplicate key "net"
     </sch:report>
    </sch:rule>
    <sch:rule
      context="$start/$pref:subnet/$pref:dhcp-options/$pref:router">
     <sch:report test=".=preceding-sibling::router">
      Duplicate leaf-list value "<sch:value-of select="."/>"
     </sch:report>
    </sch:rule>
   </sch:pattern>
   <sch:pattern id="dhcp">
    <sch:rule
      context="/nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:default-lease-time">
     <sch:assert test=". &lt;= ../dhcp:max-lease-time">
      The default-lease-time must be less than max-lease-time
     </sch:assert>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:shared-networks/dhcp:shared-network">
     <sch:report test="preceding-sibling::dhcp:shared-network
                       [dhcp:name=current()/dhcp:name]">
      Duplicate key "dhcp:name"
     </sch:report>
    </sch:rule>
    <sch:rule context="/nc:rpc-reply/nc:data/dhcp:dhcp/
                       dhcp:status/dhcp:leases">
     <sch:report test="preceding-sibling::dhcp:leases
                       [dhcp:address=current()/dhcp:address]">
      Duplicate key "dhcp:address"
```

```
     </sch:report>
    </sch:rule>
   </sch:pattern>
   <sch:pattern id="id2768196" is-a="_dhcp__subnet-list">
    <sch:param name="start" value="/nc:rpc-reply/nc:data/dhcp:dhcp"/>
    <sch:param name="pref" value="dhcp"/>
   </sch:pattern>
   <sch:pattern id="id2768215" is-a="_dhcp__subnet-list">
    <sch:param name="start"
               value="/nc:rpc-reply/nc:data/dhcp:dhcp/
                      dhcp:shared-networks/dhcp:shared-network"/>
    <sch:param name="pref" value="dhcp"/>
   </sch:pattern>
  </sch:schema>
```

C.3.4. DSRL Schema for <nc:get> Reply

```
   <?xml version="1.0" encoding="utf-8"?>
   <dsrl:maps
       xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
       xmlns:dhcp="http://example.com/ns/dhcp"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <dsrl:element-map>
     <dsrl:parent>/nc:rpc-reply/nc:data</dsrl:parent>
     <dsrl:name>dhcp:dhcp</dsrl:name>
     <dsrl:default-content>
      <dhcp:max-lease-time>7200</dhcp:max-lease-time>
      <dhcp:default-lease-time>600</dhcp:default-lease-time>
     </dsrl:default-content>
    </dsrl:element-map>
    <dsrl:element-map>
     <dsrl:parent>/nc:rpc-reply/nc:data/dhcp:dhcp</dsrl:parent>
     <dsrl:name>dhcp:max-lease-time</dsrl:name>
     <dsrl:default-content>7200</dsrl:default-content>
    </dsrl:element-map>
    <dsrl:element-map>
     <dsrl:parent>/nc:rpc-reply/nc:data/dhcp:dhcp</dsrl:parent>
     <dsrl:name>dhcp:default-lease-time</dsrl:name>
     <dsrl:default-content>600</dsrl:default-content>
    </dsrl:element-map>
    <dsrl:element-map>
     <dsrl:parent>
      /nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:subnet
     </dsrl:parent>
     <dsrl:name>dhcp:max-lease-time</dsrl:name>
     <dsrl:default-content>7200</dsrl:default-content>
    </dsrl:element-map>
    <dsrl:element-map>
```

```
     <dsrl:parent>
      /nc:rpc-reply/nc:data/dhcp:dhcp/dhcp:shared-networks/
      dhcp:shared-network/dhcp:subnet
     </dsrl:parent>
     <dsrl:name>dhcp:max-lease-time</dsrl:name>
     <dsrl:default-content>7200</dsrl:default-content>
    </dsrl:element-map>
   </dsrl:maps>
```

Author's Address

Ladislav Lhotka (editor)
CESNET

EMail: lhotka@cesnet.cz