Mapping of YANG to DSDL draft-lhotka-yang-dsdl-map

Ladislav Lhotka
⟨*lhotka@cesnet.cz*⟩

DSDL Translator

Written in Python as a plugin for *pyang*, included since version 0.9.1.

Its output is a RELAX NG schema with annotations:

- Schematron (part 3 of DSDL) – semantic constraints: **must**, **unique**, **keyref**
- Document Schema Renaming Language (DSRL; part 8 of DSDL) – **default**
- Dublin Core – module metadata: **belongs-to**, **contact**, **description** (top-level), **organization**, **reference** (top-level), **revision**
- RELAX NG DTD compatibility annotations – **description**, **reference** (except at top level)
- NETMOD-specific annotations – few *XML attributes* attached to RELAX NG elements: **config**, **key**, **status**, **units**.

  Each annotation type can be selectively switched on or off.

Alternative Output Form

The draft assumes an alternative output form where RELAX NG, Schematron and DSRL are represented as separate stand-alone schemas. This is currently obtained from the DSDL plugin output via XSLT transformations.

DSDL plugin output:

```
<element name="default-lease-time">
  ...
  <sch:assert test=". &lt;= ../max-lease-time">
  default-lease-time must be less than max-lease-time
  </sch:assert>
  <dsrl:default-content>600</dsrl:default-content>
</element>
```

Stand-alone Schemas

Schematron:

```
<sch:pattern>
  <sch:rule context="/dhcp/default-lease-time"
    <sch:assert test=". &lt;= ../max-lease-time">
    default-lease-time must be less than max-lease-time
    </sch:assert>
  </sch:rule>
<sch:pattern>
```

DSRL:

```
<dsrl:element-map>
  <dsrl:within>/dhcp</dsrl:within>
  <dsrl:name>default-lease-time</dsrl:name>
  <dsrl:default-content>600</dsrl:default-content>
<dsrl:element-map>
```

Current Status

Appendix A of the draft contains the result of an automatic translation of *dhcp* module in both XML and compact syntaxes.

The plugin handles all YANG statements and types *except*:

- Non-essential data – **position**, **value**;
- **yang-version** (but may be used by the plugin to check compatibility);
- RPC and notification signatures – **input**, **notification**, **output**, **rpc**;
- **error-app-tag**; **error-message** is used only under **must** (inside Schematron assert);
- Extension features – **argument**, **augment**, **extension**, **when**, **yin-element**;
- Refinements of **use**d groupings, multilevel derived types.

rpc and **notification**

Unlike the rest of a YANG module, which describes contents of an agent datastore, the **rpc** and **notification** statements define contents of specific NETCONF messages.

Options:

1. generate separate schemas for validating datastore content and individual RPC/notification messages.
2. one schema with multiple parts as above under a dummy root element in a special “NETMOD-tree” namespace (e.g., `<nmt:netmod-data>`).

Example Conceptual Tree

```
<nmt:netmod-data>
  <nmt:main>
    ... configuration and status data ...
  </nmt:main>
  <nmt:rpcs>
    <nmt:rpc>
      <nmt:name>...</nmt:name>
      <nmt:input>
        ...
      </nmt:input>
      <nmt:output>
        ...
      </nmt:output>
    </nmt:rpc>
  </nmt:rpcs>
```

(continued)

```
  <nmt:notifications>
    <nmt:notification>
      <nmt:name>...</nmt:name>
      ...
    </nmt:notification>
  </nmt:notifications>
</nmt:netmod-data>
```

extension

YANG language extensions with statement keywords in foreign namespaces can be freely inserted (e.g., in the YIN form) into the RELAX NG schema but it doesn't seem to make much sense without knowing the semantics.

An appropriate decision should be taken after gaining some experience with real-world YANG extensions.

augment

- It is not clear what to do if a top-level **augment** is used for adding new nodes to a foreign schema.
- When a grouping is **use**d, a sibling **augment** with a descendant XPath argument can add nodes to it. The problem is that it is not clear from the module text to which grouping the **augment** applies:

```
uses foo;
uses bar;
augment some/container { ... }
```

This would be easier to handle:

```
uses foo {
  augment some/container { ... }
}
uses bar;
```

Other Open Issues

1. RELAX NG grammar must define exactly one root element (otherwise the XML document wouldn't be well formed).
2. In XPath expressions, namespaces must be explicit.

No Root Element

YANG modules containing only typedefs and groupings (*yang-types*, *inet-types*, . . .) are translated into schemas that have no `<start>` element, i.e., contain only pattern definitions and specify no root element.

Most RELAX NG validators label such schemas as invalid but these "rootless" reusable schemas are a common practice endorsed by RELAX NG authorities. It has to be understood that such schemas can never be used as stand-alone.

Multiple Root Elements

YANG draft: *Due to the possibility of multiple roots the modeled data does not necessarily map to a well-formed XML document. Often a conceptual root node (e.g. <data> or <config> element in NETCONF RPCs) is added to overcome this problem.*

This is a more serious problem. Options are:

1. Remove the possibility of multiple roots from YANG.
2. Specify a fixed conceptual root element, such as `<nmt:netmod-data>`, and use it always as the root element in the DSDL schemas. This would also allow for integrating RPC and notification trees and solve the problem of rootless schemas, too.

XPath and Namespaces

Standard XPath caveat: names used in node tests are always *qualified names* and names without prefix are considered as having *no namespace*.

YANG draft about the XPath argument of **must**: *The null namespace is defined to be the namespace of the current module.*

Result: An XPath expression appearing in **must** usable with standard XML tools such as XSLT processors (without adding the namespace prefixes).

It seems necessary, in accord with XPath specification, to require an explicit namespace prefix (defined by the **prefix** statement) with all local names appearing in XPath expressions.

Deviations from draft-mahy-canmod-dsdl

1. YANG is considered the primary format – DSDL translation follows its semantics, naming, extensibility model, . . .
2. NETMOD-specific annotations are used (as attributes of RELAX NG elements) only where strictly necessary.
3. Some simplifications and corrections.
4. Readability of RELAX NG compact syntax is important.

import and **include**

Both YANG and RELAX NG have powerful extensibility models, however with significant differences.

The mapping algorithm pulls recursively all definitions (**grouping** and **typedef**) that are really used from the imported modules and installs them (with mangled names) in the same DSDL schema.

YANG submodules share the same namespace with the parent module, so the modularity can be retained – RELAX NG `<include>` pattern is used.

Simplifications

- The **unique** statement and *keyref* type are not represented using NETMOD-specific annotations – Schematron asserts are directly inserted. In contrast, **key** is mapped to `nm:key` attribute since it carries additional semantics.
- **mandatory** and **presence** are modeled using RELAX NG means (`<optional>`)
- Schematron asserts are not wrapped in `<pattern>` and `<rule>` elements. They are added when creating the stand-alone Schematron and DSRL schemas.

Default Values – Why DSRL?

RELAX NG DTD Compatibility: *An* `a:defaultValue` *attribute on a RELAX NG* `attribute` *element specifies the default value for the attribute.*

DSRL: *A* `dsrl:default-content` *element can be used to define a default value for an element defined in the schema.*

XML representation of YANG **leaf**s uses elements, so DSRL is the right way.

Besides, it may be useful that DSRL can be presented as a standalone schema so that default values are collected in a separate document.

RELAX NG Compact Syntax

Annotations in the compact syntax are very tricky and can easily make the schema unreadable.

- Grammar annotations: Dublin Core terms

```
dc:creator [ "yang-central.org" ]
```

- Initial annotations: NETMOD-specific attributes

```
[ nm:config = "false" ]
element status { ... }
```

- Following annotation: Schematron and DSRL

```
element max-lease-time {
  xsd:unsignedInt >> dsrl:default-content [ "7200" ]
}
```

- DTD compatibility documentation annotations: **description** and **reference**

```
## See: RFC 2132, sec. 3.17
element domain-name { inet-types__domain-name }?
```