Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC

October 31, 2016

YANG Schema Mount
draft-ietf-netmod-schema-mount-03

Abstract

This document defines a mechanism to combine YANG modules into the schema defined in other YANG modules.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Table of Contents

1. Introduction

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Background

YANG has two mechanisms for extending a data model with additional nodes; "uses" and "augment". The "uses" statement explicitly incorporates the contents of a "grouping" defined in some other module. The "augment" statement explicitly adds contents to a target node defined in some other module. In both these cases, the source and/or target model explicitly defines the relationship between the models.

In some cases these mechanisms are not sufficient. For example, suppose we have a model like ietf-interfaces [RFC7223] that is defined to be implemented in a device. Now suppose we want to model a device that supports multiple logical devices [I-D.rtgyangdt-rtgwg-device-model], where each such logical device has its own instantiation of ietf-interfaces (and other models), but at the same time, we'd like to be able to manage all these logical devices from the main device. We would like something like this:

```
  +--rw interfaces
  | +--rw interface* [name]
  |    ...
  +--rw logical-device* [name]
     +--rw name             string
     |   ...
     +--rw interfaces
       +--rw interface* [name]
          ...
```

With the "uses" approach, ietf-interfaces would have to define a grouping with all its nodes, and the new model for logical devices would have to use this grouping. This is a not a scalable solution, since every time there is a new model defined, we would have to update our model for logical devices to use a grouping from the new model. Another problem is that this approach cannot handle vendor-specific modules.

With the "augment" approach, ietf-interfaces would have to augment the logical-device list with all its nodes, and at the same time define all its nodes on the top-level. This approach is also not scalable, since there may be other models to which we would like to add the interface list.

3. Schema Mount

The schema mount mechanism defined in this document takes a different approach to the extensibility problem described in the previous section. It decouples the definition of the relation between the source and target models from the definitions of the models themselves.

This is accomplished with a YANG extension statement that is used to specify a mount point in a data model. The purpose of a mount point is to define a place in the node hierarchy where other YANG data models may be attached, without any special notation in the other YANG data models. Only "anydata" nodes can be used as mount points.

For each mount point supported by a server, the server populates an operational state node hierarchy with information about which models it has mounted. This node hierarchy can be read by a client in order to learn what is implemented on a server.

Schema mount applies to the data model, and specifically does not assume anything about how the mounted data is implemented. It may be implemented using the same instrumentation as the rest of the system, or it may be implemented by querying some other system. Future specifications may define mechanisms to control or monitor the implementation of specific mount points.

This document allows mounting of complete data models only. Other specifications may extend this model by defining additional mechanisms, for example mounting of sub-hierarchies of a module.

3.1. Augment and Validation in Mounted Data

All paths (in leafrefs, instance-identifiers, XPath expressions, and target nodes of augments) in the data models mounted at a mount point are interpreted with the mount point as the root node, and the mounted data nodes as its children. This means that data within a mounted subtree can never refer to data outside of this subtree.

3.2. Top-level RPCs

If any mounted data model defines RPCs, these RPCs can be invoked by clients by treating them as actions defined where the mount point is specified. An example of this is given in Appendix C.1.

3.3. Top-level Notifications

If the server emits a notification defined at the top-level in any mounted data model, it is treated as if the notification was attached to the data node where the mount point is specified.

4. Data Model

This document defines the YANG 1.1 module [RFC7950] "ietf-yang-schema-mount", which has the following structure:

```
module: ietf-yang-schema-mount
    +--ro schema-mounts
       +--ro namespace* [prefix]
       |  +--ro prefix    yang:yang-identifier
       |  +--ro ns-uri?   inet:uri
       +--ro mount-point* [module name]
       |  +--ro module        yang:yang-identifier
       |  +--ro name          yang:yang-identifier
       |  +--ro (subschema-ref)?
       |     +--:(inline)
       |     |  +--ro inline?       empty
       |     +--:(use-schema)
       |        +--ro use-schema* [name]
       |           +--ro name    -> /schema-mounts/schema/name
       |           +--ro when?   yang:xpath1.0
       +--ro schema* [name]
          +--ro name           string
          +--ro module* [name revision]
          |  +--ro name                yang:yang-identifier
          |  +--ro revision            union
          |  +--ro schema?             inet:uri
          |  +--ro namespace           inet:uri
          |  +--ro feature*            yang:yang-identifier
          |  +--ro deviation* [name revision]
          |  |  +--ro name        yang:yang-identifier
          |  |  +--ro revision    union
          |  +--ro conformance-type    enumeration
          |  +--ro submodule* [name revision]
          |     +--ro name        yang:yang-identifier
          |     +--ro revision    union
          |     +--ro schema?     inet:uri
          +--ro mount-point* [module name]
             +--ro module        yang:yang-identifier
             +--ro name          yang:yang-identifier
             +--ro (subschema-ref)?
                +--:(inline)
                |  +--ro inline?       empty
                +--:(use-schema)
                   +--ro use-schema* [name]
                      +--ro name    -> /schema-mounts/schema/name
                      +--ro when?   yang:xpath1.0
```

5. Schema Mount YANG Module

This module references [RFC6991] and [RFC7895].

<CODE BEGINS> file "ietf-yang-schema-mount@2016-04-05.yang"

```
  module ietf-yang-schema-mount {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount";
    prefix yangmnt;

    import ietf-inet-types {
      prefix inet;
      reference
        "RFC 6991: Common YANG Data Types";
    }

    import ietf-yang-types {
      prefix yang;
      reference
        "RFC 6991: Common YANG Data Types";
    }

    import ietf-yang-library {
      prefix yanglib;
      reference
        "RFC 7895: YANG Module Library";
    }

    organization
      "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

    contact
      "WG Web:   <https://tools.ietf.org/wg/netmod/>
       WG List:  <mailto:netmod@ietf.org>

       WG Chair:  Lou Berger
                  <mailto:lberger@labn.net>

       WG Chair:  Kent Watsen
                  <mailto:kwatsen@juniper.net>

       Editor:    Martin Bjorklund
                  <mailto:mbj@tail-f.com>

       Editor:    Ladislav Lhotka
                  <mailto:lhotka@nic.cz>";

    description
      "This module defines a YANG extension statement that can be used
       to incorporate data models defined in other YANG modules in a
       module. It also defines operational state data that specify the
       overall structure of the data model.
```

```
  revision 2016-10-26 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: YANG Schema Mount";
  }

  /*
   * Extensions
   */

  extension mount-point {
    argument name;
    description
      "The argument 'name' is a yang-identifier. The name of the
       mount point MUST be unique within the module where it is
       defined.

       The 'mount-point' statement can only be present as a
       substatement of 'anydata'.

       If a mount point is defined in a grouping, its name is bound
       to the module where the grouping is used. Note that this
       implies that such a grouping can be used at most once in a
       module.

       A mount point defines a place in the node hierarchy where
       other data models may be attached. A server that implements a
       module with a mount point, populates the
```

```
         /schema-mounts/mount-point list with detailed information on
         which data models are mounted at each mount point.";
    }

    /*
     * Groupings
     */

    grouping mount-point-list {
      description
        "This grouping is used inside the 'schema-mounts' container and
         inside the 'schema' list.";
      list mount-point {
        key "module name";
        description
          "Each entry of this list specifies a subschema for a
           particular mount point.

           Each mount point MUST be defined using the 'mount-point'
           extension in one of the modules listed in the corresponding
           YANG library instance with conformance type 'implement'. The
           corresponding YANG library instance is:

           - standard YANG library state data as defined in RFC 7895, if
             the 'mount-point' list is a child of 'schema-mounts',

           - the contents of the sibling 'yanglib:modules-state'
             container, if the 'mount-point' list is a child of
             'schema'.";
        leaf module {
          type yang:yang-identifier;
          description
            "Name of a module containing the mount point.";
        }
        leaf name {
          type yang:yang-identifier;
          description
            "Name of the mount point defined using the 'mount-point'
             extension.";
        }
        choice subschema-ref {
          description
            "Alternative way for specifying the subschema.";
          leaf inline {
            type empty;
            description
              "This leaf indicates that the server has mounted
               'ietf-yang-library' and 'ietf-schema-mount' at the mount
```

```
          point, and their instantiation (i.e., state data
          containers 'yanglib:modules-state' and 'schema-mounts')
          provides the information about the mounted schema.";
      }
      list use-schema {
        key "name";
        description
          "Each entry of this list contains a reference to a
           subschema defined in the /schema-mounts/schema list. The
           entry can be made conditional by specifying an XPath
           expression in the 'when' leaf.";
        leaf name {
          type leafref {
            path "/schema-mounts/schema/name";
          }
          description
            "Name of the referenced schema.";
        }
        leaf when {
          type yang:xpath1.0;
          description
            "This leaf contains an XPath expression. If it is
             present, then the current entry applies if and only if
             the expression evaluates to true.

             The XPath expression is evaluated once for each
             instance of the anydata node containing the mount
             point for which the 'when' leaf is defined.

             The XPath expression is evaluated using the rules
             specified in sec. 6.4 of RFC 7950, with these
             modifications:

             - The context node is the anydata instance containing
               the corresponding 'mount-point' statement.

             - The accessible tree contains only data belonging to
               the parent schema, i.e., all instances of anydata
               nodes containing the mount points are considered
               empty.

             - The set of namespace declarations is the set of all
               prefix/namespace pairs defined in the
               /schema-mounts/namespace list. Names without a
               namespace prefix belong to the same namespace as the
               context node.";
        }
      }
```

```
      }
    }
  }

  /*
   * State data nodes
   */

  container schema-mounts {
    config "false";
    description
      "Contains information about the structure of the overall data
       model implemented in the server.";
    list namespace {
      key "prefix";
      description
        "This list provides a mapping of namespace prefixes that are
         used in XPath expressions of 'when' leafs to the
         corresponding namespace URI references.";
      leaf prefix {
        type yang:yang-identifier;
        description
          "Namespace prefix.";
      }
      leaf ns-uri {
        type inet:uri;
        description
          "Namespace URI reference.";
      }
    }
    uses mount-point-list;
    list schema {
      key "name";
      description
        "Each entry specifies a schema that can be mounted at a mount
         point. The schema information consists of two parts:

         - an instance of YANG library that defines YANG modules used
           in the schema,

         - mount-point list with content identical to the top-level
           mount-point list (this makes the schema structure
           recursive).";
      leaf name {
        type string;
        description
          "Arbitrary name of the entry.";
      }
```

```
        uses yanglib:module-list;
        uses mount-point-list;
      }
    }
  }
```

<CODE ENDS>

6. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount

Registrant Contact: The IESG.

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

This document registers a YANG module in the YANG Module Names registry [RFC6020].

```
name:         ietf-yang-schema-mount
namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount
prefix:       yangmnt
reference:    RFC XXXX
```

7. Security Considerations

TBD

8. Contributors

The idea of having some way to combine schemas from different YANG modules into one has been proposed independently by several groups of people: Alexander Clemm, Jan Medved, and Eric Voit ([I-D.clemm-netmod-mount]); Ladislav Lhotka ([I-D.lhotka-netmod-ysdl]); and Lou Berger and Christian Hopps.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

9.2. Informative References

[I-D.clemm-netmod-mount]
Clemm, A., Medved, J., and E. Voit, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-05 (work in progress), September 2016.

[I-D.lhotka-netmod-ysdl]
Lhotka, L., "YANG Schema Dispatching Language", draft-lhotka-netmod-ysdl-00 (work in progress), November 2015.

[I-D.rtgyangdt-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-rtgyangdt-rtgwg-device-model-05 (work in progress), August 2016.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

Appendix A. Example: Logical Devices

Logical devices within a device typically use the same set of data models in each instance. This can be modelled with a mount point:

```
module example-logical-devices {
  yang-version 1.1;
  namespace "urn:example:logical-devices";
  prefix exld;

  import ietf-yang-schema-mount {
    prefix yangmnt;
  }

  container logical-devices {
    list logical-device {
      key name;
      leaf name {
        type string;
      }

      anydata root {
        yangmnt:mount-point logical-device;
      }
    }
  }
}
```

A server with two logical devices that both implement "ietf-interfaces" [RFC7223], "ietf-ip" [RFC7277], and "ietf-system" [RFC7317] YANG modules might populate the "schema-mounts" container with:

```
   <schema-mounts
       xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount">
     <mount-point>
       <module>example-logical-devices</module>
       <name>logical-device</name>
       <use-schema>
         <name>logical-device</name>
       </use-schema>
     </mount-point>
     <schema>
       <name>logical-device</name>
       <module>
         <name>ietf-interface</name>
         <revision>2014-05-08</revision>
         <namespace>
           urn:ietf:params:xml:ns:yang:ietf-interfaces
         </namespace>
         <conformance-type>implement</conformance-type>
       </module>
       <module>
         <name>ietf-ip</name>
         <revision>2014-06-16</revision>
         <namespace>
           urn:ietf:params:xml:ns:yang:ietf-ip
         </namespace>
         <conformance-type>implement</conformance-type>
       </module>
       <module>
         <name>ietf-system</name>
         <revision>2014-08-06</revision>
         <namespace>
           urn:ietf:params:xml:ns:yang:ietf-system
         </namespace>
         <conformance-type>implement</conformance-type>
       </module>
       <module>
         <name>ietf-yang-types</name>
         <revision>2013-07-15</revision>
         <namespace>
           urn:ietf:params:xml:ns:yang:ietf-yang-types
         </namespace>
         <conformance-type>import</conformance-type>
       </module>
     </schema>
   </schema-mounts>
```

and the "logical-devices" container might have:

```
<logical-devices xmlns="urn:example:logical-devices">
  <logical-device>
    <name>vrtrA</name>
    <root>
      <interfaces
          xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>eth0</name>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <enabled>true</enabled>
              ...
            </ipv6>
          ...
        </interface>
      </interfaces>
      <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
        ...
      </system>
    </root>
  </logical-device>
  <logical-device>
    <name>vrtrB</name>
    <root>
      <interfaces
          xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>eth0</name>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <enabled>true</enabled>
              ...
            </ipv6>
          ...
        </interface>
      </interfaces>
      <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
        ...
      </system>
    </root>
  </logical-device>
</logical-devices>
```

Appendix B. Example: Network Manager with Fixed Device Models

This example shows how a Network Manager application can use schema mount to define a data model for a network consisting of devices whose data models are known a priori and fixed.

Assume for simplicity that only two device types are used (switch and router), and they are identified by identities defined in the module "example-device-types":

```
module example-device-types {
  namespace "http://example.org/device-types";
  prefix edt;
  identity device-type;
  identity switch-device {
    base device-type;
  }
  identity router-device {
    base device-type;
  }
}
```

Schema mount is used to mount the device data models conditionally, depending on the "type" leaf that is a sibling of the mount point. This approach is similar to "ietf-interfaces" [RFC7223] where the same effect is achieved via conditional augments.

The top-level module may look as follows:

```
module example-network-manager-fixed {
  yang-version 1.1;
  namespace "urn:example:network-manager-fixed";
  prefix exf;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-schema-mount {
    prefix yangmnt;
  }
  import example-device-types {
    prefix edt;
  }

  container managed-devices {
    description
      "The managed devices and device communication settings.";

    list device {
      key name;
      leaf name {
        type string;
      }
      leaf type {
```

```
          type identityref {
            base edt:device-type;
          }
        }
        container transport {
          choice protocol {
            mandatory true;
            container netconf {
              leaf address {
                type inet:ip-address;
                mandatory true;
              }
              container authentication {
                // ...
              }
            }
            container restconf {
              leaf address {
                type inet:ip-address;
                mandatory true;
              }
              // ...
            }
          }
        }
        anydata root {
          yangmnt:mount-point managed-device;
        }
      }
    }
  }
```

The "schema-mounts" container may have the following data:

```
<data-model
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount">
  <namespace>
    <prefix>edt</prefix>
    <ns-uri>http://example.org/device-types</ns-uri>
  </namespace>
  <mount-point>
    <module>example-network-manager</module>
    <name>managed-device</name>
    <use-schema>
      <name>switch</name>
      <when>derived-from-or-self(../type, 'edt:switch-device')</when>
    </use-schema>
    <use-schema>
      <name>router</name>
      <when>derived-from-or-self(../type, 'edt:router-device')</when>
    </use-schema>
  </mount-point>
  <schema>
    <name>switch</name>
    <module>
      ...
    </module>
    ...
  </schema>
  <schema>
    <name>router</name>
    <module>
      ...
    </module>
    ...
  </schema>
</data-model>
```

The "devices" list may contain any number of instances of either type.

Appendix C. Example: Network Manager with Arbitrary Device Models

This example shows how a Network Manager application can use schema mount to define a data model for a network consisting of devices whose data models are not known in advance -- each device is expected to provide its data model dynamically.

Schema mount is used to mount the data models that each device supports, and these data models can be discovered by inspecting state data under the corresponding mount point. Every such device must

therefore implement "ietf-yang-library" and optionally
"ietf-schema-mount".

```
module example-network-manager-arbitrary {
  yang-version 1.1;
  namespace "urn:example:network-manager-arbitrary";
  prefix exa;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-schema-mount {
    prefix yangmnt;
  }

  container managed-devices {
    description
      "The managed devices and device communication settings.";

    list device {
      key name;
      leaf name {
        type string;
      }
      container transport {
        choice protocol {
          mandatory true;
          container netconf {
            leaf address {
              type inet:ip-address;
              mandatory true;
            }
            container authentication {
              // ...
            }
          }
          container restconf {
            leaf address {
              type inet:ip-address;
              mandatory true;
            }
            // ...
          }
        }
      }
      anydata root {
        yangmnt:mount-point managed-device;
      }
    }
  }
}
```

The "schema-mounts" container may have the following data:

```
<data-model
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount">
  <mount-point>
    <module>example-network-manager</module>
    <name>managed-device</name>
    <inline/>
  </mount-point>
</data-model>
```

The "devices" container might have:

```
<devices xmlns="urn:example:network-manager">
  <device>
    <name>rtrA</name>
    <transport>
      <netconf>
        <address>2001:db8::2</address>
        <authentication>
          ...
        </authentication>
        ...
      </netconf>
    </transport>
    <root>
      <modules-state
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
        <module>
          <name>ietf-system</name>
          ...
        </module>
      </modules-state>
      <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
        ...
      </system>
    </root>
  </device>
  <device>
    <name>rtrB</name>
    <transport>
      <restconf>
        <address>2001:db8::3</address>
        <authentication>
          ...
        </authentication>
        ...
      </restconf>
```

```
      </transport>
      <root>
        <modules-state
            xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
          <module>
            <name>ietf-interfaces</name>
            ...
          </module>
        </modules-state>
        <interfaces
            xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          ...
        </interfaces>
      </root>
    </device>
  </devices>
```

C.1. Invoking an RPC

A client that wants to invoke the "restart" operation [RFC7317] on the managed device "rtrA" over NETCONF [RFC6241] can send:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <managed-devices xmlns="urn:example:network-manager">
      <device>
        <name>rtrA</name>
        <root>
          <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
            <restart/>
          </system>
        </root>
      </device>
    </managed-devices>
  </action>
</rpc>
```

Appendix D. Open Issues

o Is the 'mount-point' extension really needed? Now that mount points can only appear under anydata nodes, there seems to be little need to otherwise restrict mount point locations. In the 'mount-point' list, schema node identifiers (as in 'augment' statements) can be used instead of the (module, name) pair for identifying mount points. As a useful side effect, a grouping containing mount points could be used any number of times in the same module. OTOH, by using this extension, the intention of the

data modeller is clear, and it provides a formal machine readable instruction about where mounts are allowed to occur.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: mbj@lhotka@nic.cz