Internet Engineering Task Force (IETF)
Request for Comments: 6244
Category: Informational
ISSN: 2070-1721

P. Shafer
Juniper Networks

June 2011

An Architecture for Network Management Using NETCONF and YANG

Abstract

The Network Configuration Protocol (NETCONF) gives access to native capabilities of the devices within a network, defining methods for manipulating configuration databases, retrieving operational data, and invoking specific operations. YANG provides the means to define the content carried via NETCONF, both data and operations. Using both technologies, standard modules can be defined to give interoperability and commonality to devices, while still allowing devices to express their unique capabilities.

This document describes how NETCONF and YANG help build network management applications that meet the needs of network operators.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

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). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see 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/rfc6244.

Copyright Notice

Table of Contents

1. Origins of NETCONF and YANG

Networks are increasing in complexity and capacity, as well as the density of the services deployed upon them. Uptime, reliability, and predictable latency requirements drive the need for automation. The problems with network management are not simple. They are complex and intricate. But these problems must be solved for networks to meet the stability needs of existing services while incorporating new services in a world where the growth of networks is exhausting the supply of qualified networking engineers.

In June of 2002, the Internet Architecture Board (IAB) held a workshop on Network Management [RFC3535]. The members of this workshop made a number of observations and recommendations for the IETF's consideration concerning the issues operators were facing in their network management-related work as well as issues they were having with the direction of the IETF activities in this area.

The output of this workshop was focused on current problems. The observations were reasonable and straightforward, including the need for transactions, rollback, low implementation costs, and the ability to save and restore the device's configuration data. Many of the observations give insight into the problems operators were having with existing network management solutions, such as the lack of full coverage of device capabilities and the ability to distinguish between configuration data and other types of data.

Based on these directions, the NETCONF working group was formed and the Network Configuration (NETCONF) protocol was created. This protocol defines a simple mechanism where network management applications, acting as clients, can invoke operations on the devices, which act as servers. The NETCONF specification [RFC4741] defines a small set of operations, but goes out of its way to avoid making any requirements on the data carried in those operations, preferring to allow the protocol to carry any data. This "data model agnostic" approach allows data models to be defined independently.

But lacking a means of defining data models, the NETCONF protocol was not usable for standards-based work. Existing data modeling languages such as the XML Schema Definition (XSD) [W3CXSD] and the Document Schema Definition Languages (DSDL) [ISODSDL] were considered, but were rejected because of the problem that domains have little natural overlap. Defining a data model or protocol that is encoded in XML is a distinct problem from defining an XML document. The use of NETCONF operations places requirements on the data content that are not shared with the static document problem domain addressed by schema languages like XSD or RELAX NG.

In 2007 and 2008, the issue of a data modeling language for NETCONF was discussed in the OPS and APP areas of IETF 70 and 71, and a design team was tasked with creating a requirements document [RCDML]. After discussing the available options at the CANMOD BoF at IETF 71, the community wrote a charter for the NETMOD working group. An excellent description of this time period is available at <http://www.ietf.org/mail-archive/web/ietf/current/msg51644.html>.

In 2008 and 2009, the NETMOD working group produced a specification for YANG [RFC6020] as a means for defining data models for NETCONF, allowing both standard and proprietary data models to be published in a form that is easily digestible by human readers and satisfies many of the issues raised in the IAB NM workshop. This brings NETCONF to a point where is can be used to develop standard data models within the IETF.

YANG allows a modeler to create a data model, to define the organization of the data in that model, and to define constraints on that data. Once published, the YANG module acts as a contract between the client and server, with both parties understanding how their peer will expect them to behave. A client knows how to create valid data for the server, and knows what data will be sent from the server. A server knows the rules that govern the data and how it should behave.

YANG also incorporates a level of extensibility and flexibility not present in other model languages. New modules can augment the data hierarchies defined in other modules, seamlessly adding data at appropriate places in the existing data organization. YANG also allows new statements to be defined, allowing the language itself to be expanded in a consistent way.

This document presents an architecture for YANG, describing how YANG-related technologies work and how solutions built on them can address the network management problem domain.

2. Elements of the Architecture

2.1. NETCONF

NETCONF defines an XML-based remote procedure call (RPC) mechanism that leverages the simplicity and availability of high-quality XML parsers. XML gives a rich, flexible, hierarchical, standard representation of data that matches the needs of networking devices. NETCONF carries configuration data and operations as requests and replies using RPCs encoded in XML over a connection-oriented transport.

XML's hierarchical data representation allows complex networking data to be rendered in a natural way. For example, the following configuration places interfaces in OSPF areas. The <ospf> element contains a list of <area> elements, each of which contain a list of <interface> elements. The <name> element identifies the specific area or interface. Additional configuration for each area or interface appears directly inside the appropriate element.

```
      <ospf xmlns="http://example.org/netconf/ospf">

        <area>
          <name>0.0.0.0</name>

          <interface>
            <name>ge-0/0/0.0</name>
            <!-- The priority for this interface -->
            <priority>30</priority>
            <metric>100</metric>
            <dead-interval>120</dead-interval>
          </interface>

          <interface>
            <name>ge-0/0/1.0</name>
            <metric>140</metric>
          </interface>
        </area>

        <area>
          <name>10.1.2.0</name>

          <interface>
            <name>ge-0/0/2.0</name>
            <metric>100</metric>
          </interface>

          <interface>
            <name>ge-0/0/3.0</name>
            <metric>140</metric>
            <dead-interval>120</dead-interval>
          </interface>
        </area>
      </ospf>
```

NETCONF includes mechanisms for controlling configuration datastores. Each datastore is a specific collection of configuration data that can be used as source or target of the configuration-related operations. The device can indicate whether it has a distinct "startup" configuration datastore, whether the current or "running"

datastore is directly writable, or whether there is a "candidate" configuration datastore where configuration changes can be made that will not affect the device until a "commit-configuration" operation is invoked.

NETCONF defines operations that are invoked as RPCs from the client (the application) to the server (running on the device). The following table lists some of these operations:

Operation	Description
commit	Commit the "candidate" configuration to "running"
copy-config	Copy one configuration datastore to another
delete-config	Delete a configuration datastore
edit-config	Change the contents of a configuration datastore
get-config	Retrieve all or part of a configuration datastore
lock	Prevent changes to a datastore from another party
unlock	Release a lock on a datastore

NETCONF's "capability" mechanism allows the device to announce the set of capabilities that the device supports, including protocol operations, datastores, data models, and other abilities. These are announced during session establishment as part of the <hello> message. A client can inspect the hello message to determine what the device is capable of and how to interact with the device to perform the desired tasks.

NETCONF also defines a means of sending asynchronous notifications from the server to the client, described in [RFC5277].

In addition, NETCONF can fetch state data, receive notifications, and invoke additional RPC methods defined as part of a capability. Complete information about NETCONF can be found in [RFC4741].

2.1.1. NETCONF Transport Mappings

NETCONF can run over any transport protocol that meets the requirements defined in RFC 4741, including

- connection-oriented operation
- authentication
- integrity
- confidentiality

[RFC4742] defines a mapping for the Secure Shell (SSH) [RFC4251] protocol, which is the mandatory transport protocol. Others include SOAP [RFC4743], the Blocks Extensible Exchange Protocol (BEEP) [RFC4744], and Transport Layer Security (TLS) [RFC5539].

2.2. YANG

YANG is a data modeling language for NETCONF. It allows the description of hierarchies of data nodes ("nodes") and the constraints that exist among them. YANG defines data models and how to manipulate those models via NETCONF protocol operations.

Each YANG module defines a data model, uniquely identified by a namespace URI. These data models are extensible in a manner that allows tight integration of standard data models and proprietary data models. Models are built from organizational containers, lists of data nodes, and data-node-forming leafs of the data tree.

```
module example-ospf {
    namespace "http://example.org/netconf/ospf";
    prefix ospf;

    import network-types {  // Access another module's def'ns
        prefix nett;
    }

    container ospf {   // Declare the top-level tag
        list area {    // Declare a list of "area" nodes
            key name;  // The key "name" identifies list members
            leaf name {
                type nett:area-id;
            }
            list interface {
                key name;
                leaf name {
                    type nett:interface-name;
                }
                leaf priority {
                    description "Designated router priority";
                    type uint8;  // The type is a constraint on
                                 // valid values for "priority".
                }
                leaf metric {
                    type uint16 {
                        range 1..65535;
                    }
                }
                leaf dead-interval {
                    units seconds;
                    type uint16 {
                        range 1..65535;
                    }
                }
            }
        }
    }
}
```

A YANG module defines a data model in terms of the data, its hierarchical organization, and the constraints on that data. YANG defines how this data is represented in XML and how that data is used in NETCONF operations.

The following table briefly describes some common YANG statements:

Statement	Description
augment	Extends existing data hierarchies
choice	Defines mutually exclusive alternatives
container	Defines a layer of the data hierarchy
extension	Allows new statements to be added to YANG
feature	Indicates parts of the model that are optional
grouping	Groups data definitions into reusable sets
key	Defines the key leafs for lists
leaf	Defines a leaf node in the data hierarchy
leaf-list	A leaf node that can appear multiple times
list	A hierarchy that can appear multiple times
notification	Defines notification
rpc	Defines input and output parameters for an RPC operation
typedef	Defines a new type
uses	Incorporates the contents of a "grouping"

2.2.1. Constraints

YANG allows the modeler to add constraints to the data model to prevent impossible or illogical data. These constraints give clients information about the data being sent from the device, and also allow the client to know as much as possible about the data the device will accept, so the client can send correct data. These constraints apply to configuration data, but can also be used for rpc and notification data.

The principal constraint is the "type" statement, which limits the contents of a leaf node to that of the named type. The following table briefly describes some other common YANG constraints:

Statement	Description
length	Limits the length of a string
mandatory	Requires the node appear
max-elements	Limits the number of instances in a list
min-elements	Limits the number of instances in a list
must	XPath expression must be true
pattern	Regular expression must be satisfied
range	Value must appear in range
reference	Value must appear elsewhere in the data
unique	Value must be unique within the data
when	Node is only present when XPath expression is true

The "must" and "when" statements use XPath [W3CXPATH] expressions to specify conditions that are semantically evaluated against the data hierarchy, but neither the client nor the server are required to implement the XPath specification. Instead they can use any means to ensure these conditions are met.

2.2.2. Flexibility

YANG uses the "union" type and the "choice" and "feature" statements to give modelers flexibility in defining their data models. The "union" type allows a single leaf to accept multiple types, like an integer or the word "unbounded":

```
type union {
    type int32;
    type enumeration {
        enum "unbounded";
    }
}
```

The "choice" statement lists a set of mutually exclusive nodes, so a valid configuration can choose any one node (or case). The "feature" statement allows the modeler to identify parts of the model that can be optional, and allows the device to indicate whether it implements these optional portions.

The "deviation" statement allows the device to indicate parts of a YANG module that the device does not faithfully implement. While devices are encouraged to fully abide according to the contract presented in the YANG module, real-world situations may force the device to break the contract. Deviations give a means of declaring this limitation, rather than leaving it to be discovered via run-time errors.

2.2.3. Extensibility Model

XML includes the concept of namespaces, allowing XML elements from different sources to be combined in the same hierarchy without risking collision. YANG modules define content for specific namespaces, but one module may augment the definition of another module, introducing elements from that module's namespace into the first module's hierarchy.

Since one module can augment another module's definition, hierarchies of definitions are allowed to grow, as definitions from multiple sources are added to the base hierarchy. These augmentations are qualified using the namespace of the source module, helping to avoid issues with name conflicts as the modules change over time.

For example, if the above OSPF configuration were the standard, a vendor module may augment this with vendor-specific extensions.

```
    module vendorx-ospf {
        namespace "http://vendorx.example.com/ospf";
        prefix vendorx;

        import example-ospf {
            prefix ospf;
        }

        augment /ospf:ospf/ospf:area/ospf:interfaces {
            leaf no-neighbor-down-notification {
                type empty;
                description "Don't inform other protocols about"
                          + " neighbor down events";
            }
        }
    }
```

The <no-neighbor-down-notification> element is then placed in the vendorx namespace:

```
      <ospf xmlns="http://example.org/netconf/ospf"
            xmlns:vendorx="http://vendorx.example.com/ospf">

        <area>
          <name>0.0.0.0</name>

          <interface>
            <name>ge-0/0/0.0</name>
            <priority>30</priority>
            <vendorx:no-neighbor-down-notification/>
          </interface>

        </area>
      </ospf>
```

Augmentations are seamlessly integrated with base modules, allowing them to be fetched, archived, loaded, and deleted within their natural hierarchy. If a client application asks for the configuration for a specific OSPF area, it will receive the sub-hierarchy for that area, complete with any augmented data.

2.3. YANG Translations

The YANG data modeling language is the central piece of a group of related technologies. The YANG language itself, described in [RFC6020], defines the syntax of the language and its statements, the meaning of those statements, and how to combine them to build the hierarchy of nodes that describe a data model.

That document also defines the "on the wire" XML content for NETCONF operations on data models defined in YANG modules. This includes the basic mapping between YANG data tree nodes and XML elements, as well as mechanisms used in <edit-config> content to manipulate that data, such as arranging the order of nodes within a list.

YANG uses a syntax that is regular and easily described, primarily designed for human readability. YANG's syntax is friendly to email, diff, patch, and the constraints of RFC formatting.

2.3.1. YIN

In some environments, incorporating a YANG parser may not be an acceptable option. For those scenarios, an XML grammar for YANG is defined as YIN (YANG Independent Notation). YIN allows the use of XML parsers that are readily available in both open source and commercial versions. Conversion between YANG and YIN is direct, loss-less, and reversible. YANG statements are converted to XML elements, preserving the structure and content of YANG, but enabling

the use of off-the-shelf XML parsers rather than requiring the integration of a YANG parser. YIN maintains complete semantic equivalence with YANG.

2.3.2. DSDL (RELAX NG)

Since NETCONF content is encoded in XML, it is natural to use XML schema languages for their validation. To facilitate this, YANG offers a standardized mapping of YANG modules into Document Schema Definition Languages [RFC6110], of which RELAX NG is a major component.

DSDL is considered to be the best choice as a standard schema language because it addresses not only grammar and datatypes of XML documents but also semantic constraints and rules for modifying the information set of the document.

In addition, DSDL offers formal means for coordinating multiple independent schemas and specifying how to apply the schemas to the various parts of the document. This is useful since YANG content is typically composed of multiple vocabularies.

2.4. YANG Types

YANG supports a number of builtin types, and allows additional types to be derived from those types in an extensible manner. New types can add additional restrictions to allowable data values.

A standard type library for use by YANG is available [RFC6021]. These YANG modules define commonly used data types for IETF-related standards.

2.5. IETF Guidelines

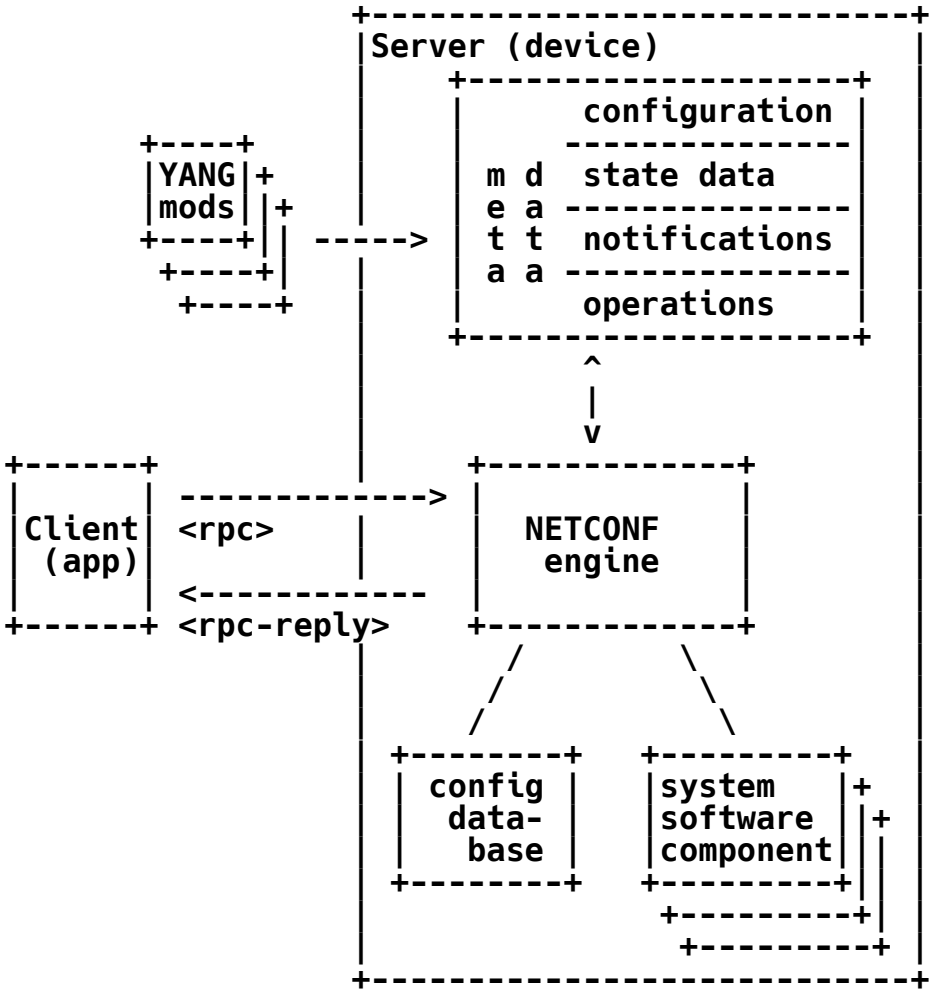
A set of additional guidelines is defined that indicate desirable usage for authors and reviewers of Standards-Track specifications containing YANG data model modules [RFC6087]. These guidelines should be used as a basis for reviews of other YANG data model documents.

3. Working with YANG

3.1. Building NETCONF- and YANG-Based Solutions

In the typical YANG-based solution, the client and server are driven by the content of YANG modules. The server includes the definitions of the modules as meta-data that is available to the NETCONF engine. This engine processes incoming requests, uses the meta-data to parse

and verify the request, performs the requested operation, and returns the results to the client.



To use YANG, YANG modules must be defined to model the specific problem domain. These modules are then loaded, compiled, or coded into the server.

The sequence of events for the typical client/server interaction may be as follows:

- A client application ([C]) opens a NETCONF session to the server (device) ([S])
- [C] and [S] exchange <hello> messages containing the list of capabilities supported by each side, allowing [C] to learn the modules supported by [S]

- [C] builds and sends an operation defined in the YANG module, encoded in XML, within NETCONF's <rpc> element
- [S] receives and parses the <rpc> element
- [S] verifies the contents of the request against the data model defined in the YANG module
- [S] performs the requested operation, possibly changing the configuration datastore
- [S] builds the response, containing the response, any requested data, and any errors
- [S] sends the response, encoded in XML, within NETCONF's <rpc-reply> element
- [C] receives and parses the <rpc-reply> element
- [C] inspects the response and processes it as needed

Note that there is no requirement for the client or server to process the YANG modules in this way. The server may hard code the contents of the data model, rather than handle the content via a generic engine. Or the client may be targeted at the specific YANG model, rather than being driven generically. Such a client might be a simple shell script that stuffs arguments into an XML payload template and sends it to the server.

3.2. Addressing Operator Requirements

NETCONF and YANG address many of the issues raised in the IAB NM workshop.

- Ease of use: YANG is designed to be human friendly, simple, and readable. Many tricky issues remain due to the complexity of the problem domain, but YANG strives to make them more visible and easier to deal with.
- Configuration and state data: YANG clearly divides configuration data from other types of data.
- Transactions: NETCONF provides a simple transaction mechanism.
- Generation of deltas: A YANG module gives enough information to generate the delta needed to change between two configuration data sets.

- Dump and restore: NETCONF gives the ability to save and restore configuration data. This can also be performed for a specific YANG module.

- Network-wide configuration: NETCONF supports robust network-wide configuration transactions via the commit and confirmed-commit capabilities. When a change is attempted that affects multiple devices, these capabilities simplify the management of failure scenarios, resulting in the ability to have transactions that will dependably succeed or fail atomically.

- Text-friendly: YANG modules are very text friendly, as is the data they define.

- Configuration handling: NETCONF addresses the ability to distinguish between distributing configuration data and activating it.

- Task-oriented: A YANG module can define specific tasks as RPC operations. A client can choose to invoke the RPC operation or to access any underlying data directly.

- Full coverage: YANG modules can be defined that give full coverage to all the native abilities of the device. Giving this access avoids the need to resort to the command line interface (CLI) using tools such as Expect [SWEXPECT].

- Timeliness: YANG modules can be tied to CLI operations, so all native operations and data are immediately available.

- Implementation difficulty: YANG's flexibility enables modules that can be more easily implemented. Adding "features" and replacing "third normal form" with a natural data hierarchy should reduce complexity.

- Simple data modeling language: YANG has sufficient power to be usable in other situations. In particular, on-box API and native CLI can be integrated to achieve simplification of the infrastructure.

- Internationalization: YANG uses UTF-8 [RFC3629] encoded Unicode characters.

- Event correlation: YANG integrates RPC operations, notification, configuration, and state data, enabling internal references. For example, a field in a notification can be tagged as pointing to a BGP peer, and the client application can easily find that peer in the configuration data.

- Implementation costs: Significant effort has been made to keep implementation costs as low as possible.

- Human-friendly syntax: YANG's syntax is optimized for the reader, specifically the reviewer on the basis that this is the most common human interaction.

- Post-processing: Use of XML will maximize the opportunities for post-processing of data, possibly using XML-based technologies like XPath [W3CXPATH], XQuery [W3CXQUERY], and XSLT [W3CXSLT].

- Semantic mismatch: Richer, more descriptive data models will reduce the possibility of semantic mismatch. With the ability to define new primitives, YANG modules will be more specific in content, allowing more enforcement of rules and constraints.

- Security: NETCONF runs over transport protocols secured by SSH or TLS, allowing secure communications and authentication using well-trusted technology. The secure transport can use existing key and credential management infrastructure, reducing deployment costs.

- Reliable: NETCONF and YANG are solid and reliable technologies. NETCONF is connection based, and includes automatic recovery mechanisms when the connection is lost.

- Delta friendly: YANG-based models support operations that are delta friendly. Add, change, insert, and delete operations are all well defined.

- Method-oriented: YANG allows new RPC operations to be defined, including an operation name, which is essentially a method. The input and output parameters of the RPC operations are also defined in the YANG module.

3.3. Roles in Building Solutions

Building NETCONF- and YANG-based solutions requires interacting with many distinct groups. Modelers must understand how to build useful models that give structure and meaning to data while maximizing the flexibility of that data to "future proof" their work. Reviewers need to quickly determine if that structure is accurate. Device developers need to code that data model into their devices, and application developers need to code their applications to take advantage of that data model. There are a variety of strategies for performing each piece of this work. This section discusses some of those strategies.

3.3.1. Modeler

The modeler defines a data model based on their in-depth knowledge of the problem domain being modeled. This model should be as simple as possible, but should balance complexity with expressiveness. The organization of the model not only should target the current model but also should allow for extensibility from other modules and for adaptability to future changes.

Additional modeling issues are discussed in Section 4.

3.3.2. Reviewer

The reviewer role is perhaps the most important and the time reviewers are willing to give is precious. To help the reviewer, YANG stresses readability, with a human-friendly syntax, natural data hierarchy, and simple, concise statements.

3.3.3. Device Developer

The YANG model tells the device developer what data is being modeled. The developer reads the YANG models and writes code that supports the model. The model describes the data hierarchy and associated constraints, and the description and reference material helps the developer understand how to transform the model's view into the device's native implementation.

3.3.3.1. Generic Content Support

The YANG model can be compiled into a YANG-based engine for either the client or server side. Incoming data can be validated, as can outgoing data. The complete configuration datastore may be validated in accordance with the constraints described in the data model.

Serializers and de-serializers for generating and receiving NETCONF content can be driven by the meta-data in the model. As data is received, the meta-data is consulted to ensure the validity of incoming XML elements.

3.3.3.2. XML Definitions

The YANG module dictates the XML encoding for data sent via NETCONF. The rules that define the encoding are fixed, so the YANG module can be used to ascertain whether a specific NETCONF payload is obeying the rules.

3.3.4. Application Developer

The YANG module tells the application developer what data can be modeled. Developers can inspect the modules and take one of three distinct views. In this section, we will consider them and the impact of YANG on their design. In the real world, most applications are a mixture of these approaches.

3.3.4.1. Hard Coded

An application can be coded against the specific, well-known contents of YANG modules, implementing their organization, rules, and logic directly with explicit knowledge. For example, a script could be written to change the domain name of a set of devices using a standard YANG module that includes such a leaf node. This script takes the new domain name as an argument and inserts it into a string containing the rest of the XML encoding as required by the YANG module. This content is then sent via NETCONF to each of the devices.

This type of application is useful for small, fixed problems where the cost and complexity of flexibility are overwhelmed by the ease of hard coding direct knowledge into the application.

3.3.4.2. Bottom Up

An application may take a generic, bottom-up approach to configuration, concentrating on the device's data directly and treating that data without specific understanding.

YANG modules may be used to drive the operation of the YANG equivalent of a "MIB browser". Such an application manipulates the device's configuration data based on the data organization contained in the YANG module. For example, a GUI may present a straightforward visualization where elements of the YANG hierarchy are depicted in a hierarchy of folders or GUI panels. Clicking on a line expands to the contents of the matching XML hierarchy.

This type of GUI can easily be built by generating XSLT stylesheets from the YANG data models. An XSLT engine can then be used to turn configuration data into a set of web pages.

The YANG modules allow the application to enforce a set of constraints without understanding the semantics of the YANG module.

3.3.4.3. Top Down

In contrast to the bottom-up approach, the top-down approach allows the application to take a view of the configuration data that is distinct from the standard and/or proprietary YANG modules. The application is free to construct its own model for data organization and to present this model to the user. When the application needs to transmit data to a device, the application transforms its data from the problem-oriented view of the world into the data needed for that particular device. This transformation is under the control and maintenance of the application, allowing the transformation to be changed and updated without affecting the device.

For example, an application could be written that models VPNs in a network-oriented view. The application would need to transform these high-level VPN definitions into the configuration data that would be handed to any particular device within a VPN.

Even in this approach, YANG is useful since it can be used to model the VPN. For example, the following VPN straw-man models a list of VPNs, each with a protocol, a topology, a list of member interfaces, and a list of classifiers.

```
    list example-bgpvpn {
        key name;
        leaf name { ... }
        leaf protocol {
            type enumeration {
                enum bgpvpn;
                enum l2vpn;
            }
        }
        leaf topology {
            type enumeration {
                enum hub-n-spoke;
                enum mesh;
            }
        }
        list members {
            key "device interface";
            leaf device { ... }
            leaf interface { ... }
        }
        list classifiers {
            ...
        }
    }
```

The application can use such a YANG module to drive its operation, building VPN instances in a database and then pushing the configuration for those VPNs to individual devices either using a standard device model (e.g., example-bgpvpn.yang) or by transforming that standard device content into some proprietary format for devices that do not support that standard.

4. Modeling Considerations

This section discusses considerations the modeler should be aware of while developing models in YANG.

4.1. Default Values

The concept of default values is simple, but their details, representation, and interaction with configuration data can be difficult issues. NETCONF leaves default values as a data model issue, and YANG gives flexibility to the device implementation in terms of how default values are handled. The requirement is that the device "MUST operationally behave as if the leaf was present in the data tree with the default value as its value". This gives the device implementation choices in how default values are handled.

One choice is to view the configuration as a set of instructions for how the device should be configured. If a data value that is given as part of those instructions is the default value, then it should be retained as part of the configuration, but if it is not explicitly given, then the value is not considered to be part of the configuration.

Another choice is to trim values that are identical to the default values, implicitly removing them from the configuration datastore. The act of setting a leaf to its default value effectively deletes that leaf.

The device could also choose to report all default values, regardless of whether they were explicitly set. This choice eases the work of a client that needs default values, but may significantly increase the size of the configuration data.

These choices reflect the default handling schemes of widely deployed networking devices and supporting them allows YANG to reduce implementation and deployment costs of YANG-based models.

When the client retrieves data from the device, it must be prepared to handle the absence of leaf nodes with the default value, since the server is not required to send such leaf elements. This permits the device to implement either of the first two default handling schemes given above.

Regardless of the implementation choice, the device can support the "with-defaults" capability [RFC6243] and give the client the ability to select the desired handling of default values.

When evaluating the XPath expressions for constraints like "must" and "when", the evaluation context for the expressions will include any appropriate default values, so the modeler can depend on consistent behavior from all devices.

4.2. Compliance

In developing good data models, there are many conflicting interests the data modeler must keep in mind. Modelers need to be aware of five issues with models and devices:

- usefulness
- compliance
- flexibility
- extensibility
- deviations

For a model to be interesting, it must be useful, solving a problem in a more direct or more powerful way than can be accomplished without the model. The model should maximize the usefulness of the model within the problem domain.

Modelers should build models that maximize the number of devices that can faithfully implement the model. If the model is drawn too narrowly, or includes too many assumptions about the device, then the difficulty and cost of accurately implementing the model will lead to low-quality implementations and interoperability issues, and will reduce the value of the model.

Modelers can use the "feature" statement in their models to give the device some flexibility by partitioning their model and allowing the device to indicate which portions of the model are implemented on the

device. For example, if the model includes some "logging" feature, a device with no storage facilities for the log can tell the client that it does not support this feature of the model.

Models can be extended via the "augment" statement, and the modeler should consider how their model is likely to be extended. These augmentations can be defined by vendors, applications, or standards bodies.

Deviations are a means of allowing the devices to indicate where its implementation is not in full compliance with the model. For example, once a model is published, an implementer may decide to make a particular node configurable, where the standard model describes it as state data. The implementation reports the value normally and may declare a deviation that this device behaves in a different manner than the standard. Applications capable of discovering this deviation can make allowances, but applications that do not discover the deviation can continue treating the implementation as if it were compliant.

Rarely, implementations may make decisions that prevent compliance with the standard. Such occasions are regrettable, but they remain a part of reality, and modelers and application writers ignore them at their own risk. An implementation that emits an integer leaf as "cow" would be difficult to manage, but applications should expect to encounter such misbehaving devices in the field.

Despite this, both client and server should view the YANG module as a contract, with both sides agreeing to abide by the terms. The modeler should be explicit about the terms of such a contract, and both client and server implementations should strive to faithfully and accurately implement the data model described in the YANG module.

4.3. Data Distinctions

The distinction between configuration data, operational state data, and statistics is important to understand for data model writers and people who plan to extend the NETCONF protocol. This section first discusses some background and then provides a definition and some examples.

4.3.1. Background

During the IAB NM workshop, operators did formulate the following two requirements, as listed in [RFC3535]:

2. It is necessary to make a clear distinction between configuration data, data that describes operational state, and statistics. Some devices make it very hard to determine which parameters were administratively configured and which were obtained via other mechanisms such as routing protocols.

3. It is required to be able to fetch separately configuration data, operational state data, and statistics from devices, and to be able to compare these between devices.

The NETCONF protocol defined in RFC 4741 distinguishes two types of data -- namely, configuration data and state data:

> Configuration data is the set of writable data that is required to transform a system from its initial default state into its current state.
>
> State data is the additional data on a system that is not configuration data such as read-only status information and collected statistics.

NETCONF does not follow the distinction formulated by the operators between configuration data, operational state data, and statistical data, since it considers state data to include both statistics and operational state data.

4.3.2. Definitions

Below is a definition for configuration data, operational state data, and statistical data. The definition borrows from previous work.

- Configuration data is the set of writable data that is required to transform a system from its initial default state into its current state [RFC4741].

- Operational state data is a set of data that has been obtained by the system at runtime and influences the system's behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

- Statistical data is the set of read-only data created by a system itself. It describes the performance of the system and its components.

The following examples help to clarify the difference between configuration data, operational state data, and statistical data.

4.3.2.1. Example 1: IP Routing Table

IP routing tables can contain entries that are statically configured (configuration data) as well as entries obtained from routing protocols such as OSPF (operational state data). In addition, a routing engine might collect statistics like how often a particular routing table entry has been used.

4.3.2.2. Example 2: Interfaces

Network interfaces usually come with a large number of attributes that are specific to the interface type and in some cases specific to the cable plugged into an interface. Examples are the maximum transmission unit of an interface or the speed detected by an Ethernet interface.

In many deployments, systems use the interface attributes detected when an interface is initialized. As such, these attributes constitute operational state. However, there are usually provisions to overwrite the discovered attributes with static configuration data, like for example configuring the interface MTU to use a specific value or forcing an Ethernet interface to run at a given speed.

The system will record statistics (counters) measuring the number of packets, bytes, and errors received and transmitted on each interface.

4.3.2.3. Example 3: Account Information

Systems usually maintain static configuration information about the accounts on the system. In addition, systems can obtain information about accounts from other sources (e.g., Lightweight Directory Access Protocol (LDAP), Network Information Service (NIS)) dynamically, leading to operational state data. Information about account usage is an example of statistical data.

Note that configuration data supplied to a system in order to create a new account might be supplemented with additional configuration information determined by the system when the account is being created (such as a unique account id). Even though the system might create such information, it usually becomes part of the static configuration of the system since this data is not transient.

4.3.3. Implications

The primary focus of YANG is configuration data. There is no single mechanism defined for the separation of operational state data and statistics since NETCONF treats them both as state data. This section describes several different options for addressing this issue.

4.3.3.1. Data Models

The first option is to have data models that explicitly differentiate between configuration data and operational state data. This leads to duplication of data structures and might not scale well from a modeling perspective.

For example, the configured duplex value and the operational duplex value would be distinct leafs in the data model.

4.3.3.2. Additional Operations to Retrieve Operational State

The NETCONF protocol can be extended with new protocol operations that specifically allow the retrieval of all operational state, e.g., by introducing a <get-ops> operation (and perhaps also a <get-stats> operation).

4.3.3.3. Introduction of an Operational State Datastore

Another option could be to introduce a new "configuration" data store that represents the operational state. A <get-config> operation on the <operational> data store would then return the operational state determining the behavior of the box instead of its static and explicit configuration state.

4.4. Direction

At this time, the only viable solution is to distinctly model the configuration and operational values. The configuration leaf would indicate the desired value, as given by the user, and the operational leaf would indicate the current value, as observed on the device.

In the duplex example, this would result in two distinct leafs being defined, "duplex" and "op-duplex", one with "config true" and one with "config false".

In some cases, distinct leafs would be used, but in others, distinct lists might be used. Distinct lists allows the list to be organized in different ways, with different constraints. Keys, sorting, and constraint statements like must, unique, or when may differ between configuration data and operational data.

For example, configured static routes might be a distinct list from the operational routing table, since the use of keys and sorting might differ.

5. Security Considerations

This document discusses an architecture for network management using NETCONF and YANG. It has no security impact on the Internet.

6. References

6.1. Normative References

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

[RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, May 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

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

[RFC4742] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)", RFC 4742, December 2006.

[RFC4743] Goddard, T., "Using NETCONF over the Simple Object Access Protocol (SOAP)", RFC 4743, December 2006.

[RFC4744] Lear, E. and K. Crozier, "Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)", RFC 4744, December 2006.

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

[RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.

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

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

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, January 2011.

[RFC6110] Lhotka, L., "Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content", RFC 6110, February 2011.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, June 2011.

[SWEXPECT] "The Expect Home Page", <http://expect.sourceforge.net/>.

[W3CXPATH] DeRose, S. and J. Clark, "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>.

[W3CXQUERY] Boag, S., "XQuery 1.0: An XML Query Language", W3C WD WD-xquery-20050915, September 2005.

[W3CXSD] Walmsley, P. and D. Fallside, "XML Schema Part 0: Primer Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-0-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>.

[W3CXSLT] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xslt-19991116>.

6.2. Informative References

[RCDML] Presuhn, R., Ed., "Requirements for a Configuration Data Modeling Language", Work in Progress, February 2008.

Author's Address

Phil Shafer
Juniper Networks

EMail: phil@juniper.net