Network Working Group R. Presuhn, Ed.
Internet-Draft Retired
Intended status: Informational February 15, 2008
Expires: August 18, 2008

Requirements for a Configuration Data Modeling Language
draft-presuhn-rcdml-03

Status of this Memo

Copyright Notice

Abstract

This memo contains a compilation of requirements for a Configuration Data Modeling Language. Although focusing on the needs of the NETCONF Protocol, these requirements potentially have broader applicability. Comments should be sent to ngo@ietf.org.

Table of Contents

1. Introduction

Following discussions at the Vancouver IETF meeting (IETF 70), Dan Romascanu organized a design team to work on Requirements for a Configuration Data Modeling Language with participation from the Applications area as well as the Operations and Management area. This memo is that design team's product.

The principal goal of this memo is to increase the chances for a successful BOF in Philadelphia at the seventy-first IETF meeting. The goal of the BOF is to decide what needs to be done to support the development of data models for configuration in the IETF, focusing on the immediate requirements for the NETCONF protocol [RFC4741]. To expedite the process, the design team started with requirements gathered at previous IETF meetings where data modeling languages had been discussed, as well as collecting requirements from the various teams working in this space. These included both efforts to develop new solutions as well as ones reusing or extending existing data modeling languages and tools. The team identified the common requirements, as well as ones motivating particular choices made in specific solutions.

The focus of the requirements described here is on the immediate needs of the Operations and Management Area and the NETCONF protocol, and builds on precedent work, such as [RFC3535]. The design team recognizes that a data modeling language based on these requirements MAY have applicability beyond NETCONF, and that consequently the decision to support any of these requirements SHOULD always be considered in the light of the potential impact on extensibility and broader applicability envisioned.

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].

2. Previous Work

Numerous languages exist to describe data types and structured data in the abstract, with associated sets of rules to describe how that data can be transfered over networks. Important ones include:

- o ASN.1 (Abstract Syntax Notation One) with various encoding rules, such as
  - * BER (Basic Encoding Rules)
  - * XER (XML Encoding Rules)
- o XML (Extensible Markup Language) Schema

o RelaxNG (Regular Language for XML, New Generation)
o XDR (External Data Representation) [RFC4506]

All of these describe type systems and the structuring of information, with relatively little emphasis given to representing the information's semantics.

There have been numerous efforts over the years to develop languages with the additional semantics needed to be suitable for defining models for management data in general and configuration data in particular. Some of the more important ones include:

o GDMO (Guidelines for the Definition of Managed Objects) with GRM (General Relationship Model) extensions;
o SMIv2 (Structure of Management Information version 2) [RFC2578] with its facilities for defining Textual Conventions [RFC2579] and conformance Statements [RFC2580];
o SPPI (Structure of Policy Provisioning Information) [RFC3159];
o SMIng (Next Generation Structure of Management Information) [RFC3780];
o CIM (Common Information Model).

All of these presume a particular meta-model of the information to be managed. To varying degrees these are all associated with specific management protocol suites. None of these has yet gained serious support as a model definition language for the NETCONF environment, although some show pockets of use.

Efforts in the area of general-purpose information modeling have largely converged in the UML effort. There has been little interest in the IETF in using UML to standardize configuration data models. There have been previous efforts to define a protocol-neutral data modeling language in the IETF, including both requirements gathering [RFC3216] and language definition [RFC3780]. While true protocol neutrality is a laudable goal in theory, the experience documented in the work-in-progress [I-D.schoenw-sming-lessons] shows that this property taken to its extreme can have negative consequences, even for very similar underlying information models.

The design team recognizes that any effort like this will necessarily be incomplete, whether considered from the perspective of breadth or depth. The work-in-progress [I-D.linowski-netconf-dml-requirements] provides additional requirements along with their motivation.

3. Taxonomy of Requirements

The grouping of requirements in this section is merely one chosen by the design team for its convenience in helping separate easily-confused requirements. The major categories are:

1. Consequences of Netconf
2. Representation of Models
3. Reusability
4. Instance Data
5. Semantics of Models
6. Extensibility
7. Conformance
8. Techno-Political Constraints

The reader will notice that certain similar-sounding issues arise in multiple categories. For example, representation of multilingual information is a concern both for the representation of models as well as for the representation of instance data. The issues are included in both areas because they potentially afford different solutions.

The notations "Agreed" and "NOT Agreed" merely indicate whether the members of the design team agreed that the requirement in question was an appropriate requirement to impose on solution candidates. In cases, requirements were not agreed to despite being adequately met by all the proposals of which the design team was aware. The "minimize SMI translation pain" requirement was an example of this.

3.1. Consequences of NETCONF

Some requirements are direct consequences of the way the NETCONF protocol works, and might be inappropriate for a protocol-neutral configuration data modeling language. These are distinct from requirements driven by the NETCONF environment's information modeling needs. Whether addressed as part of the modeling language itself, or as part of a "binding" of that language for use with the NETCONF environment, a solution needs to address these concerns.

3.1.1. Notification Definition (Agreed)

The solution MUST support defining notifications. The solution MUST describe any transformations or adaptations necessary to transport notifications using the mechanisms defined in the work-in-progress [I-D.ietf-netconf-notification].

3.1.2. Notification Get (NOT Agreed)

The solution MAY support defining notifications in a way that minimizes duplication of definitions for polled or configured content. This reduces errors caused by the failure to maintain syntactic and semantic alignment of separate definitions. It also permits what is logically the same chunk of information to be sent via a get (or modified using edit-config) as would be conveyed asynchronously in a notification. There is disagreement on just how

much of a need there is for this kind of feature. Possible analogues which have been used to argue both for and against including this requirement include Syslog [RFC3164] and the SNMP Notification Log MIB [RFC3014].

There are three ways to define notifications:

1. Completely separate the definition of the data elements and their container from the definitions of the information objects defined for get and edit-config operations.
2. Re-use the element definitions from get and edit-config operations, but define a new container object for the notification itself (as with SNMP)
3. Re-use both element definitions and containers from the definition of content for get and edit-config operations. In this method, notifications can almost be said to not require defining. (This begs the question of what the semantics of the notification itself are bound to. One possibility might be to have a definition which consists simply of the binding of the notification semantic to whatever they payload container will be.)

The solution SHOULD describe whether and how each of these is handled. Some specific use cases for this are data change notifications as well as being able to store and send alarm definitions with the same definition. Note that the third option makes log records much easier than approaches (like that used in GDMO) which require the definition of an ad hoc notification syntax and a separate definition for the log record for each notification type, and that both the second and third options may simplify fine-grained access control, if that is required.

3.1.3. Locking (Agreed)

The solution MUST NOT preclude fine grained locking, as described for the NETCONF environment in the work-in-progress [I-D.ietf-netconf-partial-lock].

3.1.4. All Base Operations (Agreed)

The solution MUST unambiguously describe how all NETCONF [RFC4741] base operations work with data defined under a model produced using the solution. This includes both how information appears on the wire as well as any effects on the configuration data store.

3.1.5. Define new NETCONF Operations (Agreed)

The solution MUST provide a means to define new NETCONF operations and their parameters (base, vendor extensions, and so on) in the same

language as is used for defining models. One could argue that this is a layering violation, however there is a clear need for this function in the NETCONF space, and the members of the design team felt there was no particular advantage to requiring a separate mechanism.

3.1.6. Separation of Operations and Payload (Agreed)

If the solution provides a means for defining new NETCONF operations, it MUST allow a clear separation between data model definitions and the definition of new NETCONF operations. This requirement exists to mitigate that apparent layering violation caused by the requirement to be able to define new NETCONF operations.

3.1.7. Error Annotation (Agreed)

The solution MUST be able to define specific error messages for a given element. Note that this could interact with support for internationalization and localization. The solution MUST describe how specific error strings are associated with error conditions as required by the NETCONF protocol.

3.1.8. No Mixed Content (Agreed)

The solution MUST prevent mixed content, i.e., tags and data mixed together as part of a value, other than to treat it as opaque information. This requirement is a consequence of the NETCONF protocol environment.

3.2. Model Representation Requirements

The requirements in this section are all connected to the mechanics of how models are represented and manipulated, rather than what they express. One issue which appeared in earlier versions of this memo, model canonicalization, straddles the border between this grouping and the group on model semantics. It was in this group because the use case for it is associated with others in this group, such as performing a "diff" between versions of a model. However, the consensus of the design team was that this was not a requirement, so it was been removed.

3.2.1. Human Readable (Agreed)

The solution MUST support a human-readable representation of data models. This requirement is independent of how an instance of a model is represented.

3.2.2. Machine Readable (Agreed)

The solution MUST support a machine-readable representation of data models.

3.2.3. Textual Representation (Agreed)

The solution MUST support a text-based representation for models. It MAY support other representations. It MUST be possible to represent model definitions as ASCII text in 72 columns so standard data models can be included in RFCs. This requirement is independent of how an instance of a model is represented.

3.2.4. Document Information (Agreed)

The solution MUST provide a means to specify document information for a data model, such as when it was created, its revision history, point of contact, and so on.

3.2.5. Ownership and Change Control (Agreed)

It MUST be clear who exercises change control and ownership over the data modeling framework, e.g., the IETF. This MUST also be clear for the technologies on which it depends.

3.2.6. Dependency Risk Reduction (Agreed)

In cases where a proposed solution depends on other specifications, there MUST be a way to reference the specific versions required, in case that specification evolves in incompatible ways. This requirement is motivated by bad experiences with how the ASN.1 specification mutated after its first version.

3.2.7. Diff-Friendly (Agreed)

It MUST be possible for an operator using existing tools such as "diff" to determine what has changed between two versions of a data model.

3.2.8. Internationalization and Localization

There are several requirements associated with issues related to languages and character sets in the representation of models. Note that, with the exception of literal strings, these are distinct from the questions about what appears in an instance of a model.

3.2.8.1. Descriptions using Local Languages (Agreed)

The solution MUST be able to support the use of Unicode text in a model definition to provide human readable descriptions of information semantics. This is effectively required by [RFC2277]. This is not the same thing as requiring a mechanism for the internationalization and localization of models, but rather a way of allowing the model definer to work in his or her preferred language.

3.2.8.2. UTF-8 Encoding (Agreed)

It MUST be possible to encode model definitions using UTF-8. This is effectively required by [RFC2277] as a consequence of the need for a textual representation and the need to be able to include descriptive text in the model definer's language of choice.

3.2.8.3. Localization Support (Agreed)

The solution MUST outline how localization of an existing model would proceed. The strong preference of members of the design team was to treat model localization as a user interface issue. The solution MUST be in alignment with the guidance given by [RFC2277].

3.2.8.4. Error String Localization (Agreed)

The solution MUST NOT preclude localization for user display of NETCONF error strings defined in conjunction with a data model. (Since the NETCONF protocol itself does not provide for language negotiation, such localization would presumably take place within the NETCONF client. The question is how to associate an error string defined as part of a model with its localization.)

3.2.8.5. Tag Names and Strings in Local Languages (NOT agreed)

The solution MAY support the use of Unicode text for tag names and strings in definitions. Note that this is not a question of internationalization and localization, but rather the use of Unicode for what are effectively protocol elements in an instance document for a model defined using the solution. Even if a solution does support the use of Unicode text for tag names and strings in definitions, it SHOULD provide guidance on the use of an ASCII subset for tags, since specifications for publication in an RFC will almost certainly need to be written with such a restriction in mind. Note also that this requirement can interact badly with solutions that use labels taken directly (without any mapping) from models to generate code in programming languages, which may have severe restrictions on identifiers.

3.3. Reusability Requirements

The reusability requirements could also be thought of as a sub-category of the semantic richness requirements, since they focus on the expressiveness of the modeling language. Their inclusion as a top-level category of requirement reflects their importance to standards-making.

3.3.1. Modularity (Agreed)

The solution MUST provide a means to define data models in discrete pieces, and support the publication of portions of models in separate files.

3.3.2. Reusable Definitions (Agreed)

The solution MUST support a way to reuse definitions from another model. A type definition is a type of reusable definition. Note that this potentially interacts with requirements to be able to revise or extend a model.

3.3.3. Modular extension (Agreed)

It MUST be possible to extend a published data model without modifying the original data model. Those specifying solutions are advised to carefully consider how this capability might interact with the ability to revise definitions and the ability to reference definitions in other models.

3.4. Instance Data Requirements

This section contains specific requirements on what instance documents for models defined using the solution will look like. These are largely independent of requirements for what the modeling language would look like, but do interact with the semantic capabilities of the language.

3.4.1. Default Values on the Wire (Agreed)

The solution MUST specify how default values affect what is and is not explicitly encoded in an instance document. Possibilities include a default-free data modeling language, protocol-specific default handling, or protocol-independent prescribed behavior.

3.4.2. Ordering

The common factor in this group of requirements is the ordering of pieces of data on the wire. The cases differ in what is being ordered, and whether the ordering of itself conveys information. The order in which things can appear in an instance document potentially affects not only the complexity of the code to validate or parse it, but can also interact with attempts to extend the corresponding models.

3.4.2.1. Ordered Lists (Agreed)

The solution MUST support the ability to specify whether the order of list entries in an instance document has semantic significance, and MUST consequently be preserved. An example of such a list might be a list of access control rules. An example of a list where the order would have no semantic significance might be a list of users. In cases where a list's order would have no semantic significance, the solution SHOULD nonetheless specify whether order is maintained, since this would affect instance data canonicalization.

3.4.2.2. Order within Containers (NOT Agreed)

A solution MAY support the ability to specify that in an instance of a data model, the order of child elements within a containing element needs to be preserved, due to semantics, backward compatibility requirements, or processing efficiency considerations.

3.4.2.3. Interleaving (NOT Agreed)

A solution MAY support the ability for a model to allow interleaving elements in the XML representation of an instance of a data model. This means that those child elements MAY appear in any order on the wire.

3.4.3. Validation

This section contains requirements pertaining to the validation of an instance document. The term "valid" means more than merely well-formed, and in the context of network management it includes the notion of conforming to the semantics of a schema. A solution proposal MUST describe precisely what is meant when by terms like "valid" and "well-formed".

3.4.3.1. Validate Instance Data (Agreed)

A solution proposal MUST provide sufficient detail to allow a determination to be made whether an instance of a data model is well-formed and valid in terms of the schema, as well as any additional considerations.

3.4.3.2. Tools to Validate Instance Data (NOT Agreed)

A solution MAY provide a means of determining whether an instance document is a valid instance of a model. (The difference between the previous requirement and this one is that the former (agreed) merely requires that a solution is known, while the latter (not agreed) requires that a solution have been implemented.)

3.4.4. Instance Canonicalization (Agreed)

The solution MUST describe how to produce a canonicalized version of the instance. This is a transform which can put the data in a form which is suitable for comparison. See "Canonical XML Version 1.0" [RFC3076] for more information. This does not imply that there is any requirement for data on the wire to be sent in canonicalized form. The design team recognizes that there is a point of diminishing returns in canonicalizing human-readable strings, and advises those proposing solutions to consider the trade-offs and not get carried away.

3.4.5. Character Set and Encoding (Agreed)

The solution SHOUL support the creation of models which can handle human-readable information in any language in an instance document. In keeping with [RFC2277], this means that models MUST be able to handle UTF-8 data appropriately.

3.4.6. Model Instance Localization (NOT Agreed)

Tags and other human-readable portions of an instance of a model SHOULD be localizable. Underlying this requirement is the question of whether tags in an instance document are really "human-readable" or merely protocol elements. This requirement needs clarification: is it a question of what an instance document looks like between a NETCONF client and server, or is it a question of how an instance document might be transformed for presentation to a user?

3.5. Semantic Richness Requirements

The requirements in this section are all related to the need to describe information models in more than purely syntactic terms.

3.5.1. Human-Readable Semantics (Agreed)

The solution MUST provide a means for the developer of an information model to associate human-readable text with components of that model, in order to describe their semantics and use.

3.5.2. Basic Types (Agreed)

The solution MUST define frequently used types. This could be accomplished in a standard data model definition as part of a solution or part of the language definition, for example.

3.5.3. Handling Opaque Data (Agreed)

It MUST be possible to perform certain operations on opaque data. This means that completely replacing the data would be supported, but not merging, for example. This data potentially does not conform to any schema definition, but may happen to be well-formed XML within the opaque data.

In light of the negative experience with the opaque date type in SNMP, one might regard this requirement as a "necessary evil."

3.5.4. Keys

The issues in this section all relate to the problem of uniquely identifying an entity in an instance of a model. A solution SHOULD spell out whether and how it supports creating models which permit instances which are not uniquely identified using keys.

3.5.4.1. Define Keys (Agreed)

The solution MUST support defining keys to data (i.e. the list of fields which uniquely identify an element, or a specially created identifier like an ifIndex or an XML id.)

3.5.4.2. Deep Keys (NOT Agreed)

The solution SHOULD support using as a key the value of an element which is not an immediate descendant of the element being "keyed".

```
<staticRoutes>
  <staticRoute>
    <prefix>
      <address>1.2.0.0</address>
      <length>16</length>
    </prefix>
    <nextHop>5.6.7.8</nextHop>
  </staticRoute>
  ...
</staticRoutes>
```

Here the desired key is constructed using both address and length. Though it could also have been be modeled with prefixAddress and

prefixLength, doing so this would prevent reuse of the prefix data type. This example actually illustrates two distinct features: "deep" keys and compound keys. Support for one does not imply support for the other.

3.5.5. Relationships

Issues discussed with respect to relationships included:

- o the tools available for modeling relationships in the solution;
- o the syntactic means available for representing relationships in configuration data;
- o the kinds of constraints on relationships that can be expressed in a model.

On the issue of constraints, the design team found it helpful to consider them as conceptually belonging to various phases, characterized by the kind of information needed to determine whether a particular relationship was "valid" or not. Specifically:

- o constraints that could be enforced purely syntactically, as in the case, for example, of a foreign key (like ifIndex) being of the correct type;
- o constraints that could be evaluated in the context of a candidate configuration as it was "under construction";
- o constraints requiring a "complete" candidate configuration to evaluate;
- o constraints requiring a running system to evaluate, such as a relationship to a physical piece of hardware;
- o constraints which can only be evaluated in a network context, such as a relationship to a peer entity on another system.

It's important to note that these kinds of validation are far beyond what is normally understood as XML validation, but have been long-standing the practice in network management.

3.5.5.1. Simple Relationships (Agreed)

The solution MUST support defining cross references between elements in different hierarchies. This SHOULD be a formal machine readable definition rather than the value of an implicitly known field. The solution SHOULD support defining reference pointers for both 1:1 and 1:n relationships.

3.5.5.2. Many-to-Many Relationships (NOT Agreed)

The solution SHOULD support defining many to many cross reference relationships.

3.5.5.3. Retrieve Relationships instance (NOT Agreed)

The solution SHOULD support a means of specifying relationships which can be represented in a configuration or on the wire with minimal redundancy. Knowledge of the model would be needed to transform this representation into a fully qualified instance identifier. The use of an integer interface index in a MIB, rather than a rowPointer [RFC2579], would be analogous to this capability. Ideally, the knowledge required to generate the fully qualified instance identifier would be present in the model in machine-readable form.

3.5.5.4. Retrieve Relationships - qualified (NOT Agreed)

The language SHOULD support the ability to learn about specific relationships in the instance document, including some context. For example, if there is a general relationship between a card and a port, it SHOULD be possible in the instance document to learn that it is specifically card 12 that is involved in the relationship and the information would be provided in a more descriptive manner to enable some interpretation of in the absence of the schema - by being displayed to a user as text for example.

3.5.6. Hierarchical Data

The solution MUST support defining data in hierarchies of arbitrary depth. This enables closer modeling of data to real world relationships, such as containment.

```
<device>
    <shelf>
        <card>
            <port>
               <subPort/>
            </port>
        </card>
    </shelf>
</device>
```

3.5.7. Referential Integrity

The issues related to referential integrity can consume a significant amount of time. There are significant differences in expectation regarding what is meant by "referential integrity checking," and the following sub-sections reflect those differences.

3.5.7.1. Referential Integrity (NOT Agreed)

It SHOULD be possible to describe in the modeling language that validation (at configuration create / merge time) of data cross references is required for a given piece of the model. For example, it SHOULD be possible to verify that related data exists, and reject a configuration if it is does not. Note this is only performed when a NETCONF operation is being done. There is no requirement to maintain this integrity over time and report issues. Cross references are only within a given device's configuration (see also extended referential integrity requirement).

This kind of checking will not always be possible while a candidate configuration is under construction, since there may be cycles in relationships which prevent checking of an incomplete instance document.

3.5.7.2. Extended Referential Integrity (NOT Agreed)

It SHOULD be possible to support more complex validating of instance data cross references. Examples, pre-provisioning, validating against unreachable resources (not just configuration data present on the device - non-configuration data on this device or configuration on other devices)

3.5.7.3. Referential Integrity Robustness (NOT Agreed)

The solution SHOULD provide a means of indicating that the presence of data cross reference which cannot be verified, or which is known to not exist at the moment a configuration becomes the active configuration, is nonetheless not to be considered a configuration error. An example of a use case would be a relationship configured with respect to a hot-swappable component, which potentially can be absent from the system being configured when the configuration is made active.

3.5.8. Characterize Data (Agreed)

The solution MUST be able to model configuration data. The solution MUST be able to model non-configuration data, such as status information and statistics. The solution MUST support characterizing data definitions in a model as configuration or non-configuration. The solution MAY support further characterization, such as identifying status or statistics.

3.5.9. Defaults

Discussion on IETF mailing lists as well as among the members of the design team has made it clear that there are many different ways of understanding "default" and correspondingly many different use cases. However, among the members of the design team, we were able to reach agreement that if a notion of "default" is to be at all useful in the context of configuration management, where knowing precisely what the configuration of a system is, and being able to bring the configuration of a system to a precisely known state are essential, "default"needs to be understood in terms of a binding contract between client and server.

A consequence of this understanding of defaults would be to minimize their use in standardized definitions, since experience has shown that there are relatively few cases where a truly standard default is possible. Furthermore, vendor experience with defaults has shown that they can be problematic to versioning; what is an appropriate default in one version of a product might not make sense in a later release. Consequently, it might make sense to assume that if defaults are supported, their primary application would be as extensions to a model reflecting a particular implementation.

3.5.9.1. Default Values (NOT Agreed)

The solution SHOULD support defining static default values for elements. In the content of NETCONF, this is understood to mean that in an edit-config "create" request, if the client does not provide the value, the server will assume the specific value defined in the model as the default. The solution SHOULD specify how this feature interacts with backwards compatibility, canonicalization, and any other NETCONF operations. The solution SHOULD specify the implications for claims of conformance to a default if the server uses a different default value.

3.5.9.2. Dynamic Defaults (NOT Agreed)

The solution must support dynamic default values. These are defaults whose value depends on the value of other fields. For example, if the disk size is 2 gigs then the maximum file size is 1 meg, but if the disk is 1 meg, then the maximum file size is 1 K.

3.5.10. Formal Constraints

As with relationships, one way of conceptualizing the validation of formal constraints is to think of them in terms of what information is needed to determine whether a particular constraint is satisfied. Roughly, these are:

- o syntactic constraints determined by the base data type
- o range and pattern constraints which can be evaluated without considering any other configuration or status data
- o constraints which can be evaluated in terms of other information present in a candidate configuration, but which do not require that the complete configuration be present
- o constraints which can be evaluated in the context of a complete candidate configuration
- o constraints which require knowledge of non-configuration data expressed in the model
- o constraints which can only be evaluated in a running system
- o constraints which require knowledge of resources elsewhere in the network

3.5.10.1. Formal Description of Constraints (Agreed)

It MUST be possible to specify constraints on the value of an element such as uniqueness, ranges, patterns, etc. These constraints can be evaluated in isolation and not related to other elements.

3.5.10.2. Multi-element Constraints (NOT Agreed)

A solution MAY provide a means to define constraints on an element which involve another element of the configuration. An example would be where the value of an MTU depends on the ifType. Additional use cases might include dependencies on some non-configuration data, such as presence of a particular piece of hardware, or inter-system constraints.

3.5.10.3. Non-Key Uniqueness (Agreed)

The solution MUST provide a way to specify constraints on uniqueness of non-key data elements. The scope of the uniqueness MUST be specified (parent, device, etc.) The extent of checking and enforcement needs to be spelled out. The solution MUST spell out whether, for a model to be valid, this constraint always holds true, or is it only required to be true at the time the configuration is created or merged.

3.5.11. Units (Agreed)

The solution MUST provide a means of associating units with values, since unit errors in the configuration of values have potentially catastrophic consequences.

3.5.12. Define Actions (NOT Agreed)

The solution MAY provide a way to define specific actions to be performed. Here "actions" are understood as being associated with specific elements (objects) of information models. This requirement is distinct from the requirement to support the addition of new "operations", even though the two may be indistinguishable at the level of the protocol. If supported, the solution MUST describe how these are mapped into the NETCONF protocol. One of the motivations for this feature is the desire to avoid mapping action semantics onto data, as was necessary in the SNMP SMI [RFC2578]. The resulting "go buttons" proved unpopular. Another concern with this kind of feature is how to accommodate actions in a fine-grained access control framework, if one is ever developed.

3.6. Extensibility Requirements

There are two broad categories of extensibility requirements: those having to do with the modeling language, and those related to the extensibility of models defined using that language.

3.6.1. Language Extensibility

This section lists requirements concerned with the extensibility of the modeling language itself, rather than of models defined using that language.

3.6.1.1. Language Versioning (Agreed)

The modeling language itself MUST be versioned. This requirement is motivated by the requirements for language extensibility below.

3.6.1.2. User Extensions (NOT Agreed)

It SHOULD be possible for the users to extend the language. This means the ability of the user of the data modeling language, to add new statements or functionality to the language.

It is useful for two things:
- o standards evolution;
- o proprietary / vendor-specific annotations.

Extensibility SHOULD be done in a way that unknown extensions MAY be ignored.

3.6.1.3. Mandatory Extensions (NOT Agreed)

The solution SHOULD support defining language extensions which the solution MUST understand; a tool which does not understand one of

these modeling language extensions MUST treat it as an error.

3.6.2. Model Extensibility

The requirements in this section concern the extensibility of specific models.

3.6.2.1. Model Version Identification (Agreed)

Different versions of a given schema MUST be unambiguously identified. This assumes that the schema itself can be uniquely identified.

3.6.2.2. Interaction with defaults (NOT Agreed)

The solution SHOULD define interaction of model definition with defaults. What happens when defaults are added to model or whether a default can be changed.

3.6.2.3. Conformance Interference (NOT Agreed)

The solution SHOULD define how revising a model interacts with claims of conformance to its earlier versions, as well as what the impact is on claims for conformance to other models which have re-used definitions from the earlier version.

3.6.2.4. Obsolete Portions of a Model (Agreed)

The solution MUST provide a way to signify that elements of a schema are obsolete.

3.6.3. Instance Data Extensibility

These requirements deal with the impact on the instance data of modifications to the model. Use cases for some of the requirements discussed in this section are described in Appendix B.1.

3.6.3.1. Schema Version of Instance (NOT Agreed)

The solution SHOULD provide a means to determine what schema version was used to generate an instance document.

3.6.3.2. Interaction with default Values (NOT Agreed)

The solution SHOULD define its interactions with default values in the instance, if supported. How does the fact that something is defaulted show up on the wire and what happens when defaults are added, removed or modified, for example.

3.6.3.3. Backwards Compatibility (Agreed)

The solution MUST support the ability to extend the model and still be usable to use it. A NETCONF client familiar with an older version of the schema should still be able to function. An old client should be able to work with a new server.

3.6.3.4. Forwards Compatibility (NOT Agreed)

The solution should support the ability to extend the model and still interoperate with older versions. A NETCONF client employing a newer version of the schema should still be able to function with a server using an older version.

3.6.3.5. Must-Understand Model Extensions (NOT Agreed)

The solution should support defining model extensions which the client MUST understand or otherwise error. Adding mandatory objects to an update to a Schema for example.

3.7. Talking About Conformance

This section contains several requirements, all tied to the question of what it means to claim conformance. The two major categories are:

- o Conformance to the modeling language
- o Conformance to a specific data model

Discussion of issues related to conformance to a specific data model can also be further refined into questions of what "conformance" means for a NETCONF server and what it means for a NETCONF client.

3.7.1. Conformance to the Modeling Language (NOT Agreed)

A solution MUST spell out what is meant by "conformance" to that particular modeling language specification. This requirement is motivated by the need to evaluate whether or not a tool supports the chosen solution.

3.7.2. Conformance to a Model (Agreed)

When a solution is used to define specific data models, it is important to be able to know what is meant by a claim of "conformance" to a particular model, both from the perspective of a client implementation and of a server implementation. The solution SHOULD support indicating conformance requirements for a Schema. There is not a requirement that conformance requirements would be stated separately from the model

3.7.2.1. Conditional Conformance (NOT Agreed)

The solution should provide a means of providing conditional compliance. If this is MPLS, then you need the following stuff supported, for example.

3.7.2.2. Server Conformance to Schema (Agreed)

The solution MUST support a method of indicating whether support of an object is required in order to claim conformance to a particular schema.

A solution MUST spell out whether a means for specifying server conformance to a schema exists. If such a means exists, it SHOULD allow an automated determination of the elements (and possibly subtypes or extensions) which MUST be processed (and not merely ignored) by a server handling a NETCONF edit-config create or merge operation.

3.7.2.3. Client Conformance To Schema (NOT Agreed)

The solution should support a method of indicating whether presence of an object is required in order to be a valid configuration. This has been explained as "mandatory to use (in a create) as NETCONF client as opposed to mandatory to implement on NETCONF server", but this explanation begs the question of what the server that doesn't implement the object is supposed to do with the create request in which the object is required to be present.

A solution MUST spell out whether a means for specifying client conformance to a schema exists. If such a means exists, it SHOULD allow an automated determination of the elements (and possibly subtypes or extensions) which MUST be processed (and not merely ignored) by a server handling the response to NETCONF get-config operation.

Note that one could formulate this in terms of what is sent in an edit-config operation, but that could be problematic if the solution supports some types of defaults.

3.7.2.4. Versioned Conformance (NOT Agreed)

The solution should provide a means of specifying what is required for compliance when the schema is updated. One of the motivations for this requirement is the need to know both what conformance to the current version of a schema entails, as well as what conformance to a previous version would entail. This becomes particularly important when definitions are incorporated by reference in another model,

which is itself subject to revision.

3.8. Techno-Political Constraints

The requirements in this section arise mainly from the relationship of this work to other standards, and technical constraints motivated by factors other than the need to define network configuration management data.

3.8.1. Standard Technology (NOT Agreed)

The solution SHOULD leverage existing widely used language and tools to define the NETCONF content, redefining as little as possible the work that w3c and other relevant bodies have already done.

3.8.2. Translate Models to Other Forms (Agreed)

The solution MUST support the ability to translate a model definition to RelaxNG and XML Schema. Any proposed solution MUST describe whether this translation is lossy or lossless and if lossy, what information is lost.

3.8.3. Minimize SMI Translation Pain (NOT Agreed)

Minimize translation pain from SMI into NETCONF content. Translation of NETCONF content into SMI is not a consideration. Disagreement about this requirement stems from concern about the possibility that this might be interpreted as requiring the perpetuation of SMI-style models, rather than merely accommodating the basic types, as described in the work-in-progress [I-D.ietf-opsawg-smi-datatypes-in-xsd].

3.8.4. Generate Models from Other Forms (NOT Agreed)

The solution SHOULD support higher level modeling languages. An example would be generating configuration data models from UML descriptions. This requirement gets interesting regarding the question of whether anything needed in a network configuration data model might be impossible to represent in UML in machine-readable form.

3.8.5. Isolate Models from Protocol (NOT Agreed)

The solution, and data models developed using the solution, SHOULD NOT be too tightly coupled to the NETCONF protocol. It should be possible to evolve the NETCONF protocol and data models independently. One use case is that it should also be possible to transport the data model instance (NETCONF content) over other

protocols, such as FTP.

3.8.6. Library Support (NOT Agreed)

The solution SHOULD have wide support from development languages C, etc. The element of disagreement among the members of the design team is whether the evaluation of a solution depends on the existence or on the feasibility of such support.

3.8.7. RFC 3139 Considerations

[RFC3139] defines "Requirements for Configuration Management of IP-based Networks", which should be taken into consideration when identifying a solution for use with NETCONF. Note that it is possible that not all of these requirements will necessarily be applicable to the current problem.

3.8.8. RFC 3216 Considerations

[RFC3216] defines "SMIng Objectives", which should be taken into consideration when identifying a solution for use with NETCONF. Note that not all of these requirements will necessarily be applicable to the current problem.

4. Requirement Interactions

Numerous interactions are identified in conjunction with individual requirements. In general, the interactions between re-use, extensibility, and conformance demand special attention.

5. IANA Considerations

This document requires no action by IANA. Specifications based upon these requirements will specify what, if any, IANA considerations are appropriate.

6. Security Considerations

Although this document only lists requirements, some of these requirements have security implications. For example, the requirements for modular definitions open up the consideration of possible attacks based on the modification of a component of a model which is not under the control of the model developer. Other concerns include robustness when an instance document conforms to a model different from the one to which it purportedly conforms. The evaluation of constraints specified in a model, which require examining other configuration data, possibly even data from other systems, opens another possible avenue of attack for consideration. The kinds of models that can be defined by a solution will have implications for an access control framework. Finally, as has long been known, compilers and code generation tools are themselves open to attack.

7. References

7.1. Normative References

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

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

7.2. Informative References

[I-D.ietf-netconf-notification]
Chisholm, S. and H. Trevino, "NETCONF Event Notifications", draft-ietf-netconf-notification-11 (work in progress), November 2007.

[I-D.ietf-netconf-partial-lock]
Lengyel, B. and M. Bjorklund, "Partial Lock RPC for NETCONF", draft-ietf-netconf-partial-lock-00 (work in progress), January 2008.

[I-D.ietf-opsawg-smi-datatypes-in-xsd]
Natale, B. and Y. Li, "Expressing SNMP SMI Datatypes in XML Schema Definition Language", draft-ietf-opsawg-smi-datatypes-in-xsd-00 (work in progress), February 2008.

[I-D.linowski-netconf-dml-requirements]
Linowski, B., Storch, M., Ersue, M., and M. Lahdensivu, "NETCONF Data Modeling Language Requirements", draft-linowski-netconf-dml-requirements-00 (work in progress), January 2008.

[I-D.schoenw-sming-lessons]
Schoenwaelder, J., "Protocol Independent Network Management Data Modeling Languages - Lessons Learned from the SMIng Project", draft-schoenw-sming-lessons-01 (work in progress), September 2007.

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

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3014] Kavasseri, R., "Notification Log MIB", RFC 3014, November 2000.

[RFC3076] Boyer, J., "Canonical XML Version 1.0", RFC 3076, March 2001.

[RFC3139] Sanchez, L., McCloghrie, K., and J. Saperia, "Requirements for Configuration Management of IP-based Networks", RFC 3139, June 2001.

[RFC3159] McCloghrie, K., Fine, M., Seligson, J., Chan, K., Hahn, S., Sahita, R., Smith, A., and F. Reichmeyer, "Structure of Policy Provisioning Information (SPPI)", RFC 3159, August 2001.

[RFC3164] Lonvick, C., "The BSD Syslog Protocol", RFC 3164, August 2001.

[RFC3216] Elliott, C., Harrington, D., Jason, J., Schoenwaelder, J., Strauss, F., and W. Weiss, "SMIng Objectives", RFC 3216, December 2001.

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

[RFC3780] Strauss, F. and J. Schoenwaelder, "SMIng - Next Generation Structure of Management Information", RFC 3780, May 2004.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

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

Appendix A. Acknowledgments

This members of the design team that produced this memo were:
Martin Bjorklund
Sharon Chisholm
Alex Clemm
Rohan Mahy
Chris Newman
David Partain
Randy Presuhn
The editor would like to thank everyone for the time, thought, and text they contributed to make this memo possible.

Many of the ideas gathered here were extracted from discussions held at various IETF meetings and on IETF mailing lists.

The members of the design team would also like to thank those who have provided helpful comments on earlier versions of this memo:
Andy Bierman
Balasz Lengyel
David Harrington
Eric Rescorla

Appendix B. Use Cases

The use cases presented here were chosen to illustrate requirements which proved contentious or were not agreed by the members of the design team or otherwise needed clarification. For requirements which were not contentious, we did not specifically devise use cases.

B.1. Multi-Version Scenarios

The following discusses typical deployment scenarios behind backwards and forwards requirements described in Section 3.6.3.

B.1.1. Tightly Coupled

This is the simplest case. This is where the application which manages the device is upgraded at the same time as the devices and only needs to worry about managing a single version of a single type of network element. Tightly coupled solutions can be costly and impractical, so are not a realistic solution to the problem of version management.

B.1.2. Loosely Coupled - Support Matrix

It is more typical that a single application is required to support different types of network elements and often more than one version of that network element. A support window of the current version and some set of older versions is commonly assumed. It should be possible to support multiple versions of the same schema with as few changes to application code (including data-driven configuration) as possible.

B.1.3. Forwards Compatibility

There are a number of reasons that the management application might not be upgraded when network elements are upgraded, including being developed by a third party releasing on a different cadence. In this case, the earlier version of the management application should be able to continue to provide the same level of support against the newer versions of the schema as it did in the older version.

B.1.4. Mixed-Mode Forwards and Backwards Compatibility

In mixed mode, different network elements, all supporting different versions of the schema are present. There are also different applications in the network, which each support different versions of the schema.

All the applications should be able to support all the versions of the schema. As in the case of forwards compatibility, best effort support will be provided.

B.1.5. Multiple Extensions

This case is the same as the mixed-mode case above, except that different network element support zero, one or more extensions to the model.

Appendix C. Example Instance Documents

The instance documents presented here are examples to illustrate specific requirements or their possible consequences.

```
<?xml version="1.0" encoding="UTF-8"?>
<dhcp xmlns="http://example.org/ns/dhcp"
      xmlns:cal="http://example.org/ns/cal">

  <!-- one simple validation constraint is that the default-lease-time
```

```
         can't be larger than max-lease-time. These times are in
         seconds -->
    <default-lease-time>600</default-lease-time>
    <max-lease-time>7200</max-lease-time>

      <!-- There is a list of subnets followed by a list of
           shared-networks.  In this instance document some lists have
           their own parent container and some do not -->
      <subnet>
        <!-- The key to the list of subnets is the combination of the
             network and prefix-length elements. -->
        <!-- validation: check that none of the subnets overlap -->
        <network>10.254.240.0</network>
        <prefix-length>22</prefix-length>

        <!-- A subnet has an optional range definition. If a range is
             given, it means that the clients on the subnet get
             addresses dynamically. If a range element is present, a low
             and high address MUST be specified. dynamic-bootp is
             optional. -->
        <range>
          <dynamic-bootp/>
          <low>10.254.240.10</low>
          <high>10.254.240.230</high>
        </range>

        <!-- Next is a collection of DHCP options -->
        <dhcp-options>
          <!-- router-list is a list of routers where the order is
               semantically significant -->
          <router-list>
            <router>10.254.240.1</router>
            <router>10.254.240.2</router>
          </router-list>
          <!-- Here the data model was extended to support a new DHCP
               option in the cal namespace -->
          <cal:timezone>US/Indiana</cal:timezone>
        </dhcp-options>

        <!-- This is a per-subnet maximum lease time -->
        <max-lease-time>1200</max-lease-time>

        <!-- The leases top level container contains status information
             about active leases.  This container cannot be present in a
             configuration operation (ex: Netconf set/get-config) -->
        <leases>
          <!-- The IP address attribute is the key to the lease -->
          <lease ip-address="10.254.240.12">
```

```
          <starts>2008-10-10T08:00:00Z</starts>
          <ends>2008-10-10T08:20:00Z</ends>
          <mac-address>00:11:22:33:44:55</mac-address>
        </lease>
        <lease ip-address="10.254.240.47">
          <starts>2008-10-10T08:04:22Z</starts>
          <ends>2008-10-10T08:24:22Z</ends>
          <mac-address>11:22:33:44:55:66</mac-address>
        </lease>
      </leases>

      <!-- The DHCP server will only respond to requests for this subnet
           when they come from one of these interfaces -->
      <interface-filter>
        <!-- lo0 and en2 are keyrefs.  They refer to a key in the
             ifName element in a list of interfaces in a module with
             the http://example.com/ns/int namespace -->
        <interface>lo0</interface>
        <interface>en2</interface>
     </interface-filter>
    </subnet>

    <!-- Shared networks contain one of more subnets. The DHCP server
         can provide addresses on any subnet in the shared network.
         For the sake of this example, assume that the name attribute
         is the key to the list of shared-networks.  The key could be
         an XML ID in a real instance. -->
    <shared-network name="Lab network">
      <subnet>
        <network>10.17.224.0</network>
        <prefix-length>24</prefix-length>

        <range>
          <low>10.17.224.10</low>
          <high>10.17.224.250</high>
        </range>
        <dhcp-options>
          <router-list>
            <router>10.17.224.1</router>
          </router-list>
          <!-- domain-list is a list of domains where order is not
               semantically significant -->
          <domain-list>
            <domain>example.org</domain>
            <domain>example.net</domain>
          </domain-list>
          <!-- custom options can either contain a single IPv4 address
               (expressed as four octets in the DHCP packet) or an
```

```
               opaque string.  The option attribute is the key. -->
          <custom option="150">
            <ip-address>192.168.45.29</ip-address>
          </custom>
          <custom option="171">
            <string>192.168.45.29,foo.example.com</string>
          </custom>
        </dhcp-options>
      </subnet>
      <subnet>
        <network>10.0.29.0</network>
              <prefix-length>24</prefix-length>
        <range>
          <dynamic-bootp/>
          <low>10.0.29.10</low>
          <high>10.0.29.230</high>
        </range>
        <dhcp-options>
          <router-list>
            <router>10.0.29.1</router>
          </router-list>
        </dhcp-options>
      </subnet>
    </shared-network>
</dhcp>
```

Author's Address

Randy Presuhn (editor)
Retired

Email: randy_presuhn@mindspring.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).