NETCONF Data Modeling Language Requirements
draft-linowski-netconf-dml-requirements-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on July 31, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2008).
Abstract

This document collects requirements for a Data Modeling Language (DML) and has been prepared as a contribution to the RCDML design team working currently on the "Requirements for a Configuration Data Modeling Language". Comments can be sent to ngo@ietf.org.
1. Introduction

This document is a working document and collects requirements for a Data Modeling Language (DML) and has been prepared as additional contribution for the design team currently working on the "Requirements for a Configuration Data Modeling Language" (RCDML) [4].

RCDML design team is preparing a requirements document by focusing on the immediate needs of the OPS area and NETCONF protocol as well as by taking into consideration the need for extensibility and the opportunity of providing one data modeling language solution for different other IETF problems in the APPS area. We think the requirements in this document are complementary to the requirements discussion until now and should be seen as an additional input for the design team work on the RCDML.

The requirements in this document are derived from a modeling language (O2ML) which is in use at NSN to address various aspects of network management, especially Configuration Management. The requirements have been prepared based on the experience with O2ML over the last two years.
2. Conventions used in this document

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 RFC 2119 [1].
3. Terminology

Following is the terminology used in this document:

- **annotation**: Additional metadata that refines semantics of a model element. A typed annotation consists of elements which have a type and a value.

- **attribute**: A named data element that can hold a value (structural feature).

- **attribute group**: Group of attributes supposed to be used only to define the contents of classes (or structures).

- **base type**: The type from which a refined type was derived, which may be either a built-in type or another derived type.

- **built-in type**: A data type defined in the DML, such as uint32 or string.

- **class**: A language construct used to describe the structural features of instances with an own identity and life-cycle.

- **data model**: A mapping of the contents of an information model into a form that is specific to a particular type of data store or repository. A "data model" the rendering of an information model according to a specific set of mechanisms for representing, organizing, storing and handling data.

- **DML**: Data Modeling Language

- **enumeration**: Data type with an enumerated set of values.

- **identifier**: Used to uniquely identify a model element (class, attribute, etc.) in the containing namespace.

- **information model**: An abstraction and representation of the entities in a managed environment, their properties, attributes and operations, and the way that they relate to each other. It is independent of any specific repository, software usage, protocol, or platform.

- **list**: Sequence of elements of the same type.

- **managed object**: An abstract representation of network resources that are managed. A managed object may represent a physical entity, a network service, or an abstraction of a resource that exists independently of its use.
o object: Instance of a class

o struct: Set of attributes without own identity

o relationship: Definition of an association between instances of two classes.
4. Requirements

4.1. Basic Requirements

4.1.1. Base Data Types

Description:
DML must support the base data types such as int8, int16, int32, int64, uint8, uint16, uint32, uint64, float32, float64, string, boolean, enumeration, octet, char, date-time, binary, empty.

Motivation:
Base data types are used frequently and simplify the module development.

4.1.2. Default Values of Defined Types and Attributes

Description:
DML must support default values to be used for different purposes such initial, runtime and failure fall-back.

Motivation:
Default values simplify configuration for start-up, failure and other scenarios.

4.1.3. Language Extensibility

Description:
The language must have characteristics, so that future modules can contain information of future syntax without breaking original DML translation tools and parsers.

Motivation:
Achieve language extensibility with downward compatibility.

4.1.4. Model Modularity

Description:
Network element model must be composable from separate sub-models. Circular dependencies between the model modules should not be allowed.
Motivation:

A sub-model can be used to encapsulate a certain management model. This can be an independent partition or a basic module that provides definitions for other modules. The whole model for the network element should be possible to derive from these separate model modules.

4.1.5. Protocol Independence

From: RFC 3216

Description:

DML must define data definitions in support of Netconf protocol. DML may define data definitions in support of protocols other than Netconf.

Motivation:

DML data definitions may be used with multiple protocols and multiple versions of those protocols.

4.1.6. Translation to Other Data Definition Languages

Description:

DML constructs must be translatable to other basic languages such as XSD or RelaxNG. DML may not contain constructs which are not possible to translate to other basic languages.

Motivation:

Use of existing tools based on such as XSD or RelaxNG e.g. for validation.

4.1.7. Module Conformance

Description:

DML must provide mechanisms to enable the verification of the module conformance defining minimum requirements implementers must meet for conformance.

Motivation:

Ability to convey conformance requirements.
4.2. Common Requirements

4.2.1. Model Element Identifiers

Type: Clarification / Refinement

Description:

Each referable model element must have an identifier that uniquely identifies it in the scope of the containing model element. This identifier must start with a letter (a - z, A - Z) and otherwise consist only of alphanumerical characters, including the underscore (a - z, A - Z, 0 - 9, _). It must have a limited maximum length.

Identifier equality should be non-case-sensitive.

Motivation:

Needed to create readable, hierarchical names that uniquely identify individual elements inside the model. The restricted set of legal characters should enable lossless mapping to languages as well as storage schema specifications.

Notes:

Limiting the set of allowed characters to alphanumeric characters is done to facilitate mapping to other domains like other data definition or data access languages (e.g. DDL, SQL). It should be taken into account that not all mapping target languages have case-sensitive identifiers, especially SQL.

4.2.2. Model Element Documentation Text

Type: Clarification / Refinement

Description:

It must be possible to provide a textual documentation (e.g. description) for each model element.

All kinds of characters, including language specific characters should be supported.

Motivation:

Model element documentation can be used inside user facing applications to provide or enrich online help.
Providing at least basic documentation for a model element helps to ensure that it is properly (re)used inside models as well as that correct instance data is provided for instances of that model element.

Notes:
Separating basic model documentation from model definition easily leads to incomplete documentation or even inconsistencies between model and documentation.

4.2.3. Model Element Presentation Name

Type: New

Description:
It should be possible to add a presentation name for the model element. The presentation name is for use in interfaces to human end users.

All kinds of characters, including language specific characters must be supported.

Motivation:
Technical identifiers become hard to read when they contain abbreviations or if they were constructed by concatenating several words, separated only by underscores or alternating lower-case with upper-case letters. In such cases a separate presentation names helps to create user friendly end user interfaces.

Notes:
Localizing presentation names could be done "on top" of the model if needed. But leaving presentation names completely out of the model often leads to hard to understand GUI’s as the technical identifiers are used for purposes they are not designed for (e.g. data entry field captions). The other alternatives are to add another description mechanism on top of the model or to reuse extension mechanism like annotations. However, such mechanisms lead to a higher DML usage and model maintenance cost.

4.3. Model Releases
4.3.1. Release Support

Type: New

Description:

It must be possible to specify the release (version) of the model. The release of the model supports to distinguish the different releases of the according network element.

The release implicitly applies to all model elements in the model.

The release identifier should at least allow alphanumeric characters plus ‘.’ (dot).

Motivation:

Network elements evolve over time; new features are added, existing ones are enhanced or slightly changed, others are dropped. Already in networks of moderate size, it is quite common that several releases of the same type of network element software are used in parallel.

Typing versions to individual model elements in one release agnostic (compound-) model wouldn’t be practical. For example, sometimes there is a need to state that a certain feature of a class is not supported in a particular release. But if that feature is still supported in other releases, it must still be visible. In general, reading of a compound-model becomes difficult as a release specific views must be explicitly constructed.

4.3.2. Multiple sub-module revisions in one model

Type: New

From: Network management layer modeling

Description:

It must be possible to combine multiple revisions of a model into one cohesive umbrella model without loss of information.

Motivation:

A network management system deals with multiple network elements of the same type, each of which supports a model on a certain, yet possibly different revision level. A model of the network management system – e.g. when defining its management capabilities for a higher-
level management system - would have to be the union of the individual models (and revisions) of the network elements.

Notes:

If not available, the network management system has two options:

- provide the latest revision of the network element’s model as part of the own model. Always transform the data complying to earlier revisions to the latest revision of a model.
- rename the network element models so that different revisions maps to different model element names. Always transform the data according to the element renaming rules.

When implemented, imported modules cannot be referenced by their name only, but need to be referenced by a naming convention including the revision. If different namespace prefixes are used in the umbrella model, then the model elements coming from different revisions can be distinguished.

Also some discriminator on data level would be required,

e.g. `<router yang:revision="2007-05-14"> <interface> <tcp/> ...`

4.4. Modeling Feature Requirements

4.4.1. Concrete Classes

Type: New

From: Basic network modeling

Description:

It must be possible to define classes as a cohesive package of metadata that at least define its data structure and its relationships to other model elements.

Motivation:

Fundamental modeling concept for encapsulating and grouping of structural features (i.e. attributes) for reuse.

Notes:

The behavioral features of classes as used in object oriented programming languages might not be of relevance for a data modeling...
language. But aspects such as structural features (attributes, relationships / associations) and relationship to other classes (inheritance) are. Therefore introducing the class concept in the context of a DML makes it highly useful.

4.4.2. Abstract Classes

Type: New

From: Basic network modeling

Description:

It must be possible to define classes which only cover the invariant structural features of derived, specialized classes. Abstract classes cannot be instantiated.

Motivation:

It must be possible to model abstract entities which itself cannot be instantiated but serve as a blueprint for derived, specialized data groups. This allows defining concepts that are useful for generic applications without the need to provide an implementation, respectively without the side-effect that an implementation is generated in addition.

Notes:

Abstract classes can also be used as the endpoints for relationships.

4.4.3. Single Class Inheritance

Type: New

From: Basic network modeling

Description:

It must be possible to derive a class from at most one superclass. An is-a relationship is established from the derived class to the superclass. The derived class inherits all features (attributes, relationships, constraints) of the superclass. In effect, instances of the derived class can act as substitutes for instances of the superclass in all contexts in which the superclass appears (Liskov substitution principle).

Motivation:
Class inheritance is needed in order to

- enable semantically rich modeling (is-a relationship)
- avoid redundancy (feature inheritance)
- ensure consistent modeling (superclass reuse)

Notes:

Inheritance is one of the key and most widely used concepts for creating semantically rich yet concise models. Leaving it out of the DML makes it hardly usable for use cases that require structuring a model in layers of different abstraction.

4.4.4. Attributes

Type: Clarification / refinement

From: Basic network modeling

Description:

It must be possible to define attributes as members of classes and attribute groups.

For each attribute following properties must be possible to define:

- The type. Primitive types as well as constructed types (sequences, structures) must be supported by the DML.

- The initialization mode of the attribute. At least the following modes must be supported:
  
  * Required: An attribute value must be provided during instance creation
  
  * Optional: An attribute value may be provided during instance creation
  
  * None: An attribute value must not be provided during instance creation ("none" is a correct, meaningful value in case of read-only attributes; there might be cases when an attribute can only be set afterwards.)

- Changeability after instance creation

For each attribute following properties should be possible to define:
The unit in which attribute values are measured.

A default value.

Motivation:

Fundamental data modeling concept to describe properties of managed objects which can be represented with data values.

Notes:

Specifying the unit as part of the attribute definitions allows describing the actual use of the attribute without having to create a new datatype that only wraps a primitive type. E.g. an attribute "acceptableCRCErrors" could just use the primitive datatype "int32" and set the value of 'unit' to "CRC-Errors / Sec". There is no need to define a datatype "CRCErrorsRate" that is possibly used only once in the whole model.

4.4.5. Attribute Groups

Type: Clarification / refinement

From: Basic network modeling

Description:

It must be possible to combine multiple attributes into a group that can be reused in the definition of classes. It must be possible that a class can reuse one or more attribute groups.

Motivation:

The purpose of attribute groups is mainly the grouping of related attributes which are typically always used together. That should help to avoid an inconsistent modeling of such attributes or having to copy and paste attribute definitions.

Notes:

Using classes don’t establish an is-a relationship to the used attribute groups. Attribute groups also help to alleviate the lack of multiple inheritance as it makes mix-in classes superfluous which are used just to provide commonly used attributes to derived classes.
4.4.6. Reference Relationships

Type: New

From: Basic network modeling

Description:

It must be possible to describe references between instances of two classes.

Adding reference relationships must be possible without changing the definition of the source- or target-end class

Motivation:

Often manageable network resources are associated in some way which is neither a containment relationship nor can be expressed by referring to attributes of related classes.

Notes:

Reference relationships can be realized based on XPath statements by sequencing object node names or by storing object key values of a referenced object.

Reference relationships are especially important to connect classes representing configurable resources in the network with rather abstract and typically network management related concepts.

A simple example is stating the relationship of a port number with the corresponding sub-network.

Another use case is describing at which geographical location a network resource was placed, represented by attributes like longitude, latitude and altitude or city, street, building. But many network elements do not allow storing such information as part of their configuration data. This issue can be solved by defining a location class and associate that class with a reference relationship to network resource classes.

4.4.7. Containment Relationships

Type: New

From: Basic network modeling

Description:
It must be possible to define where in a containment hierarchy instances of a particular class can be placed. In order to do that, a containment relationship relates the container object class (source end) with a class of contained objects (target end).

It must be possible to specify the maximum and minimum number of instances that can be contained in a container class instance.

Adding containment relationships must not require to alter the container class.

Motivation:

Containment hierarchies are a commonly used organization principle for manageable resources in network elements as well as for distinct but related elements in a network.

Notes:

It makes a difference if a containment is defined implicitly (instances of B are contained in A as the definition of B is placed inside the definition of A) or if containment relationships are defined outside of a data containers (classes). The second alternative has the advantage that new contained elements could be placed into an existing container without altering the definition of the container. A coexistence of both approaches is also an option.

4.4.8. Simple Attribute Constraints

Type: Clarification / Refinement

From:

Description:

It must be possible to constrain / refine the set of legal values for simple type attributes. The constraint specification language must at least support

- Length constraints for string / text-types
- Regular expression pattern constraints (e.g. 
  
  "(+|-)\d+\s*(mm|cm|m|km)"

- Numeric range constraints ([mix ... max])

It should be possible to combine several basic attribute constraints into a constraints expression by using the logical operators ‘and’,

'or' and 'not'.

Motivation:

Helps to avoid the input or use of invalid configuration data.

Simple Attribute Constraints Expressions allow specifying constraints that can be easily formulated by combining two simple constraints. This is useful for addressing cases where configuration parameters have a value range representing the normal mode of operation and one or more special values (e.g. "0", "-1", "", "-" "**") that represent special states.

Notes:

Supporting such constraints explicitly in a model has the advantage that evaluating them is possible wherever there is access to the model (metadata). E.g. checking such constraints is very useful - or rather expected - in GUI components that allow editing the configuration of network resources. Tools can check the constraints on the fly, preventing the provision of erroneous values.

4.5. Extension Mechanisms

4.5.1. Typed Annotations

Type: New

From: Basic network modeling

Description:

It must be possible to define typed annotations with multiple annotation elements (tagged values). In order to ensure correct use of annotations, it must be possible to define:

- To what types of model elements an annotation can be attached
- Whether an annotation element (for each model element of defined types) is required or optional (alternatively minimum and maximum occurrence)
- The type of an annotation element. Alternatively a regular expression that specifies the allowed values for the sting representation of the annotation element value.

Motivation:
Annotations allow enriching the model with additional information that might be necessary to take full advantage of the model respectively of data that instantiates the model. A typical use case is to refine the semantics of a model element. In that role annotations fulfill the same purpose as Stereotypes in UML. Annotations can also help to support maintenance and to achieve backward compatibility to other metadata.
5. IANA Considerations

Requirements on a Data Modeling Language are not IANA relevant.
6. Security Considerations

Requirements on a Data Modeling Language are not security relevant.
7. Acknowledgements

We would like to thank to Leo Hippelainen for his contributions and review.
8. References

8.1. Normative References


8.2. Informative References


Authors’ Addresses

Bernd Linowski
Nokia Siemens Networks
Email: bernd.linowski@nsn.com

Martin Storch
Nokia Siemens Networks
Email: martin.storch@nsn.com

Mehmet Ersue
Nokia Siemens Networks
Email: mehmet.ersue@nsn.com

Mikko Lahdensivu
Nokia Siemens Networks
Email: mikko.lahdensivu@nsn.com