Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2008

B. Linowski
M. Storch
M. Lahdensivu
M. Ersue (ed.)
Nokia Siemens Networks
March 11, 2008

Kalua - A Data Modeling Language for NETCONF
draft-ersue-netconf-kalua-dml-01

Status of this Memo

Abstract

This document specifies a Data Modeling Language (Kalua DML), which is designed to be used as a specification language for the payload of the IETF NETCONF protocol [RFC4741] as well as to specify other management related data models. The Kalua DML aims to fit the requirements specified in draft-linowski-netconf-dml-requirements [Linowski] and supports most of the requirements in RCDML [RCDML].

Comments can be sent to ngo@ietf.org.

Table of Contents

1. Introduction

This document specifies a Data Modeling Language (Kalua DML), which is designed to be used as a specification language for the payload of the IETF NETCONF protocol [RFC4741] as well as to specify other configuration management (CM) related data models. The Kalua DML aims to fit the requirements specified in draft-linowski-netconf-dml-requirements [Linowski] and supports most of the requirements in RCDML [RCDML].

XML-based data exchange and management has become increasingly important because of its flexibility, readability, and ease of use. XML supports hierarchical data structures and is supported by a large number of management applications. The NETCONF Working Group has completed the standardization of the XML-based configuration management protocol and its notification mechanism supporting asynchronous notifications.

However, a standardized way of configuration data modeling for Netconf is missing. There is a need to define a standard content layer based on a data-modeling framework for the development of standard and vendor defined data model modules.

The necessary Data Modeling Language should address the requirements of the Netconf protocols fully. However, the optimal case would be if the DML can be used also for management fragments other than the Configuration Management. We need to take into consideration the increasing need for extensibility and the opportunity of providing one data modeling language solution for different IETF problems also other than O&M issues e.g. application servers. The aim for a new DML solution at the IETF should be to support extensibility, broader applicability to different kind of management issues and harmonization of data models for manifold management applications.

Having this in mind the definition of a common O&M meta-model seems to be sensible where diverse management fragments, such as configuration management, can derive or inherit commonly used data modeling structures and do not define these themselves if already available. For the achievement of such flexibility, we propose an object-oriented approach where attribute groups and meta class definitions can be inherited to avoid data redundancy and to enable data model design flexibility.

In the following chapters the KALUA Data Modeling Language fitting to the requirements defined in draft-linowski-netconf-dml-requirements [Linowski] is specified. The specification defines the model elements of the KALUA language.

KALUA language defines how one can model basic concepts which apply to a specific management fragment, such as fragment identifiers, annotations, content trees, and common principles that apply to fragments of the metamodel (such as identifiers of model objects and references between the objects).

KALUA language is designed to model the operation and maintenance interface, that is, the crossing point where the network equipment and management system meet. Kalua defines how one can model configuration management concepts and can use and combine the features of Kalua to create expressive and concise data models.

Kalua language provides built-in abstract and concrete object classes, attributes, attribute groups and data types that the data models may use by inheritance. This approach harmonizes concepts, which are widely used in operations and maintenance data models.

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 [RFC2119].

3. Documentation conventions

3.1. Terminology

- o abstract class: abstract class is a class, which cannot be instantiated. That is, an object instance cannot be created with an abstract class as its class. However, abstract classes as a super-class of a class does not prevent the non-abstract class to be instantiated.

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

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

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

- o attribute container: A model element that contains attributes. Abstraction of attribute groups, structures and classes.

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

- o built-in type: A data type defined in the Kalua, such as 'unsignedInt' or string.

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

- o 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" is the rendering of an information model according to a specific set of mechanisms for representing, organizing, storing and handling data.

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

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

- o 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.

- management fragment: category of management tasks. For example configuration management, performance management, and fault management are management fragments.

- model element: A building block of the Kalua language.

- module: A set of related definitions

- object: Instance of a class

- relationship: Definition of an association between instances of two classes.

- struct: Set of attributes without own identity

- super class or superclass: A class from which the derived class is directly derived.

3.2. Model element descriptions

Each model element is described in its own section, with the model element name as a title. Attributes of a model element are described in an attributes section. Model elements, which are contained in a model element, are listed in a sub-elements section. XML elements, which are not model elements, are described in a leaf sub-elements section.

In the sub-elements section, a minimum and maximum number of occurrences per containing model element are defined for each sub-element. If maximum number of occurrences is 'unbounded', there is no upper limit on how many times the sub-element may be repeated. The status of the elements has been stated as M(andatory or O(ptional).

The model element sections give a definition example of that model element type in Kalua language, the relevant part of the Kalua XML schema, and if applicable, an example of the Netconf payload model by using the model element definition.

3.3. Constraints

In case of complex restrictions on sets of acceptable values for features of a Kalua element, the constraints are listed in a separate constraints section.

4. Language Introduction

4.1. Language Design Fundamentals

Kalua is an XML-based language for configuration data modeling and network respectively system information modeling.

Using XML as a technological foundation for Kalua was chosen since:

- o NETCONF is an XML passed protocol in which protocol elements as well as payload is encoded in XML. So using XML also as a basis for the language that is supposed to define the structure of NETCONF avoids technological fragmentation.

- o XML is well known in the IT industry. Using it as a basis for a language decreases the barrier for adoption on the syntax level. People can mainly focus on understating the language concepts and their semantics instead of learning an entirely new syntax that requires new tools for validation and processing.

- o XML and the specification language for XML documents (XML schema) have almost ubiquitous support in all kinds of IT systems. So creating, validating and processing of language documents should only require low to modest effort.

Kalua combines concepts of data modeling (such as structures, sequences, attribute groups etc.) with concepts from object-oriented domain modeling (classes, relationships and class inheritance). This was done because of several reasons:

- o Network management in particular but also system management in general is often done in a hierarchical manner. That means some kind of manager, a network management system, a mediator but also network elements with management functions have to deal with plain configuration data as well as with some form of data represented by more or less generic models. Being able to express both types of data in Kalua helps to keep the integration and mediation effort low when integrating different systems, operating at different levels in the abstraction hierarchy. Even if other modeling languages like UML (at the highest levels) or SMI (at the lower levels) are used in different layers, being able to represent core concepts of such languages in Kalua helps to avoid "semantic gaps" that require cumbersome and costly "bridges" (in form of mediators, model transformers, etc.)

- o Enhanced expressiveness: It is very useful to be able to specify plain data structures and links as well as concepts with a more refined semantics like classes and relationships (associations).

Being able to choose the most appropriate modeling construct ensures correctness, maintainability, and reusability.

- o Flexibility: Managed systems with individual elements are in a continuous change, i.e. new types of network elements are introduced, new versions of such elements are created, and additional ways of connecting (relating) elements need to be represented. The challenge here is to integrate new parts into the picture without having to alter the models for existing parts. Object oriented modeling provides mechanisms to address this challenge. Abstract classes allow defining and relating generic concepts. For example, this facilitates implementing generic management functionality that can be applied to instances of all concrete classes which specialize particular abstract classes. Inheritance allows to add new enhanced or otherwise refined without touching the base class. Relationship refinement allows detailing how newly created resources (represented by classes) participate in already defined relationships.

- o TM Forum NGOSS SID (Shared Information/Data) model, which is the basis of OSS/J API data models, and the DMTF CIM (Common Information Model) are examples of data models in the network management domain that make use of object- oriented concepts.

4.2. Language Overview

This section introduces the core concepts of Kalua, puts them in perspective to common problems in configuration and network modeling, and illustrates their use and interrelations with examples.

4.2.1. Network resource configuration modeling

As a language that is supposed to specify the structure of the payload of the Netconf protocol, Kalua has various language features for specifying the data structures representing configuration elements and state description elements of systems.

4.2.1.1. Property modeling

As a DML for configuration management, it must be possible to describe the properties of manageable resources that are subject of configuration. In addition, it must be possible to specify properties that represent the actual state of such a manageable resource. Such properties are modeled in Kalua in form of attributes.

```
<attribute name="linkSpeed">
      <type>kalua:long</type>
         ...
</attribute>
```

An attribute is simply a property of some manageable entity, which has a name and a type, among some other features that control how the attribute can be used. Attributes can have primitive types (int, boolean, string, etc.), refined simple types, as well as complex types (structures, sequences, etc.).

4.2.1.2. Primitive Types and Refinement of Primitive Types

Manageable resources typically have many attributes of primitive type, for example numeric parameters, string type parameters, boolean flags or configuration items with an enumerable set of values.

Kalua therefore supports all non-XML specific and not date-related primitives and build-in types defined in [XML schema 1.1 Part 2: Datatypes]. In effect that means that all signed integer types (short, int, long etc.), all unsigned integer types ("unsignedInt", "unsignedLong" etc.), "float" and "double" as well as other basic primitives like "string" and "boolean" are supported. Predefined types are used by referring to them in type elements.

```
<type>kalua:int</type>
```

Also "dateTime" and "duration" are supported in order to express points in time or periods of time. Other XML schema data types that deal with time related values are not part of Kalua in order to avoid that systems interpreting Kalua defined contents have to cope with many types that only provide little additional value (XML schema 1.1 Part 2: Datatypes defines 11 different time related types). In addition, the XML centric types like "NCName" or "QName" are left out from Kalua as their use would require detailed XML knowledge and their value outside XML centric applications is questionable.

Since many network resource parameters accept only a subset of the range of values that can be hold by a primitive type instance, it is quite essential that such restrictions can be exactly yet easily expressed.

For example, for an attribute that specifies an angle in degrees, it should be possible to limit the range of acceptable values to start from 0.0 inclusive and end at 360.0 exclusive. In Kalua, restricting the set of values that can be applied to an attribute is done by refining an existing primitive type with constraints.

```
<attribute name="angle">
      <simple-type>
          <restriction>
            <type>kalua:double</type>
            <minInclusive" value="0.0">
            <maxExclusive" value="360.0">
          </restriction>
      </simple-type>
      ...
   </attribute>
```

4.2.1.3. Complex Datatype Modeling

Apart from creating new types by refining primitive types, Kalua also supports the construction of complex types, namely structures, sequences, and enumerations.

Many network element parameters accept only a few distinct value representing different configuration options. Also, the state of resources is often described with a few distinct literals. The operational state as defined in ITU-T X.721 is a typical example. In order to be able to define attributes that accept only a value from a fixed set of alternatives, Kalua supports enumerations.

```
<attribute name="operationalState" ... >
    <simple-type>
      <enum>
         <enum-literal name="enabled"/>
         <enum-literal name="disabled"/>
      </enum>
    </simple-type>
   ...
</attribute>
```

Many network elements have some kind of data that is organized in lists, arrays or tables, simply because some basic set of data has to present in multiple instances in order to describe configuration or state data properly and completely.

Kalua support this kind of data structures in form of sequences. Depending on the properties of the sequence, it can represent arrays (sequences with a fixed length), list (sequences with a variable length) or even bags (sequences in which the actual position of an element has no meaning).

```
<sequence minLength="32" maxLength="32" elementName="timeslot">
   <type>kalua:boolean</type>
   ...
```

```
   </sequence>
```

In order to represent the configuration or state data of network resources accurately, it is often needed to group various attributes with probably different types into an own organization unit. Kalua allows defining structures in order to address this need.

```
   <structure>
      <attribute name="host">
         <type>kalua:string</type>
      </attribute>
      <attribute name="port">
         <type>kalua:unsignedShort</type>
      </attribute>
      ...
   </structure>
```

Apart from consolidating attributes into a bigger unit, structures are also useful to structure the namespace for properties of a manageable resource.

The full potential of sequences and structures is only unleashed when they are combined. Many network elements and other kinds of configurable systems have some kind of data tables. In Kalua, a table can be modeled as a sequence of structures. The member attributes of the structure define the columns, the properties of the sequence define the extend of the table.

```
<sequence>
    <structure>
         <attribute name="host">
            <type>kalua:string</type>
          ...
         </attribute>
         <attribute name="port">
            <type>kalua:unsignedShort</type>
          ...
         </attribute>
         ...
      </structure>
      ...
</sequence>
```

It is allowed to nest complex type definitions in an arbitrary fashion. Therefor it is possible to define structures that contain sequence-type attributes, sequences of sequences, etc.

4.2.1.4. Named Data Types

In the examples above, the structures, sequences and enumerations were not given a name. In case a complex datatype is defined inside an attribute, a name is not needed as the name of the attribute is used to address the element of the datatype. In case a complex datatype is defined inside a sequence, an index-number is used.

Often complex types can be reused in various different places of the configuration of a configurable system (network element). For example, many IP-based network elements must deal with several IP-addresses as part of their configuration. It is useful to create a structure, which combines the attributes, e.g. describing an IP-address, and to give it a name enabling the usage wherever needed. Kalua introduces the typedef element for this purpose.

A type definition (typedef) simply gives a datatype a name. The name specified as part of the typedef is applied to the datatype defined inside the type definition element.

```
<typedef name="ipAddress">
    <structure>
         <attribute name="host">
            <type>kalua:string</type>
          ...
          </attribute>
         <attribute name="port" ...>
            <type>kalua:unsignedShort</type>
          ...
         </attribute>
         ...
      </structure>
      ...
   </typedef>
```

Such a user-defined data type can now be in each context where a type is expected. For example, such a type can be used inside attribute definitions and sequence definitions.

```
   <attribute name="eventReceiver">
      <type>ipAddress</type>
       ...
   </attribute>
```

Type definitions can only appear in the scope of a module (they are top-level elements).

4.2.2. Network Modeling and Network Management Support

Manageable network elements do not exist in isolation. Almost any kind of manageable network resource is in some form related to other network resources. This web of network elements connected via all kinds of relationships - the network topology - is one of the cornerstones of effective network management. It has even significant impact on the management of individual network elements as many of their configuration elements realize or depend on the topology and typically a large portion of their state data can only be interpreted with respect to its place in the network topology.

- o A network resource is contained in another resource. Containment means that a manageable resource is physically contained in another one, for example a card that is plugged into a rack. However, containment could also mean that a resource is only conceptually enclosed in another resource. This is often synonymous with being dependent in terms of manageability on the containing resource.

- o A network element is connected to another network element. For example, data transmission channels like PCM links can be seen as connections.

- o Network resources are in some form related to conceptual entities like maintenance regions or locations.

- o Network resources use functionality of another resource.

Being able to model network resources as well as conceptual resources and the relationships between them is quite essential for many management tasks. Kalua therefore supports two concepts form object oriented domain modeling, namely classes and relationships.

4.2.2.1. Classes

The main purpose of classes is to represent configurable entities that share the following characteristics:

- o they have an own identity,

- o they have an own, independent life cycle, and

- o they are often treated as being a more abstract entity.

- o Instances are often related to instances of other classes in some form.

In Kalua, classes are containers of attributes. That makes them similar to structures. But in contrast to structures:

o Classes can have a superclass. A class inherits all attributes (and involvement in relationships) from its superclass. The same applies for the superclass of the superclass and further ancestor classes.

o Instances of classes can be used wherever instances of the superclass can be used (Liskov substitution principle). So an is-a relationship is established between the derived class and its superclass.

o Classes can be abstract. That is useful to define concepts that serve as a blueprint for concrete derived classes. In addition, relationships can be defined that involve an abstract with a concrete class or even with another abstract class.

o Classes must have a key in case they are asscoiated to other classes by reference relationships. As instances of such classes have to have an own identity, it must be possible to address class instances uniquely by a key value. That is also needed in order to realize relationships between classes respectively between instances of related classes.

Classes are a vehicle to model "first-class network resources type" like types of network resources with an own O&M interface as well as independent conceptual entities like regions, sites, policies, plans etc.

4.2.2.2. Relationships

Kalua also supports the concept of relationships. A relationship associates instances of two classes, which can be two distinct classes or the same class. The two endpoints of the relationship are called source and target. The naming of the endpoints should lead to a consistent usage of relationship definitions. This does not mean that a relationship can only be navigated from source to target.

An important aspect of relationships in Kalua are that they are defined outside the classes they connect. That has several advantages:

o It is not necessary to anticipate and define all kinds of relationship that might be needed in the future. Instead, relationship can be added "on top" of the classes they connect later when really needed. That also prevents having to deal with relationships that were once introduced because it was assumed

they would be needed but actually are not used.

o Relationships and classes can be separated into different modules. Therefore, it is possible to define classes and their inner structure (by using and defining all kinds of data types) in one module and put relationships and supplementary definitions for network modeling in another module. While a network element agent only deals with the first module, a higher-level management system can use the second module, which includes the first.

o They are very much decoupled. In case the definition of a relationship would be owned by one of its endpoints (or the definition would be even shared between both endpoints), introducing a new relationship would require that the modules that contain the classes to be related have to be changed. That might not be a big technical problem, but is often more an organizational issue.

```
<class name="Manager">
    ...
</class>
<class name="ManagedObject">
    ...
</class>
<relationship name="manages">
    <source>
       <class>Manager</class>
       ...
    </source>
   <target>
         <class>ManagedObject</class>
                  ...
   </target>
    ....
</relationship>
```

Kalua also supports relationship refinement. I.e. it is possible to define relatively high- level (abstract) relationships, which are refined by relationships that connect more concrete classes.

```
<relationship name="controllerFunctionOf">
    <base-relationship name="core:manages">
    ....
</relationship>
```

A typical use case for this feature is the proper modeling of

containment relationships, especially such that define the management hierarchy of network resources. The parent-child relationship is then defined at the root of the class hierarchy for managed objects (managed object contains an arbitrary number of other managed objects).

In order to properly define the relationship semantics as well as to enhance flexibility and expressiveness, three kinds of relationships are supported in Kalua:

- o Reference relationships. They simply associate two classes.

- o Containment relationships. This kind of relationship defines a strict existence dependency between the contained object (target end) and the containing end (source end). It means that if the container object is deleted all contained objects are deleted as well.

- o Calculated relationships. Here, it is defined which instances of the source and target end class (and their subclasses) are actually related by a relationship expression. In effect, the contents of the relationship are actually defined by the state of the instances at the source and target end. Such relationships have to be evaluated "online" whenever they are queried.

Two important aspects of relationships as defined in Kalua deserve some explicit discussion.

When specifying the containment of class instances in Kalua, this has to be done by defining containment relationships between them. While this requires some effort for explicit specification of the containment relationships, it has several advantages:

- o Especially, it is possible to specify more than one containment relationship that has the same contained class at the target end. I.e. a class (respectively their instances) can be contained in multiple different classes.

- o It is possible to specify in a common way that a previously defined class can be contained in a new class, so that the definition of containment is not constrained by any other organizational aspect.

- o The multiplicity of instances of the contained class (target end) can be exactly specified.

Kalua supports the specification of prescriptive reference relationships and containment relationships. In addition, it allows

the specification of descriptive calculated relationships. It is important to support both types of relationships as they address different use cases:

- In many cases references to resources in the same or other network elements are realized with attributes that contain some kind of implicit key value or a part of a composite key.

  For example, in order to describe an "is-connected-to" relationship associating two network elements that are physically connected via PCM links, the reference to the other end of the association could be represented by a numerical id of the PCM link terminated in the network element plus the index of the timeslot in that PCM link. The PCM id as well as the timeslot index is then represented as two numerical attributes.

  This kind of association is best represented with a calculated relationship with an expression that compares tuples of instances of the class at the target end and instances of the class at the source end by checking if they have the same PCM link id and timeslot index stored in two particular attributes.

  Realizing such a descriptive relationship with a reference relationship typically leads to the problem of keeping track with the configuration changes.

- Now if such network elements should be associated to a maintenance region, a conceptual entity that groups resources based on their geographical or even logical location in a network, using a descriptive approach does not work, respectively is rather inappropriate.

  Many network elements do not have configuration elements that can be used to store a key of a maintenance region, even putting this information directly into a resource in the network is not a good idea in the first place. This kind of association is best realized in a prescriptive way by defining a reference relationship. It is then left to the system that in managing relationships specified in Kalua where and how the actual relationship information is stored. The key specification feature, which is part of Kalua, just describes which attributes of a resource could be used for storing relationships.

4.2.2.3. Information Modeling with Classes and Relationships

The example below illustrates some of the essential aspects of classes and relationships. First, network elements and locations are modeled as abstract classes because network elements and sites

obviously have an independent life cycle. A network element comes into existence terms of management at the latest when it is deployed into the network and switched on. A location comes into existence when somebody creates it in some kind of management system.

Network element as well as location is abstract concepts. By itself they do not contain enough information to be usable for most (but not necessarily all) concrete management operations. However, defining them allows the establishment of a relationship between them, which provides already some value. For example, it is possible to tell if two network elements are placed at the same location and get the description of the location of a network element.

As instances of "ManagedObject" are related to the instance of "Location", they can be also related to instances of "Site" as a specific type of location. It is even possible to define another class that derives from "Location", e.g. one that uses geographical coordinates to describe a location, and uses instances of that class to represent the location for any type of network elements.

```
<class name="Location">
    <abstract>
   <attribute name="locationId">
      <type>kalua:long"</type>
        ...
   </attribute>
   <attribute name="description">
      <type>kalua:string"</type>
        ...
   </attribute>
    <key>
       <member>locationId</member>
    </key>
</class>

<class name="NetworkElement">
    <super-class>ManagedObject</super-class>
   <attribute name="vendor">
      <type>kalua:string</type>
        ...
   </attribute>
</class>

<relationship name="locatedAt">
    ...   <source>
       <class>ManagedObject</class>
        ...
    </source>
```

```
      <target>
          <class>Location</class>
          ...
       </target>
       ....
   </relationship>

   <class name="Site">
       <super-class>Location</super-class>
      <attribute name="contactDetails">
         <type>kalua:string</type>
          ...
      </attribute>
       <attribute name="street">
         <type>kalua:string</type>
          ...
      </attribute>
       <attribute name="city">
         <type>kalua:string</type>
          ...
      </attribute>
      <attribute name="postalCode">
         <type>kalua:string</type>
          ...
      </attribute>
   </class>
```

4.2.3. Notifications

Same data model applies to change notifications, get (get-config, get) and edit-config operations of Netconf.

In the context of Netconf protocol, operation attribute of a data element within the notification indicates whether the data has been merged, created, replaced, or deleted. The semantics of updating the data with notification from server to the client are the same as changing data with an edit-config from client to server. Default behavior is to merge.

Several changes can be combined in the same notification, and changed changes done with one edit-config may be sent as several notifications.

The following example demonstrates how the same Kalua model is used in get-config and edit-config requests and in a change notification.

The following is the definition of the module used in the example:

```
<module name="example profile"
xmlns="urn:ietf:params:xml:ns:kalua:1"
   <ns-uri>http://www.example.com/profile</ns-prefix>
   <ns-prefix>pr/<ns-prefix>
   <release>1.0</release>

   <class name="profile">
      <attribute name="name">
         <type>kalua:string</type>
      </attribute>
      <attribute name="name">
         <type>string</type>
      </attribute>
      <key>
         <member>name</member>
      </key>
   </class>

</ module>
```

First, the client requested get-config shows the initial state of the configuration:

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running/>
      </source>
   </get-config>
</rpc>

<rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data>
      <top xmlns="http://example.com/schema/1.2/config">
         <profile>
            <name>profile-1</name>
            <type>auto</type>
         </profile>
         <profile>
            <name>profile-2</name>
            <type>manual</type>
         </profile>
      </top>
   </data>
</rpc-reply>
```

Then, another Netconf client requests edit-config from the same server to change the running configuration:

```
<rpc message-id="102"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
      <target>
         <running/>
      </target>
      <config>
         <top xmlns="http://example.com/schema/1.2/config">
            <profile>
               <name>profile-1</name>
               <type>manual</type>
            </profile>
         </top>
      </config>
   </edit-config>
</rpc>

<rpc-reply message-id="102"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <ok/>
</rpc-reply>
```

Then, a change event notification is sent to all clients which have subscribed the notifications for this part of the data:

```
<notification
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data>
      <top xmlns="http://example.com/schema/1.2/config">
         <profile xc:operation="create">
            <name>profile-1</name>
            <type>auto</type>
         </profile>
       </top>
   </data>
</notification>
```

The client can also see the change in the get-config result. The data parts of the data which are not included in the change notification have not been changed.

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running/>
      </source>
   </get-config>
</rpc>

<rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data>
      <top xmlns="http://example.com/schema/1.2/config">
         <profile>
            <name>profile-1</name>
            <type>auto</type>
         </profile>
         <profile>
            <name>profile-2</name>
            <type>manual</type>
         </profile>
      </top>
   </data>
</rpc-reply>
```

4.2.4. Simplicity and Ease of Use

While being complete in terms of expressive power, the DML should be as simple as possible. The main reason for that is ease of use and understandability.

Simplicity in the context of a modeling language means:

- As few language elements as possible. The more elements a language has, the more time it takes to learn them.

- No complicated rules that restrict how and in which way language elements can be used. Having the freedom to use language elements wherever they make sense allows concentrating on the modeling problem at hand.

Kalua is an XML based language, because:

- The XML syntax is well known and the structure of basic syntax elements like elements, attributes is quite simple. As long as the overall structure of the DML language documents remains at a

decent level and the verbosity is minimized, the usage of XML syntax is acceptable.

- There are also a huge amount of tools available that allow editing, validating, and processing XML. That compensates the existing annoyances of XML as a language.

- Finally, usability needs to be seen from the point of view of an engineer that is supposed to implement software that is reading, writing, or transforming DML documents. In case of an XML-based language, one can use existing, well-known software components with standardized or at least widely accepted interfaces.

4.2.5. Straightforward and Lossless Mapping

4.2.5.1. Mapping to XML Schema

As Kalua is an XML-based language, which describes structural aspects of network resources, it is possible to translate a Kalua model (document) into an XML schema. That allows to use off the shelf XML parsers to read Kalua model files, check their well- formedness and validate their contents.

4.2.5.2. Mapping to UML2

As Kalua supports with classes, relationship (associations), and inheritance some of the core object oriented design concepts, translating Kalua models to UML2 models is done smoothly. Here the integration of object oriented concepts and their distinction from data modeling concepts pays back as it does not need to "guess" (by using some formalized heuristics) by what UML2 metamodel element a Kalua element is correctly represented.

4.2.5.3. Mapping from UML2

Also for mapping in the other direction, from UML2 to Kalua, the integration of object- oriented concepts makes things easier. The mapping from models specified with UML (version 2 as well as previous 1.x releases) is important as many standard models in the telecommunication domain like TMF SID, CIM and 3GPP are specified in UML.

4.2.6. Kalua as Metamodel

Models can also be seen as instances of a metamodel. This point of view is especially important because the metamodel prescribes to a large extend how models are represented in object oriented terms as the predominant paradigm in programming languages today.

The language features of Kalua are designed so that they can be easily mapped to metamodel classes and mix-in-classes (that can be also represented as interfaces). That should foster a structurally common representation of Kalua model elements in programming environments.

4.2.7. Release Management

Managed resources change over time, new features are introduced, existing features are removed. Thus their models must also change to accurately reflect these changes. At the same time, existing data, created with the old model, must be usable together with the new data. Applications developed against old model should be usable after upgrade. For example, views created in a management system should not become unusable whenever there is a change in the managed resources. They should be usable as long as the change is not affecting the parts of the model that they rely on. For example, adding an attribute to a class does not affect an application reading the data, and adding an optional attribute to a class does not affect the application writing the data.

Kalua allows any number of releases of a module. Each module release may add or remove module elements compared to other releases. Model elements with the same name appearing in different releases of the same module are considered to represent the same type of managable resources.

Multiple releases frequently occur when building systems that are both in a manager and agent role. E.g. a management system manages several agents via the NETCONF protocol, and exposes its own configuration data to an upper level management system via the NETCONF protocol, including the configuration data from the managed agents. Each agent defines its configuraton data as a Kalua module. The management system imports each of these modules into one composite Kalua module. This composite module defines the configuration data the management system exposes towards any upper level management system via the NETCONF protocol.

In the upper scenario, the need for multiple releases occurs when there are Kalua modules of different releases but same type among the modules of the agents.

Hierarchies of NETCONF interfaces may appear in the following system architectures:

o A controller device providing a NETCONF interface manages several subdevices via NETCONF.

- An element management system providing a NETCONF interface manages several devices via NETCONF.

- A regional management system providing a NETCONF interface manages several element management systems via NETCONF.

- A global management system manages several regional management systems via NETCONF.

```
<module name="com.example.controller">
   <presentation>Example Controller</presentation>
   <ns-uri>http://www.example.controller.com/</ns-uri>
   <ns-prefix>controller</ns-prefix>
   <release>2.1</release>
   <organization>Example</organization>
</module>
<import>
      <ns-uri>http://www.subdevice.com/</ns-uri>
      <ns-prefix>subdevice1.0</ns-prefix>
      <release>1.0</release>
</import>
<import>
      <ns-uri>http://www.subdevice.com/</ns-uri>
      <ns-prefix>subdevice2.0</ns-prefix>
      <release>2.0</release>
</import>
```

4.3. Use of XML and XML Schema

Kalua is an XML-based language. The Kalua syntax and basic part of its semantics are specified in an XML schema - the Kalua schema (see Appendix A.).

The syntax of Kalua was designed along the following guidelines:

- Primary language concepts that can contain other language concepts are realized as XML elements.

- Properties of primary language elements that potentially have long text values are represented as leaf XML elements with simple type contents.

- Only the name of definitions and properties that always have short values are realized as XML attributes.

- o Mixed content is prohibited.

5. Kalua Elements

KALUA specification introduces a set of language elements. Many of the concepts defined in KALUA contain a set of common attributes or features defined in the sub- chapters below. Each concept definition specifies which of these attributes, if any, are applicable to the concept and whether the attribute is mandatory or not in this context.

5.1. Common Kalua Elements

The elements in this section are commonly used by Kalua language elements that define model elements. The value for each of the elements is provided as element body text.

5.1.1. name

Type: string [a-zA-Z][_A-Za-z0-9]{0,29}

Description:

"name" is a unique and permanent identifier of the KALUA element within a single module. "name" cannot be changed during the lifetime of the element (that is, across releases of a module); if the identifier changes, the element is considered to be new and the old element is considered to be deleted. Thus, the old data corresponding to this object is no longer accessible through the adaptation.

The case of characters does not play a role in the uniqueness criteria. This means 'BTS' and 'bts' are overlapping identifiers. However, references to the "name" use the same case as in the definition of the element (see Section 5.1.4.).

"name" is used to refer to the KALUA element in KALUA files, database schemata, data files, other metadata/configuration files, APIs, and everywhere where references to elements need to be interpreted by applications. However, "presentation" (described below) should be used in user interfaces to identify the elements.

In context of adaptation development for each NE type, the uniqueness criteria of identifiers needs to be defined so that uniqueness constraints described in this specification and KALUA fragment specifications are met.

Elements of different type may have the same "name".

Specific model element types may have additional constraints for

uniqueness of the "name" for a specific concept, defined separately for each model element type.

5.1.2. presentation

Type: string, maximum length 100

Description:

A short name visible to the end-user in application user interfaces. "presentation" is not used to refer to a KALUA element in data or metadata. Only end-user documentation should refer to the elements using "presentation". "presentation" can be changed without breaking the compatibility of existing data or other parts of metadata.

"presentation" included in the model files is just a default. Default presentation could be overridden by language specific "presentation". If "presentation" is omitted it defaults to the value of name.

"presentation" does not need to be unique within an adaptation or across adaptations. However, as "presentation" normally serves as an identifier of the element for the user, you should avoid overlapping values where two elements may be used in the same context (for example, in AttributeDef sub-elements of one ClassDef).

5.1.3. description

Type: string, maximum length 2000

Description:

"description" is a longer text, which helps end users to understand the purpose and other details of the element. It can span multiple lines, and can contain any characters.

Applications displaying the "description" are also capable of deriving and displaying such information directly from metadata.

5.1.4. References to KALUA elements

KALUA language specifies references from model elements to other model elements. For each reference, a target model element type, for example attribute-group, is defined. While the semantics of a reference depend on model element type, and are definied for each model element type separately, KALUA uses a common syntax for the references to the model elements. The reference consists of a namespace prefix, a colon, and a model element name, that is, ns-

prefix:name.

Each reference has one target model element. The module where the model element is defined is considered target module of the reference.

If the target module is the module containing the reference, the ns-prefix part of the reference must equal to the ns-prefix element of the module.

If the target module is another module, there must exist an import element, within the module containing the reference or some modules imported by the module containing the reference, where value of the ns-prefix element equals to the ns-prefix part of the reference and ns-uri element equals to the ns-uri element of the target module.

The name part of a reference must be equal to the value of the name attribute of the target model element.

5.1.5. Types

KALUA supports a subset of the primitive types respectively build-in types defined in 'XML Schema 1.1 Part 2: Datatypes'. Most of the numeric types are supported, the types for specifying a point in time and a duration, string and boolean as well as a selection of name types.

The table below lists all primitive types supported by Kalua. Their value range, support for special values (like NaN, INF, -0), and lexical mapping is supported as defined in 'XML Schema Part 2: Datatypes' [XSD-TYPES].

In addition:

- a value for type dateTime can also be specified in seconds from Epoch 00:00:00 on January 1, 1970, encoded as non-negative, decimal integer.
- A value for type duration can also be specified in seconds, encoded as non-negative, decimal integer.

In effect, values matching the regular expression '\+?[1-9][0-9]*' has to be interpreted as seconds from Epoch (dateTime) respectively duration in seconds (duration).

Identifier	Description
string	String of characters. In case no length or max-length restriction is specified, it is not guaranteed that strings longer than 250 characters can be stored in attributes that use a the string type or a type that is based on string without any length restriction..
boolean	Boolean value
byte	Signed 8 bit integer.
short	Signed 16 bit integer
int	Signed 32 bit integer
long	Signed 64 bit integer
unsignedByte	Unsigned byte
float	IEEE 754 compliant, 32 bit floating point value
double	IEEE 754 compliant, 64 bit floating point value
unsignedShort	Unsigned 16 bit integer
unsignedInt	Unsigned 32 bit integer
unsignedLong	Unsigned 64 bit integer
dateTime	Date and time value
duration	A duration of time
decimal	Fixed point decimal with p decimal digits before the decimal point and s digits behind the decimal point. p is the precision and s the scale. The default precision is 10 and the default scale is 6. Specify precision and scale with a precision constraint.
integer	Signed integer up to p decimal digits, where p is the precision. The default precision is 10..

5.2. module

"module" is the root element of the Kalua definitions. Each Kalua document must contain exactly one "module" element. All other definitions of the model are contained in a module element.

"module" is a logical grouping of definitions. However, the split of definitions to separate modules also implies the following semantics:

- o "module" implies the version of the definitions
- o "module" implies the namespace of the definitions
- o applicability of annotation types are limited to those "modules" which define or import them
- o references to model elements can occur only to definitions in the "module" itself, or any directly or indirectly imported module

5.2.1. Element Attributes

Attribute	Type [Default]	Description	S
name	xsd:string	Unique identifier of the module within systems using this module. To select globally unique identifiers, identifiers should begin with an inverse domain name of the entity defining the module.	M

5.2.2. Leaf Sub-Elements

Sub-Element	Type [Default]	Description	S
presentation	xsd:string	See Section 5.1.2	O
description	xsd:string	See Section 5.1.3	O
ns-uri	xsd:string	Defines the target namespace of the all definitions in this module. In Kalua, this uniquely identifies the module.	M
ns-prefix	xsd:string	Defines the prefix used to attach the namespace to the model element names.	M
release	xsd:string maximum length 20. [a-zA-Z0-9]+ (\.[a-zA-Z0-9]+)*	Indicates the release version of this specific definition of the module.	M
organization	xsd:string maximum length 100	Name of the organization responsible for the module.	M

5.2.3. Sub-Elements

Sub-Element	MinOccurs	MaxOccurs	Description
annotation	0	unbounded	See Section 5.19
import	0	unbounded	See Section 5.3
typedef	0	unbounded	See Section 5.13
attribute-group	0	unbounded	See Section 5.5
class	0	unbounded	See Section 5.17
relationship	0	unbounded	See Section 5.18
annotation-type	0	unbounded	See Section 5.21

5.2.4. XSD

```
<xsd:element name="module">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:group ref="kalua:NamedElementProperties"/>
               <xsd:group ref="kalua:ModuleIdentityProperties"/>
               <xsd:element name="organization">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string">
                                 <xsd:maxLength value="100"/>
                           </xsd:restriction>
                     </xsd:simpleType>
               </xsd:element>
               <xsd:element
                     name="import"
                     type="kalua:importType" minOccurs="0"
                     maxOccurs="unbounded"/>
               <xsd:sequence>
                     <xsd:choice minOccurs="0"
                      maxOccurs="unbounded">
                           <xsd:element
                                 name="typedef"
                                 type="kalua:typedefType"
                                 minOccurs="0"
                                 maxOccurs="unbounded"/>
                           <xsd:element
                                 name="attribute-group"
```

```
                                    type="kalua:AttributeGroup"
                                    minOccurs="0"
                                    maxOccurs="unbounded"/>
                              <xsd:element name="class"
                                    type="kalua:Class"
                                    minOccurs="0"
                                    maxOccurs="unbounded"/>
                              <xsd:element
                                    name="relationship"
                                    type="kalua:Relationship"
                                    minOccurs="0"
                                    maxOccurs="unbounded"/>
                              <xsd:element
                                    name="annotation-type"
                                    type="kalua:annotation-typeType"
                                    minOccurs="0"
                                    maxOccurs="unbounded"/>
                        </xsd:choice>
                  </xsd:sequence>
            </xsd:sequence>
            <xsd:attributeGroup
             ref="kalua:NamedElementAttributes"/>
      </xsd:complexType>
   </xsd:element>
```

5.2.5. Element Examples

```
   <module name="com.example.ethernetInterface">
      <presentation>Example Ethernet Interface</presentation>
      <ns-uri>http://www.example.com/</ns-uri>
      <ns-prefix>example</ns-prefix>
      <release>2.1</release>
      <organization>Example</organization>
   </module>
```

5.3. import

"import" element makes all definitions from another module available in the module containing the "import" element. All definitions imported to that other module from any other modules are imported as well.

5.3.1. Leaf Sub-Elements

Sub-Element	Type [Default]	Description	S
ns-uri	xsd:string	Selects the imported module.	M
ns-prefix	xsd:string	Defines the namespace prefix used in the module containing the import element to attach the namespace to the model element names. When writing a module, ns-prefix of the imported module should be used as the ns-prefix of the import element, unless there is a conflicting ns-prefix already in use in the module.	M
release	xsd:string maximum length 20. [a-zA-Z0-9]+ (\.[a-zA-Z0-9]+)*	Selects from which release of the module the definitions are imported.	M
description	xsd:string	See Section 5.1.3	O

5.3.2. Sub-Elements

Sub-Element	MinOccurs	MaxOccurs	Description
annotation	0	unbounded	See Section 5.19

5.3.3. Constraints

Model element references are only allowed to model elements in modules, which are directly or indirectly imported by the module in which the reference is given.

A module must not directly or indirectly import definitions from several releases of a module.

A module must not directly or indirectly import itself. That is, circular imports between modules are not allowed.

All namespace prefixes, including the prefix of the module itself, must be unique within the module.

5.3.4. XSD

```
<xsd:complexType name="importType">
   <xsd:sequence>
         <xsd:group ref="kalua:ModuleIdentityProperties"/>
         <xsd:group ref="kalua:ModelElementProperties"/>
   </xsd:sequence>
</xsd:complexType>
```

5.3.5. Element Examples

```
<import>
   <ns-uri>http://www.example.com/</ns-uri>
   <ns-prefix>example</ns-prefix>
   <release>2.1</release>
</import>
```

5.4. attribute

An "attribute" represents structural features of some kind of manageable resource. As such, "attributes" can be part of an attribute group, a class, or a structure.

An "attribute" can be addressed via its name. Each "attribute" name must be unique with respect to all "attributes" defined in the containing element (class, structure or attribute group) and the "attributes" inherited from superclasses and incorporated "attribute" groups.

The most important property of an "attribute" is its type. "Attributes" can have a primitive type (for example, string, long, or boolean) as well as a constructed data type, that is, a sequence type, structure type or an enumeration. It is possible to refer to named types (see Section 5.13.) via the type element as well as to define the type inline by using sequence, structure or primitive elements inside the attribute element.

5.4.1. Attributes

Attribute	Type [Default]	Description	Use
name	name-string (see Section 5.1.1)	The name of the attribute.	required

5.4.2. Leaf Sub-Elements

Sub-Element	Type [Default]	min oc.	max oc.	Description
presentation	xsd:string	0	1	The presentation name used for the attribute. This name should be used in GUI's when presenting the attribute to human end users.
description	xsd:string	0	1	The description of the attribute
mandatory	none	0	1	If present must be provided during creation of the owning entity. Otherwise providing a value is optional
optional	none	0	1	If present, a value might be provided during creation of the owning object. This is the default behavior.
read-only	none	0	1	if present, the attribute is read only. It neither possible to provide a value during creation of the owning object creation nor to change the value afterwards.

read-write	none	0	1	If present, the attribute can be read and written. This is the default behavior.
unchangeable	none	0	1	If present, the attribute might be or must be assigned a value during creation of the owning object (depending on the presence of "mandatory" or "optional"), but cannot be changed afterwards.
default ValueLiteral	xsd:string	0	1	Specifies the default value for the attribute. Default values can only be specified for primitive types. If the type is not 'string', a valid value literal must be used.
unit	xsd:string	0	1	Specifies the unit in which the value for this attribute is measured. In case a unit is specified for the attribute type, this is overwritten by this unit.

Only one of the elements "read-write", "read-only" or "unchangeable" might be present in an attribute element. Only "mandatory" or "optional" might be present (but not both). In case "read-only" is present, neither "mandatory" nor "optional" could be present.

5.4.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	See Section 5.19
constraint	0	unbounded	Constraints applied in the context of the attribute.
type	0	1	The named type of the attribute.
primitive	0	1	The inline defined primitive type of this attribute
structure	0	1	The inline defined structure type of this attribute.
sequence	0	1	The inline defined sequence type of this attribute.

5.4.4. XSD

```
   <xsd:element name="attribute" type="kalua:Attribute" minOccurs="0"
   maxOccurs="unbounded"/>

   <xsd:complexType name="Attribute">
      <xsd:sequence>
            <xsd:group ref="kalua:NamedElementProperties"/>
            <xsd:element name="mandatory" minOccurs="0">
                  <xsd:complexType>
                        <xsd:attribute name="value"
                             type="xsd:boolean" default="true"/>
                  </xsd:complexType>
            </xsd:element>
            <xsd:element name="readonly" minOccurs="0">
                  <xsd:complexType>
                        <xsd:attribute name="value"
                             type="xsd:boolean" default="true"/>
                  </xsd:complexType>
            </xsd:element>
            <xsd:group ref="kalua:Datatype"/>
            <xsd:element name="defaultValueLiteral"
                 type="xsd:string" minOccurs="0"/>
            <xsd:element name="unit"
                 type="xsd:string" minOccurs="0"/>
            <xsd:group ref="kalua:ConstrainableElementProperties"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
   </xsd:complexType>
```

5.4.5. Element Examples

```
<attribute name="frequency">
   <presentation>Frequency</presentation>
   <type>kalua:int</type>
    <defaultValueLiteral>1800<defaultValueLiteral>
    <description>
      Bearer channel frequency.
   </description >
</attribute>

<attribute name="location">
   <structure>
      <attribute name="x">
         <type>kalua:float</type>
       </attribute>
      <attribute name="y">
         <type>kalua:float</type>
       </attribute>
   </structure
</attribute>
```

5.4.6. NETCONF Payload Examples

```
<frequency>1900</frequency>

<location>
    <x>120.87</x>
   <y>97.334</y>
</location>
```

5.5. attribute-group

"attribute groups" are a means to bundle a set of attributes that are typically used together. For example, an "attribute group" for describing an IP address would bundle a string-typed attribute for the hostname and a short-typed attribute for the port. The main purpose of "attribute groups" is to be incorporated (used) by model elements that can contain attributes, so classes, structures and other attribute groups. Using an "attribute group" means that all the attributes that belong to the attribute group are imported into the using element. Attributes incorporated from an attribute group are otherwise treated as if they were directly inside incorporating element.

No 'is-a' relationship is established between the "attribute group" and a using class or structure. As the "attribute group" concept is

only addressing organizational purposes, "attribute groups" cannot be instantiated.

5.5.1. Attributes

Feature	Type	Description	Use
name	name-string (see Section 5.1.1)	The name of the attribute group. It must be unique within the containing module.	required

5.5.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
presentation	xsd:string	0	1	The presentation name used for the attribute group.
description	xsd:string	0	1	The description of the attribute group.

5.5.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
constraint	0	unbounded	Constraints that apply in the context of this attribute group.
attribute	0	unbounded	The attributes belonging to this attribute group.
use	0	unbounded	The attribute groups used by this attribute group
key	0	unbounded	Key definitions

5.5.4. XSD

```
<xsd:element name="attribute-group"
     type="kalua:AttributeGroup" minOccurs="0"
     maxOccurs="unbounded"/>

<xsd:complexType name="AttributeGroup">
   <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
         <xsd:group ref="kalua:AttributeContainer"/>
         <xsd:group ref="kalua:ConstrainableElementProperties"/>
   </xsd:sequence>
   <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
</xsd:complexType>
```

5.5.5. Element Examples

```
<attribute-group name="IpAddressable">
   <presentation>IP Address</presentation
   <description>
      IP address attributes
   </description>
   <attribute name="host">
      <type>kalua:string</type>
   </attribute>
   <attribute name="port">
      <type>kalua:unsignedShort</type>
    </attribute>
</attribute-group>
```

5.5.6. NETCONF Payload Examples

```
.
.
<host>www.example.com</host>
<port>30100</port>
.
.
```

5.6. structure

"structures" define cohesive sets of named properties of potentially varying type. In Kalua, "structure" elements define ordered sets of attribute elements. Each member attribute must have a name that is unique with respect to the other attributes contained in the "structure" or incorporated from used attribute groups.

A "structure" itself has no name. A "structure" can be either a sub-element of an attribute definition, sequence definition or a sub-element of a type definition. In case a structure is defined inside an attribute or sequence, the structure can be addressed only by the name of the owning attribute respectively the element index in the sequence.

In case a structure is part of a type definition, the type name is applied to the structure. Such structure types can be reused (referred to) in all contexts where a data type is expected.

Since structures are attribute containers, structures can use or incorporate attribute groups. All attributes defined in used attribute groups become members of the "structure" and can be treated as any other attribute directly defined in the structure.

Also keys can be defined for structures. In case the scope is global, the key attributes uniquely identify each "structure" instance within all instances of this "structure" definition. Keys with local scope identify "structure" instances only within the context of the containing object. Such keys are primarily used for structures that are used as elements of sequences. This allows to also addressing them via a key value.

5.6.1. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
description	name-string (see Section 5.1.1)	0	1	The description of the structure type.

5.6.2. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the structure.
constraint	0	unbounded	Constraints that apply in the context of this attribute group.
attribute	0	unbounded	The attributes belonging to this attribute group.
use	0..n	unbounded	The attribute groups used by this attribute group
key	0..n	unbounded	Key definitions

5.6.3. XSD

```
<xsd:element name="structure">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:group ref="kalua:ModelElementProperties"/>
               <xsd:group ref="kalua:AttributeContainer"/>
               <xsd:group
ref="kalua:ConstrainableElementProperties"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

5.6.4. Element Examples

```
<structure name="Point">
   <presentation>(x,y)</presentation>
   <attribute name="x">
       <type>kalua::double</type>
    </attribute>
   <attribute id="y">
       <type>kalua:double</type>
   </attribute>
</structure>
```

5.6.5. NETCONF Payload Examples

```
.
.
.
<point>
   <x>441.2</x>
   <y>172.5</y>
</point>
.
.
```

5.7. sequence

A "sequence" is a data structure that contains several objects of the same type that can be addressed by their position in the "sequence". Sequences are therefore an ordered collection.

Each "sequence" has a base type that determines the type of its elements. Constraints have a minimum length and a maximum length. By setting the maximum length to unbounded, it is possible to define sequences with arbitrary length.

Sequences are also constrainable elements. Constraints defined within a "sequence" are applied in the context of the "sequence" object, they are not implicitly applied to each member. However, since the member type can be defined inline, it is no problem to refine an exiting data type with additional constraints.

Sequence types have no name. A "sequence" can be either a sub-element of an attribute definition, another "sequence" definition, or a sub-element of a type definition. In case a "sequence" is defined inside an attribute or "sequence", the "sequence" type cannot be reused in another context. Instances can be addressed by the name of the owning attribute respectively the element index in the "sequence".

In case a "sequence" is part of a type definition, the type name is applied to the "sequence". Such "sequence" types can be reused (referred to) in all contexts where a data type is expected.

When representing sequence contents as XML elements, two cases have to be distinguished:

- In case the elementName attribute of the sequence element is specified, the contents of each element of the sequence is wrapped into an XML element with the given name. Making the "boundaries" of a sequence element value explicit in the serialized form is

sometimes needed in order to make sure that the original structure of the contents can be reconstructed from the serialized form. For example, a serialized form of a sequence of sequence of unbounded maximum length that does not demarcate the start and beginning of the elements of the inner sequence does not allow reconstructing the original input.

- In case elementName is not defined, each element is serialized as if it was directly contained in the owning element (typically an attribute). In effect, the sequence is "flattened".

5.7.1. Attributes

Attribute	Type [default]	Description	Use
minLength	xsd:unsignedInt [0]	The minimum sequence length	optional
maxLength	xsd:unsignedInt [unbounded]	The maximum sequence length	optional
elementName	name-string (see Section 5.1.1)	The name used for elements in the sequence data representation.	optional
ordered	xsd:boolean [false]	Tells if the sequence represents an ordered collection or rather a bag of elements. In the second case, the ordering of the elements is arbitrary and therefore has no meaning.	optional

5.7.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
description	xsd:string	0	1	The description of the sequence type.

5.7.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the attribute group.
constraint	0	unbounded	Constraints that apply in the context of this attribute group.
type	1	1	The element type

5.7.4. XSD

```
<xsd:element name="sequence">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:group ref="kalua:ModelElementProperties"/>
         <xsd:group ref="kalua:Datatype"/>
         <xsd:group ref="kalua:ConstrainableElementProperties"/>
      </xsd:sequence>
      <xsd:attribute name="minLength"
           type="xsd:nonNegativeInteger" default="0"/>
      <xsd:attribute name="maxLength" default="unbounded">
         <xsd:simpleType>
            <xsd:union memberTypes="xsd:nonNegativeInteger">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                     <xsd:enumeration value="unbounded"/>
                  </xsd:restriction>
                           </xsd:simpleType>
            </xsd:union>
         </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="ordered" type="xsd:boolean"
           default="true"/>
      <xsd:attribute name="elementName" type="xsd:NCName"/>
   </xsd:complexType>
</xsd:element>
```

5.7.5. Element Examples

```
<typedef name="point">

</typedef>

.
.
.
<attribute name="path">
    <sequence minLength="2" elementName="point">
         <structure>
               <presentation>(x,y)</presentation>
               <attribute name="x">
                      <type>kalua::double</type>
                </attribute>
               <attribute name="y">
                      <type>kalua:double</type>
               </attribute>
         </structure>
   </sequence>
   .
   .
</attribute>


<attribute name="usedSlots">
    <sequence maxLength="16" elementName="slot">
         <type>kalua:int</type>
   </sequence>
</typedef>

<attribute name="serialNumber">
    <sequence minLength="1" maxLenegth="3">
         <type>kalua:string</type>
   </sequence>
</typedef>
```

5.7.6. NETCONF Payload Examples

```
.
.
.
<path>
   <point>
         <x>441.2</x>
         <y>172.5</y>
   </point>
   <point>
         <x>441.2</x>
         <y>198.3</y>
   </point>
   <point>
         <x>343.8</x>
         <y>198.3</y>
   </point>
<path>
.
.
.
<usedSlots>
    <slot>2</slot>
   <slot>3</slot>
   <slot>8</slot>
   <slot>15</slot>
</usedSlots>
.
.
.
<serialNumbers>r6687120-01</serialNumber>
<serialNumbers>r6687124-07</serialNumber>
<serialNumbers>r6687201-03</serialNumber>
```

5.8. simple-type

"simple-type" elements are used to define new types that have simple values. However, like definitions of complex types (structures, sequences), they can be described and annotated. The unit in which a value of this simple type is measured can be stated and constraints can be specified for them.

The set of legal values for the "simple-type" is defined by enumeration named values (enumeration element), by restricting existing simple types (restriction), by combining simple types (union) or by referring to an existing named "simple-type" (type).

5.8.1. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
unit	xsd:string	0	1	The unit in which values of simple types is measured.

5.8.2. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the attribute group.
constraint	0	unbounded	Constraints that apply in the context of this attribute group.
type	0	1	Reference to an existing simple type. Allowing to refer to a complex datatype (structure, sequence) is not allowed in the scope of a simple type.
enum	0	1	Enumeration of simple type values.
union	0	1	Union of simple types.
restriction	0	1	Restriction of another simple type.

Exactly one of the type, enum, union, or restriction elements must be present in a "simple- type".

5.8.3. XSD

```
<xsd:element name="simpleType">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:group ref="kalua:ModelElementProperties"/>
               <xsd:group ref="kalua:simpleTypeDefinition"/>
               <xsd:element name="unit" type="xsd:string"
                    minOccurs="0"/>
               <xsd:group
                    ref="kalua:ConstrainableElementProperties"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:group name="simpleTypeDefinition">
   <xsd:sequence>
         <xsd:choice>
               <xsd:element name="restriction"
                    type="kalua:restrictionType"/>
               <xsd:element name="union" type="kalua:unionType"/>
               <xsd:element name="enum" type="kalua:enumType"/>
               <xsd:group ref="kalua:simpleTypeReference"/>
         </xsd:choice>
   </xsd:sequence>
</xsd:group>
```

5.9. enum

Enum elements define enumeration types, so types that are characterized by a fixed of named values. Each legal enumeration value is specified by an enum-literal element.

As a simple-type, enumerations do not have an own description, annotations or constraints, the ones defined inside the owning simple-type element apply implicitly to the enumeration type.

5.9.1. Sub-Elements

Sub-Element	min occurs	max occurs	Description
enum-literal	0	unbounded	The enumeration values.

5.9.2. XSD

```
<xsd:complexType name="enumType">
   <xsd:sequence>
         <xsd:element name="enum-literal"
              type="kalua:enum-literalType"
              maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

5.9.3. Element Examples

```
<typedef name="AdministrativeState">
   <simple-type>
          <enum>
               <enum-literal name="unknown"/>
               <enum-literal name="locked"/>
               <enum-literal name="shuttingDown"/>
               <enum-literal name="unlocked"/>
          </enum>
    </simple-type>
</typedef>
.
.
.
<attribute name="adminState">
    <type>AdministrativeState</type>
    .
      .
</attribute>
```

5.9.4. NETCONF Payload Examples

```
<adminState>locked</adminState>
```

5.10. enum-literal

An "enum-literal" defines one of the values that can be assigned to an enumeration that is defined by the containing enum element Each "enum literal" must have a different name.

In addition to the name and presentation that can be specified for the literal, also a value may be provided. This value is used to represent the enum value in the implementing system.

This could be useful in case a simple network element represents enumeration values as numbers. However, "enum literal" values should not be used when an enumeration type is defined that is standardized or otherwise implementation agnostic. E.g., the administrative state as defined in X.731 defines the values "unknown", "locked" , "shuttingDown" and "unlocked", so these terms are best used as literal names. However, is does not mandate any particular storage representation, so none should be given in the definition of the according "enum literals".

The presence of the value attribute also controls the enum value representation:

- o In case the value attribute is not used, the name of the "enum literal" is used to represent the enum value

- o In case the value attribute was assigned a value, that value is used to represent the literal.

The value attribute must be used consistently. Either all "enum-literal" elements have a value attribute or none.

5.10.1. Attributes

Feature	Type	Description	Use
name	xsd:string	The name of the attribute. It must be unique within the set of attributes which is the union of the attributes defined in the same attribute container (class, structure, attribute-group), the attributes inherited from superclasses (class) and the set of attributes incorporated from attribute groups (class, structure, attribute-group).	required
value	xsd:string	The string representation of the storage value of the enum literal	optional

5.10.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
presentation	xsd:string	0	1	The presentation name used for the attribute group.
description	xsd:string	0	1	The description of the attribute group.

5.10.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the attribute group.

5.10.4. XSD

```
<xsd:element name="enum-literal"
type="kalua:enum-literalType" maxOccurs="unbounded"/>

<xsd:complexType name="enum-literalType">
   <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
   </xsd:sequence>
   <xsd:attribute name="value"/>
   <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
</xsd:complexType>
```

5.10.5. Element Examples

```
<attribute name="processState">
    <simple-type>
         <enum>
           <enum-literal name="notStarted" value="0">
                     <presentation>Not started</presentation>
           </enum-literal>
           <enum-literal name="running" value="1">
                     <presentation>Running</presentation>
           </enum-literal>
           <enum-literal name="suspended" value="2">
                 <presentation>Suspended</presentation>
           </enum-literal>
           <enum-literal name="stopped" value="8">
                 <presentation>Stopped</presentation>
           </enum-literal>
         </enum>
       </simple-type>
    .
    .
</attribute>
```

5.10.6. NETCONF Payload Examples

```
.
.
<processState>1</processState>
.
.
.
<processState>8</processState>
```

5.11. union

A "union" combines two or more simple types. The set of legal values of this type is the "union" of all sets of legal values of each included simple type.

As a simple type, "unions" do not have an own description, annotations or constraints, the ones defined inside the owning simple-type element apply implicitly to the enumeration type.

5.11.1. Sub-Elements

Sub-Element	min occurs	max occurs	Description
type	0	unbounded	A named simple type that is part of the union type.
enumeration	0	unbounded	An enumeration that is part of the union type.
restriction	0	unbounded	A restricted simple type that is part of the union type.
union	0	unbounded	A subordinate union type that is part of this union type.

5.11.2. XSD

```
<xsd:complexType name="unionType">
   <xsd:sequence>
         <xsd:group
              ref="kalua:simpleTypeDefinition"
              maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

5.11.3. Element Examples

```
<attribute name="voltage">
   <simple-type>
         <union>
               <restriction>
                     <type>kalua:int</type>
                     <minInclusive value="100">
                     <maxInclusive value="240">
               <restriction>
               <enum>
                     <enum-literal name="none" value="0">
                </enum>
         </union>
    </simple-type>
    .
    .
</attribute>
.
.
```

5.11.4. NETCONF Payload Examples

```
<voltage>110</adminState>
.
.
<voltage>230</adminState>
.
:
<voltage>0</adminState>
```

5.12. restriction

A "restriction" element specifies a restricted simple type. That is done by applying restriction facets to the contained or referred base simple type.

It is possible to apply multiple restriction facets. A legal value for the restricted type must comply with all "restrictions", including the "restrictions" already applied to the base type.

The restriction facets supported by Kalua are a subset of the restriction facets as defined in 'XML Schema 1.1 Part 2: Datatypes'.

5.12.1. Sub-Elements

Sub-Element	min occurs	max occurs	Description
type	0	1	The restricted named base type
enumeration	0	1	The restricted enumeration base type
restriction	0	1	The further restricted base type.
union	0	1	The restricted union type.

5.12.2. Restriction Facet-Elements

Sub-Element	min occurs	max occurs	Description
minInclusive	0	unbounded	Inclusive lower bound for a numerical base type.
minExclusive	0	unbounded	Exclusive lower bound for a numerical base type
maxInclusive	0	unbounded	Inclusive upper bound for a numerical base type.
maxExclusive	0	unbounded	Exclusive upper bound for a numerical base type
totalDigits	0	unbounded	The total digits of a decimal base type
fractionDigits	0	unbounded	The fraction digits of a decimal base type
length	0	unbounded	The length of a string base type
minLength	0	unbounded	The minimum length of a string base type
maxLength	0	unbounded	The maximum length of a string base type

pattern	0	unbounded	A regular expression that must be matched. Applies to string base type

5.12.3. XSD

```
<xsd:element name="restriction" type="kalua:restrictionType"/>

<xsd:complexType name="restrictionType">
   <xsd:sequence>
       <xsd:group ref="kalua:simpleTypeDefinition"/>
      <xsd:group ref="kalua:facets"
           minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:group name="facets">
   <xsd:choice>
      <xsd:element name="minExclusive" type="kalua:facet"
           id="minExclusive"/>
      <xsd:element name="minInclusive" type="kalua:facet"
           id="minInclusive"/>
      <xsd:element name="maxExclusive" type="kalua:facet"
           id="maxExclusive"/>
      <xsd:element name="maxInclusive" type="kalua:facet"
           id="maxInclusive"/>
      <xsd:element name="totalDigits" id="totalDigits">
         <xsd:complexType>
            <xsd:complexContent>
               <xsd:restriction base="kalua:numFacet">
                  <xsd:attribute name="value"
                       type="xs:positiveInteger"
                          use="required"/>
               </xsd:restriction>
            </xsd:complexContent>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="fractionDigits" type="kalua:numFacet"
           id="fractionDigits"/>
      <xsd:element name="length" type="kalua:numFacet" id="length"/>
      <xsd:element name="minLength"
           type="kalua:numFacet" id="minLength"/>
      <xsd:element name="maxLength"
           type="kalua:numFacet" id="maxLength"/>
      <xsd:element name="pattern" type="kalua:facet" id="pattern"/>
   </xsd:choice>
```

```
</xsd:group>
```

5.12.4. Element Examples

```
<attribute name="voltage">
   <simple-type>
         <union>
               <restriction>
                     <type>kalua:int</type>
                     <minInclusive value="100">
                     <maxInclusive value="240">
               <restriction>
               <enum>
                     <enum-literal name="none" value="0">
                </enum>
         </union>
    </simple-type>
    .
    .
</attribute>
.
.
```

5.13. typedef

The "typedef" element is used to give otherwise anonymous datatypes a name which can be used to refer to this type wherever a datatype is required. This is done by using the type name as body value of the type element.

5.13.1. Attributes

Feature	Type	Description	Use
name	name-string	The name of the type. It must be unique within the containing module.	required

5.13.2. Sub-Elements

Sub-Element	min occurs	max occurs	Description
structure	0	1	The structure type to give a name.
sequence	0	1	The sequence type to give a name.
simple-type	0	1	A simple-type that is given a name.
type	0	1	Provides an alias for the referred named type.

Exactly one of the structure, sequence, simple-type or type elements must be specified as child of the typedef element.

5.13.3. XSD

```
<xsd:element name="typedef"
     type="kalua:typedefType"
     minOccurs="0" maxOccurs="unbounded"/>

<xsd:complexType name="typedefType">
   <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
         <xsd:group ref="kalua:Datatype"/>
   </xsd:sequence>
   <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
</xsd:complexType>
```

5.13.4. Element Examples

```
<typedef name="complex">
   <structure>
         <attribute name="real">
                <type>kalua:double</type>
          ...
       </attribute>
         <attribute name="imag">
                <type>kalua:double</type>
          ...
       </attribute>
   </structure>
</typedef>
```

5.13.5. NETCONF Payload Examples

```
.
.
<phaseAmplitude>
    <real>0.73001</real>
   <imag>0.239</imag>
</phaseAmplitude>
.
.
```

5.14. use

"use" elements specify what attribute groups are used in the owning attribute container.

All attributes of the used (incorporated) attribute group become part of the containing element. Using an attribute from an attribute group is equivalent with defining an attribute directly as part of the container.

Therefore, it is not allowed that attributes incorporated from an attribute group have the same name as an attribute that is already part of the container namespace.

Since attribute groups are only organizing facilities, no "is-a relationship" is established between the used attribute group and the using container (class, structure).

Note that also attribute groups themselves can use other attribute groups.

"use" elements are processed in the order as they are defined in

their attribute container.

5.14.1. Attributes

Feature	Type	Description	Use
attribute-group	name-string	The reference of the used attribute group.	required

5.14.2. Sub-Elements

Sub-Element	min occurs	max occurs	Description
description	0	1	The description of the use element.
annotation	0	unbounded	Annotations attached to the use element.

5.14.3. XSD

```
<xsd:element name="use" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:group ref="kalua:ModelElementProperties"/>
         </xsd:sequence>
         <xsd:attribute name="attribute-group"
              type="kalua:ModelElementReference"/>
   </xsd:complexType>
</xsd:element>
```

5.14.4. Element Examples

```
<attribute-group name="IpAddressable">
   <presentation>IP Address</presentation>
   < description >
      IP address attributes
   </description >
   <attribute name="host">
      <type>kalua:string</type>
   </attribute>
   <attribute name="port">
      <type>kalua:unsignedShort</type>
   </attribute>
</attribute-group>

<class name="router">
   <use attribute-group="IpAddressable">
   .
   <attribute name="connectedPorts" ...>
   .
</class>
```

5.14.5. NETCONF Payload Examples

```
.
<router>
.
.
   <host>www.example.com</host>
   <port>30100</port>
   <connectedPorts>12</connectedPorts>
.
.
</router>
```

5.15. key

"key" elements specify which attributes belonging to an attribute container (class, structure, attribute group) form a "key". A "key" is a set of one or more distinct member attributes. Member attributes could be all attributes that cannot be left unset. Attributes directly defined in the scope of a class or group can be combined with attributes inherited from superclasses or incorporated from attribute groups. An attribute could be part of multiple keys.

One important aspect of a "key" is its scope:

- global: indicates a globally unique key
- local: indicates that the key only identified instances in the scope of their containing (scoping) object

"Keys" can have slightly different semantics depending in which context they are defined respectively used. In effect, the context of use also determines which scopes can be used:

- A "key" that is part of a class uniquely identifies the instances of that class any meaningful context of use. For example, a "key" of a network element class must identify its instances (NE's) in the whole network. A "key" for a mobile equipment class must uniquely identify the equipment among all other mobile equipments (so an attribute capable of holding the 15 digit IMEI might do the job).
- In contrast to that, "keys" that are part of structures may only uniquely identify the structure instance inside its containing object. In this case the scope of the key is 'local'. It is also possible that a "key" uniquely identifies the structure instance among all other instances. In this case, the scope of the "key" is 'global'.
- In case a "key" is defined in an attribute group, its exact semantics is undefined. A concrete semantics is applied by using the attribute group in a class or structure.

The member attributes of a "key" are enumerated by the member elements contained in the "key" element.

5.15.1. Attributes

Attribute	Type	Description	Use
scope	xsd:enumeration	The reference of the used attribute group.	required

5.15.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
description	xsd:string	0	1	The description of the key.
member	name-string	0	unbounded	The names of the member attributes.

5.15.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the use element.
member	1	unbounded	The members of the key

5.15.4. XSD

```
<xsd:element name="key" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:group ref="kalua:ModelElementProperties"/>
         <xsd:element name="member" type="xsd:string"
              maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="scope" default="local">
         <xsd:simpleType>
               <xsd:restriction base="xsd:string">
                <xsd:enumeration value="local"/>
                <xsd:enumeration value="global"/>
             </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
   </xsd:complexType>
</xsd:element>
```

5.15.5. Element Examples

```
<class name="MobileEquipment">
   <description>Mobile Equipment</description>
   <attribute name="vendor">
     ...
   </attribute>
   <attribute name="serialNr">
      <mandatory>
      <type>kalua:string</type>
   </attribute>

   <attribute name="imei">
      <mandatory>
      <type>kalua:string</type>
   </attribute>
   .
   .
   <key scope="global">
      <member>imei</member>
   </key>
   <key scope="global">
      <member>vendor</member>
      <member>serialNr</member>
   </key>

</class>
```

5.15.6. NETCONF Payload Examples

```
<MobileEquipment>
.
.
<vendor>SpaceMobil</vendor>
<serialnr>23-2308263673</serialnr>
<imei>800282737266302</imei>
.
.
</MobileEquipment>
```

5.16. constraint

"constraint" elements can be used to formulate constraints that are applied in the scope of the containing element.

For example, "constraints" defined as child elements of an attribute element constrain the set of values that can be assigned to that attribute. "Constraints" defined as sub- elements of attribute containers (structures, attribute groups, classes) can express restrictions that values of one attribute can have to other attributes in that container. "Constraints" defined as part of relationships allows narrowing the set of instances that can actually be associated by that relationship. "Constraints" defined in classes could even navigate via relationship ends to other object in order to express constraints between instances of different classes.

The "constraint" expression is specified as body value of the expression element. Therefore, exactly one expression element must be present.

5.16.1. Attributes

Feature	Type	Description	Use
name	name-string	The name of the constraint.	required

5.16.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
description	xsd:string	0	1	The description of constraint
expression	xsd:string	0	1	The constraint expression.

5.16.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the attribute group.

5.16.4. XSD

```
<xsd:element name="constraint"
     minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:group ref="kalua:ModelElementProperties"/>
               <xsd:element name="expression"
                    type="xsd:string"/>
         </xsd:sequence>
         <xsd:attribute name="kind"
              type="xsd:string"/>
   </xsd:complexType>
</xsd:element>
```

5.16.5. Element Examples

```
<constraint name="PermittedPowerConsumption">
   <expression>@voltage * @ampere <= @maxPower</expression>
</constraint>
```

5.17. class

"Classes" are used to describe objects that have an own identity and a potentially independent life cycle. "Classes" can represent concrete manageable resources in the network such as specific types of network elements or abstract concepts (e.g. managed objects or network resources).

A "class" can have one superclass at the maximum. From the superclass a "class" does not only inherit all attributes and relationships, but also an implicit 'is-a' relationship is established between the "class" and its super class. This means that wherever the super class is used or referred to, also the derived "class" can be used.

Kalua supports only single inheritance. This is to avoid the huge complexity caused by inheriting from multiple base classes.

For reusing sets of attribute definitions that describe a certain aspect of an object, a "class" can use 0..n attribute groups (see Section 5.5). This means that all attributes of a used attribute group become members of the using "class". This does NOT mean that an 'is-a' relationship is established between the "class" and its base attribute groups.

Instance of classes must be uniquely identifiable in case they are associated by reference relationships, so some kind of key needs to be available. Such a key is obtained in the following way:

- o If a key with global scope is defined in the "class" directly, inherited from a super class, or incorporated from an attribute group, this key is used.

- o In case a class is contained in another class and a key with local scope is defined, the global scope key for the actual class is composed from the key of the owing class and the own local scope key.

- o In case the class not contained in any another class and has a local key, this is used, as it is assumed that a NETCONF agent is able to uniquely identify the instance in its given context.

In order to describe how class instances can be accessed, class definitions can contain a max-access element. Its body text value can have one of the following values:

- o not-accessible: The class is used only for internal purposes. Instances cannot be accessed in any way.

- o accessible-for-notify: Only notifications are generated when instances are created, modified or deleted.

- o read-only: It is only possible to read the actual state of instances of this class.

- o read-write: Instances of this class can be read and written, but not created.

- o read-create: It is possible to create, write, and read instances.

In case instances of a class are readable, notifications might also be send when they are created, changed or deleted.

5.17.1. Attributes

Feature	Type	Description	use
name	name-string	The name of the class.	required

5.17.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
description	xsd:string	0	1	The description of the class
presentation	xsd:string	0	1	The presentation name used for the class respectively instances of the class.
abstract	xsd:boolean	0	1	If present, the class is abstract.
superclass	name-string	0	1	The name of the superclass.
max-access	xsd:enumeration			Specifies how class instances can be accessed.

5.17.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the class.
constraint	0	unbounded	Constraints that apply in the context of this class.
attribute	0	unbounded	The attributes directly declared in the class.
use	0	unbounded	The attribute groups used by the class.
key	0	unbounded	Key definitions

5.17.4. Constraints

A class must not be its own super class - neither directly nor indirectly. This implies a cycle in the inheritance graph and does not have well-defined semantics.

If a class is abstract and inherits from a superclass, this superclass must be abstract as well.

A class must not inherit from an attribute group that is already inherited by one of its ancestor classes (that is, its super class or the super class of the super class, and so on).

5.17.5. XSD

```
<xsd:element name="class" type="kalua:Class" minOccurs="0"
maxOccurs="unbounded"/>

<xsd:complexType name="Class">
   <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
         <xsd:element name="abstract" minOccurs="0">
               <xsd:complexType/>
         </xsd:element>
         <xsd:element name="superClass"
              type="kalua:ModelElementReference"
              minOccurs="0">
         </xsd:element>
         <xsd:group ref="kalua:AttributeContainer"/>
         <xsd:group ref="kalua:ConstrainableElementProperties"/>
   </xsd:sequence>
   <xsd:attributeGroup
        ref="kalua:NamedElementAttributes"/>
</xsd:complexType>
```

5.17.6. Element Examples

```
<class
   id="Site"
   presentation="Site">
   <description>
      A site at which some network resources are located
   </description >
   <attribute name="name">
      <type>kalua:string</type>
   </attribute>
   <attribute name="street">
      <mandatory>
      <type>kalua:string</type>
   </attribute>
   <attribute name="city">
       <type>kalua:string</type>
   </attribute>
   <attribute name="zipCode">
      <type>kalua:unsigedInt</type>
   </attribute>
   <attribute name="location">
      <struct>
         <attribute id="longitude">
            <type>kalua:string</type>
         </attribute>
         <attribute id="latitude" type="kalua:double">
            <type>kalua:string</type>
         </attribute>
         <attribute id="altitude"  type="kalua:double">
            <type>kalua:string</type>
         </attribute>
      </struct>
   </attribute>
   <key>
      <member>name</member>
   </key>
</attribute-group>
```

5.17.7. NETCONF Payload Examples

```
.
.
<Site>
   <name>RANC</host>
   <street>Alphabet street 123</street>
   <city>Megalopolis</city>
   <zipCode>65432</zipCode>
   <location>
         <longitude>23o48'7''</longitude>
         <latitude>56o23'45''</latitude>
         <altitude>125m</altitude>
   </location>
</Site>
.
.
```

5.18. relationship

"relationship" elements specify associations between two classes or an attribute group and a class. With "relationships", it is possible to describe the way instances of particular classes relate to each other. This covers the simple 'is-used-by' or 'is-managed-by' "relationship" as well as containment "relationships," such as the well-known parent-child "relationship" between managed objects. Three different types of relationships are distinguished:

- reference: Simple bi-directional references between instances of two classes. For example, a 'managed-by' "relationship" indicates that the instance at the source end is managed by the instance at the target end.

- containment: A containment "relationship" in which the instance at the source end contains all instances at the target end. This also means that if the containing object is deleted, all contained objects are also deleted.

- calculated: Dynamically calculated "relationships". Instances of the classes at the source and target end are related if they are in the result set produced by the evaluation of a "relationship" expression at runtime. With this type of "relationship" you can define that two objects are related if they have the same parent in the containment tree and a particular attribute has the same value in both instances.

A "relationship" can also refine an existing "relationship". This feature is usually used to narrow down the usage of a generic

"relationship" inherited by the source and target end class. For example, an abstract class 'Resource' has a relationship 'managedResources' that refers to all resources managed by a particular resource. If two concrete classes, 'ProtectionGroup' and 'ProtectionUnit', inherit from 'Resource', you can refine the generic relationship 'managedResources' by creating a "relationship" definition 'managedUnits' between 'ProtectionGroup' and 'ProtectionUnit' that has 'managedResource' as its base relationship. This would mean that a protection group could only manage protection units and no other types of resources.

There are two main reasons to specify abstract relationships that are refined by concrete relationships:

- o The generic "relationship" can be used to navigate between object instances without needing to know what type of refined relationships exist for each concrete class. In the example above, this means that an application that only deals with resources can find out the managed units of a protection group instance by following the 'managedResources' relationship.

- o The system that has to store and restore class instances can take advantage of the fact that a "relationship" refines another one by reusing storage space.

In the example above this means that if 'ProtectionGroup' and 'ProtectionUnit' instances are stored in a relational database, the foreign key columns that are used to address the managing resource can be reused for addressing the controlling protection group.

5.18.1. Attributes

Attribute	Type	Description	Use
name	xsd:string	The name of the relationship	required

5.18.2. Leaf Sub-Elements

Sub-Element	Type	min oc.	max oc.	Description
description	xsd:string	0	1	The description of the relationship.
kind	xsd:enumeration	1	1	The kind of relationship (reference, containment, calculated)
readonly	xsd:boolean	0	1	Tells if relationship can be modifier, which is the default, or only read.
base Relationship	name-string	0	1	The name of the base relationship

5.18.3. Sub-Elements

Sub-Element	min occurs	max occurs	Description
annotation	0	unbounded	Annotations attached to the relationship
constraint	0	unbounded	Constraints that apply in the context of this attribute group.
source	1	1	The specification of the source-end of the relationship.
target	1	1	The specification of the target-end of the relationship

5.18.4. Source and Target Leaf Sub-Elements

The table below describes the simple typed XML elements that have to appear in source and target elements of a relationship element.

Sub-Element	Type	min oc.	max oc.	Description
class	name-string	1	1	The class at one of the ends of the relationship.
role	name-string	1	1	The role name of a relationship-end-class
minCardinality	xsd:unsignedLong	1	1	The minimum number of objects that are addressed at the source or target end of this relationship. Typical values for the minimum cardinality are 0 (the target endpoint can remain undefined) or 1 (the target endpoint must be defined).
maxCardinality	xsd:unsignedLong	1	1	The maximum number of objects that are addressed at the source or target end of this relationship. The value "unbounded" can be used to denoted an unlimited number of objects at the given relationship end.

5.18.5. Constraints

Several constraints must be fulfilled by an relationship:

- A valid source end multiplicity requires that either the maximum cardinality is unbounded or the minimum cardinality is not greater than the maximum cardinality.

- The source end multiplicity specification must allow that at least one object can be addressed at the source end, so source-end max cardinality must be greater than zero.

- A valid target end multiplicity requires that either the maximum cardinality is unbounded or the minimum cardinality is not greater than the maximum cardinality.

- The target end multiplicity specification must allow that at least one object can be addressed at the source end, so target-end max cardinality must be greater than zero.

- An object can only be contained in another object at the same point in time. So in case of an containment relationship, it implies that the source-end max cardinality is one.

When refining relationships, several constraints have to be considered:

- If the relationship refines a base relationship, the source end class must be equal or derived from the base relationship source end class.

- If the relationship refines a base relationship, the target end class must be equal or derived from the base relationship target end class.

- If the relationship refines a base relationship, the relationship type must be the same as for the base relationship.

- If the relationship refines a base relationship, the target minimum cardinality must be equal or greater than the target minimum cardinality at the base relationship.

- If the relationship refines a base relationship, the target maximum cardinality must be equal or smaller than the target maximum cardinality at the base relationship.

- If the relationship refines a base relationship, the source minimum cardinality must be equal or greater than the source minimum cardinality at the base relationship.

- If the relationship refines a base relationship, the source maximum cardinality must be equal or smaller than the source maximum cardinality at the base relationship.

5.18.6. XSD

```
  <xsd:element name="relationship"
     type="kalua:Relationship" ="0" maxOccurs="unbounded"/>

  <xsd:complexType name="Relationship">
    <xsd:sequence>
       <xsd:group ref="kalua:NamedElementProperties"/>
       <xsd:element name="baseRelationship"
            type="kalua:ModelElementReference"
            minOccurs="0"/>
       <xsd:choice minOccurs="0">
            <xsd:element name="read-only">
                 <xsd:complexType/>
            </xsd:element>
            <xsd:element name="read-write">
                 <xsd:complexType/>
            </xsd:element>
       </xsd:choice>
       <xsd:element name="kind">
            <xsd:complexType>
                 <xsd:choice>
                      <xsd:element name="containment"/>
                      <xsd:element name="reference"/>
                      <xsd:element name="calculated">
                           <xsd:complexType>
                                <xsd:sequence>
                                     <xsd:element name="condition">
                                        <xsd:complexType>
                                             <xsd:simpleContent>
                                                  <xsd:extension
                                                   base="xsd:string">
                                                   <xsd:attribute
                                                   name="language"
                                          type="xsd:normalizedString"/>
                                                  </xsd:extension>
                                             </xsd:simpleContent>
                                        </xsd:complexType>
                                    </xsd:element>
                                 </xsd:sequence>
                            </xsd:complexType>
                      </xsd:element>
                 </xsd:choice>
            </xsd:complexType>
       </xsd:element>
       <xsd:element name="source" type="kalua:RelationshipEnd"/>
       <xsd:element name="target" type="kalua:RelationshipEnd"/>
       <xsd:group ref="kalua:ConstrainableElementProperties"/>
```

```
    </xsd:sequence>
    <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
  </xsd:complexType>

  <xsd:complexType name="RelationshipEnd">
    <xsd:sequence>
          <xsd:element name="class" type="kalua:ModelElementReference"/>
          <xsd:element name="role" type="kalua:nameType"/>
          <xsd:element name="minCardinality"
          type="xsd:nonNegativeInteger" default="0" minOccurs="0"/>
          <xsd:element name="maxCardinality"
               default="unbounded" minOccurs="0">
                <xsd:simpleType>
                      <xsd:union memberTypes="xsd:nonNegativeInteger">
                            <xsd:simpleType>
                                  <xsd:restriction base="xsd:string">
                                        <xsd:enumeration
                                        value="unbounded"/>
                                  </xsd:restriction>
                            </xsd:simpleType>
                      </xsd:union>
                </xsd:simpleType>
          </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

5.18.7. Element Examples

```
  <relationship name="ip_interface">
     <kind>
       <calculated>
         <condition>$source/ipAdEntIfIndex=$target/ifIndex</condition>
       </calculated>
     </kind>
     <source>
           <class>ipAddrEntry</class>
           <role>ipAddress</role>
           <minCardinality>0</minCardinality>
           <maxCardinality>1</maxCardinality>
     </source>
     <target>
           <class>ifEntry</class>
           <role>interface</role>
           <minCardinality>1</minCardinality>
           <maxCardinality>1</maxCardinality>
     </target>
  </relationship>
```

```
     <!--
        Small fraction of
        TMF GB922 - SID Consolidated Model
        Version 7.0, 2006 Telemanagement Forum
      -->

  <class name="RootEntity">
     <attribute name="objectID">
         <mandatory/>
         <type>kalua:string</type>
     </attribute>
     <attribute name="commonName">
         <mandatory/>
         <type>kalua:string</type>
     </attribute>
     <attribute name="description">
         <optional/>
         <type>kalua:string</type>
     </attribute>
     <key scope="global">
         <member>objectId</member>
     </key>
     <key scope="local">
         <member>commonName</member>
     </key>
  </class>

  <class name="Entity">
     <abstract/>
     <super-class>RootEntity</super-class>
     <attribute name="version">
         <optional/>
         <type>kalua:string</type>
      </attribute>
  </class>

  <typedef name"ManagementMethod">
     <simple-type>
        <enum>
           <enum-literal name="Unknown"     value="0"/>
           <enum-literal name="None"        value="1"/>
           <enum-literal name="CLI"         value="2"/>
           <enum-literal name="SNMP"        value="3"/>
           <enum-literal name="TL1"         value="4"/>
           <enum-literal name="CMIP"        value="5"/>
           <enum-literal name="Proprietary" value="6"/>
        </enum>
     </simple-type>
```

```
</typedef>

<class name="ManagedEntity">
   <abstract/>
   <super-class>Entity</super-class>
   <attribute name="managementMethodCurrent">
       <type>ManagementMethod</type>
   </attribute>
   <attribute name="managementMethodSupported">
      <sequence minLength="1" maxLength="5">
         <type>ManagementMethod</type>
      </sequence>
   </attribute>
</class>


<class name="Resource">
   <abstract/>
   <super-class>ManagedEnity</super-class>
   <attribute name="usageState">
      <simple-type>
         <enum>
            <enum-literal name="Unknown" value=""/>
            <enum-literal name="NotInstalled" value=""/>
            <enum-literal name="Installed" value=""/>
            <enum-literal name="Inactive" value=""/>
            <enum-literal name="Idle" value=""/>
            <enum-literal name="Active" value=""/>
            <enum-literal name="Busy" value=""/>
         </enum>
      </simple-type>
   </attribute>
</class>

<class name="PhysicalResource">
   <abstract/>
   <super-class>Resource</super-class>
   <attribute name="manufactureDate">
      <type>kalua:dateTime</type>
   </attribute>
   <attribute name="otherIdentifier">
      <type>kalua:string</type>
   </attribute>
   <attribute name="powerState">
      <simple-type>
         <enum>
            <enum-literal name="" value=""/>
         </enum>
```

```
        </simple-type>
     </attribute>
     <attribute name="serialNumber">
        <mandatory/>
        <type>kalua:string</type>
     </attribute>
     <attribute name="versionNumber">
        <type>kalua:string</type>
     </attribute>
  </class>


  <class name="LogicalResource">
     <abstract/>
     <super-class>Resource</super-class>
     <attribute name="lrStatus">
        <simple-type>
           <enum>
              <enum-literal name="Unknown" value="0"/>
              <enum-literal name="OK" value="^1"/>
              <enum-literal name="Initializing" value="2"/>
              <enum-literal name="Starting" value="3"/>
              <enum-literal name="Paused" value="4"/>
              <enum-literal name="Stopping" value="5"/>
              <enum-literal name="Stopped" value="6"/>
              <enum-literal name="Degraded" value="7"/>
              <enum-literal name="Stressed" value="8"/>
              <enum-literal name="PredictedFailure" value="9"/>
              <enum-literal name="ErrorGemeral" value="10"/>
              <enum-literal name="ErrorNotRecoverable" value="11"/>
              <enum-literal name="NotInstalledOrNotPresent" value="12"/>
              <enum-literal name="InMaintenance" value="13"/>
              <enum-literal name="UnableToContact" value="14"/>
              <enum-literal name="LostCommunications" value="15"/>
           </enum>
        </simple-type>
     </attribute>
     <attribute name="serviceState">
         <simple-type>
            <enum>
               <enum-literal name="Unkown" value="0"/>
               <enum-literal name="InService" value="1"/>
               <enum-literal name="OutOfService" value="2"/>
               <enum-literal name="Testing" value="3"/>
               <enum-literal name="InMaintenance" value="4"/>
               <enum-literal name="NotAvailable" value="5"/>
               <enum-literal name="NotApplicable" value="6"/>
            </enum>
```

```
        </simple-type>
    </attribute>
    <attribute name="isOperational">
        <mandatory/>
        <type>kalua:boolean</type>
    </attribute>
</class>



<relationship name="PResourceSupportsLResource">
    <kind>
        <reference/>
    </kind>
    <source>
          <class>PhysicalResource</class>
          <role>physicalResource</role>
          <minCardinality>0</minCardinality>
          <maxCardinality>unbounded</maxCardinality>
    </source>
    <target>
          <class>LogicalResource</class>
          <role>logicalResource</role>
          <minCardinality>0</minCardinality>
          <maxCardinality>unbounded</maxCardinality>
    </target>
</relationship>
```

5.18.8. NETCONF Payload Examples

```
 <!-- Card is a PhysicalResource -->
   .
 <Card>
   .
   .
    <objectId>NW-Card-11783</objectId>
    <commonName>C12</commonName>
      .
      .
    <serialNumber>N-737362183-34</serialNumber>
      .
      .
    <PResourceSupportsLResource>
       <logicalResource kalua:type="TerminationPoint">
          <objectId>TP-83838</objectId>
       </logicalResource>
```

```
         <logicalResource>
            <!-- Kalua:type is optional -->
            <objectId>TP-83845</objectId>
         </logicalResource>
      </PResourceSupportsLResource>

   </Card>



   <!-- TerminationPoint is a LogicalResource -->

   <TerminationPoint>
      .
      .
      <objectId>TP-83838</objectId>
      <commonName>IP-TP-3</commonName>
        .
        .
      <isOperational>true</isOperational>
        .
        .
      <PResourceSupportsLResource>
         <physicalResource kalua:type="Card">
            <objectId>NW-Card-11783</objectId>
         </Card>
      </PResourceSupportsLResource>

   </TerminationPoint>


   <TerminationPoint>
      .
      .
      <objectId>TP-83845</objectId>
      <commonName>IP-TP-10</commonName>
       .
       .
      <isOperational>false</isOperational>
       .
       .
      <PResourceSupportsLResource>
         <physicalResource kalua:type="ManagedHardware">
           <!-- ManagedHardware is superclass of Card -->
           <objectId>NW-Card-11783</objectId>
         </physicalResource>
      </PResourceSupportsLResource>
```

</TerminationPoint>

5.19. annotation

Kalua supports an "annotation" mechanism to allow extensions to the model language, "annotation" is a set of additional properties associated with a model element. There can be several "annotations" associated with the same model element.

"Annotations" are 'typed', that is, each "annotation" must instantiate a particular annotation type also defined in the module or in a directly or indirectly imported module. The annotation type specifies which entries can be or must be present in an "annotation". The annotation type also implies semantics of the annotation data; however, semantics are not modeled and are conveyed in the description within the annotation-type definition only.

An annotation type is defined with an annotation-type element. Reader of the module must not treat unrecognized annotation types as errors.

A model element is annotated by adding an annotation sub-element to the element definition. This element contains any number of annotation-property elements, which are pairs made up of a name and a value. The order of annotation-property elements has no semantic value.

The name and values in the annotation-property elements must match a corresponding element specification contained in the annotation-property elements of the referred annotation-type element.

5.19.1. Element Attributes

Attribute	Type [Default]	Description	S
name	xsd:string	Defines type of the annotation.	M

5.19.2. Sub-Elements

Sub-Element	MinOccurs	MaxOccurs	Description
e: annotation-property	0	unbounded	Defines values for the properties of the annotation.

5.19.3. Constraints

The annotation-type element, which is available in the module and which has the same value for attribute name as the annotation element has, must be applicable to the model element type which contains the "annotation".

5.19.4. XSD

```
<xsd:complexType" name="annotationType">
   <xsd:sequence>
         <xsd:element
               name="e"
               type="kalua:annotation-propertyType"
               minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

5.19.5. Element Examples

```
<annotation>
   <type>objectStorage</type>
   <e name="cache">true</e>
   <e name="persist">true</e>
</annotation>
```

5.20. annotation-property

"annotation-property" element defines value of a property within an annotation. Semantics of the "annotation-property" values depend on the annotation-type element referred to by the annotation element containing the "annotation-property".

5.20.1. Element Attributes

Attribute	Type [Default]	Description	S
name	xsd:string	A name of the annotation property. The name must match with id attribute of one of the Annotation Entry Type elements within Annotation Type element to which the containing Annotation refers to.	M

Table 1: Annotation-property element attributes

5.20.2. Constraints

No two "annotation-property" elements within one annotation element may have the same value for attribute name.

For each "annotation-property" element, there must be an annotation-property-type element in the referred annotation-type element.

Content of the "annotation-property" element must match with the pattern of the corresponding annotation-property-type element.

All annotation properties which are defined for the annotation-type of the annotation must be present in the annotation, or must be defined as optional in the annotation- type.

5.20.3. XSD

```
<xsd:complexType name="annotation-propertyType">
   <xsd:simpleContent>
      <xsd:extension base="xsd:string">
         <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

5.20.4. Element Examples

```
<e name="cache">true</e>
```

5.21. annotation-type

A module can define "annotation types" for use in that module, or other modules that import the module. "annotation-types" define the structure of additional information that can be added to the Kalua model elements. Each "annotation-type" applies to a selected subset of model elements.

5.21.1. Element Attributes

Attribute	Type [Default]	Description	S
Name	xsd:string	Identifies the annotation type.	M
multiple	xsd:boolean [true]	If true, multiple instances of the annotation type can be attached to a model element. If false, only one instance with this annotation type can be attached to a model element.	O

5.21.2. Leaf Sub-Elements

Sub-Element	Type [Default]	Description	S
presentation	xsd:string	See Section 5.1.2	O
description	xsd:string	See Section 5.1.3	O

5.21.3. Sub-Elements

Sub-Element	MinOccurs	MaxOccurs	Description
annotable-type	1	unbounded	Defines to which Kalua element types annotations with this type can be added.
annotation	0	unbounded	See Section 5.19

```
| annotation-    |     0     | unbounded | Defines which properties |
| property       |           |           | must or may be present   |
|                |           |           | in annotations of this   |
|                |           |           | type.                    |
+----------------+-----------+-----------+--------------------------+
```

5.21.4. XSD

```
<xsd:complexType name="annotation-typeType">
   <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
         <xsd:element
               name="annotation-property-type"
               type="kalua:annotation-property-typeType"
               minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element
               name="annotable-type"
               type="kalua:annotable-typeType"
               maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
   <xsd:attribute name="multiple" type="xsd:boolean" default="true"/>
</xsd:complexType>
```

5.21.5. Element Examples

```
<annotation-type id="objectStorage">
   <annotable-type>class</annotable-type>
   <annotation-property-type name="cached">
         <pattern>true|false</pattern>
   </annotation-property>
   <annotation-property-type name="persist">
         <pattern>true|false</pattern>
   </annotation-property-type>
</annotationType>
```

5.22. annotable-type

"annotable-type" defines that an annotation, of the type, which contains this element, can be attached to the model elements of the given type. The content of this element is the name of the model element type.

5.22.1. XSD

```
<xsd:simpleType name="annotable-typeType">
   <xsd:restriction base="xsd:string">
         <xsd:enumeration value="module"/>
         <xsd:enumeration value="import"/>
         <xsd:enumeration value="attribute"/>
         <xsd:enumeration value="attribute-group"/>
         <xsd:enumeration value="structure"/>
         <xsd:enumeration value="sequence"/>
         <xsd:enumeration value="enum"/>
         <xsd:enumeration value="enum-literal"/>
         <xsd:enumeration value="typedef"/>
         <xsd:enumeration value="use"/>
         <xsd:enumeration value="key"/>
         <xsd:enumeration value="member"/>
         <xsd:enumeration value="constraint"/>
         <xsd:enumeration value="class"/>
         <xsd:enumeration value="relationship"/>
         <xsd:enumeration value="annotation-type"/>
         <xsd:enumeration value="annotation-property-type"/>
   </xsd:restriction>
</xsd:simpleType>
```

5.22.2. Element Examples

```
<annotable-type>class</annotable-type>
```

5.23. annotation-property-type

"annotation-property-type" defines a property within the annotation-type element. It defines the allowed values for the property and whether property is optional.

5.23.1. Leaf Sub-Elements

Sub-Element	Type [Default]	Description	S
presentation	xsd:string	See Section 5.1.2	O
description	xsd:string	See Section 5.1.3	O
optional	none	Defines whether a property is optional or mandatory in the annotation element. If the element is present, the property is optional. If the element is not present, the property is mandatory.	O
pattern	xsd:string [.*]	Language of the allowed values for the property. The language is defined using a regular expression, according to regular expression syntax and semantics specified in the 'XML Schema Part 2: Datatypes Second Edition' [XSD-TYPES].	O

5.23.2. Sub-Elements

Sub-Element	MinOccurs	MaxOccurs	Description
annotation	0	unbounded	See Section 5.19

5.23.3. XSD

```
<xsd:complexType name="annotation-property-typeType">
   <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
         <xsd:element name="optional" minOccurs="0"/>
         <xsd:element name="pattern" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
</xsd:complexType>
```

5.23.4. Element Examples

```
<annotation-property-type name="cached">
   <pattern>true|false</pattern>
</annotation-property-type>

<annotation-property-type name="persist">
   <optional/>
   <pattern>true|false</pattern>
</annotation-property-type>

<annotation-property-type name="organization">
</annotation-property-type>
```

6. IANA Considerations

A registry for standard Kalua modules needs to be set up. Each entry shall contain the unique module name, the unique XML namespace from the Kalua URI Scheme and some reference to the module's documentation.

The URIs for the Kalua XML namespace will be registered in the IETF XML registry RFC 3688 [RFC3688].

7. Security Considerations

Kalua DML itself has no security impact on the Internet. Security issues might be related to the usage of data, which is modeled with Kalua. These issues need to be discussed in documents describing the data models and related interfaces.

8. Acknowledgements

We would like to thank to David Kessens and Leo Hippelainen for their contributions and review.

9. References

9.1. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

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

[XSD-TYPES] Biron, P V. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition, W3C REC REC-xmlschema-2-20041028", October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html>.

9.2. Informative References

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

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

[Linowski] Linowski, B., "NETCONF Data Modeling Language Requirements", February 2008, <draft-linowski-netconf-dml-requirements-01>.

[RCDML] Presuhn, R., "Requirements for a Configuration Data Modeling Language", February 2008, <draft-presuhn-rcdml-03>.

[I-D.bjorklund-netconf-yang] Bjorklund, M., "YANG - A data modeling language for NETCONF", draft-bjorklund-netconf-yang-02 (work in progress), February 2008.

Appendix A. Kalua XML Schema


```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:kalua="urn:ietf:params:xml:ns:kalua:1"
targetNamespace="urn:ietf:params:xml:ns:kalua:1" version="0.2">
   <xsd:simpleType name="nameType">
      <xsd:restriction base="xsd:normalizedString">
      <xsd:pattern value="[a-zA-Z][_A-Za-z0-9]*"/>
      <!--xsd:maxLength value="30"/-->
   </xsd:restriction>
   </xsd:simpleType>
   <xsd:attributeGroup name="NamedElementAttributes">
      <xsd:attribute name="name" type="kalua:nameType" use="required"/>
   </xsd:attributeGroup>
   <xsd:group name="ModelElementProperties">
      <xsd:sequence>
         <xsd:element name="description"
           type="xsd:string" minOccurs="0"/>
         <xsd:element name="annotation"
           type="kalua:annotationType" minOccurs="0"
           maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:group>
   <xsd:group name="NamedElementOnlyProperties">
      <xsd:sequence>
         <xsd:element name="presentation" minOccurs="0">
            <xsd:simpleType>
               <xsd:restriction base="xsd:normalizedString">
                  <xsd:maxLength value="100"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
      </xsd:sequence>
   </xsd:group>
   <xsd:group name="NamedElementProperties">
      <xsd:sequence>
         <xsd:group ref="kalua:NamedElementOnlyProperties"/>
         <xsd:group ref="kalua:ModelElementProperties"/>
      </xsd:sequence>
   </xsd:group>
   <xsd:group name="ConstrainableElementProperties">
      <xsd:sequence>
         <xsd:element name="constraint" minOccurs="0"
          maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
```

```
                  <xsd:group ref="kalua:ModelElementProperties"/>
                  <xsd:element name="expression"
                   type="xsd:string"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:group>
   <xsd:group name="AttributeContainer">
      <xsd:sequence>
         <xsd:element name="use"
          minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:group ref="kalua:ModelElementProperties"/>
               </xsd:sequence>
               <xsd:attribute name="attribute-group"
                type="kalua:ModelElementReference"/>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="attribute" type="kalua:Attribute"
          minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="key" minOccurs="0"
          maxOccurs="unbounded">
            <xsd:annotation>
               <xsd:documentation>
   Keys on classes can be globally unique or locally
   unique (i.e unique within the scope of their containing
   element).
   Keys on structures are always locally unique.
   Keys on attribute-groups are merged to the set of keys
   of the element using them. If a class uses the
   attribute-group, the key can be globally unique.
               </xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:group ref="kalua:ModelElementProperties"/>
                  <xsd:element name="member" type="xsd:string"
                   maxOccurs="unbounded"/>
               </xsd:sequence>
               <xsd:attribute name="scope" default="local">
                  <xsd:simpleType>
                     <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="local"/>
                        <xsd:enumeration value="global"/>
                     </xsd:restriction>
                  </xsd:simpleType>
```

```
               </xsd:attribute>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:group>
   <xsd:group name="ModuleIdentityProperties">
      <xsd:sequence>
         <xsd:element name="ns-uri" type="xsd:string"/>
         <xsd:element name="ns-prefix" type="xsd:string"/>
         <xsd:element name="release" minOccurs="0">
            <xsd:simpleType>
               <xsd:restriction base="xsd:string">
                  <xsd:maxLength value="20"/>
                  <xsd:pattern
                   value="[a-zA-Z0-9]+(\.[a-zA-Z0-9]+)*"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
      </xsd:sequence>
   </xsd:group>
   <xsd:element name="module">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:group ref="kalua:NamedElementProperties"/>
            <xsd:group ref="kalua:ModuleIdentityProperties"/>
            <xsd:element name="organization">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                     <xsd:maxLength value="100"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:element>
            <xsd:element name="import"
             type="kalua:importType" minOccurs="0"
             maxOccurs="unbounded"/>
            <xsd:sequence>
               <xsd:choice minOccurs="0"
                maxOccurs="unbounded">
                  <xsd:element name="typedef"
                   type="kalua:typedefType"
                   minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="attribute-group"
                   type="kalua:AttributeGroup"
                   minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="class"
                   type="kalua:Class"
                   minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="relationship"
```

```
                    type="kalua:Relationship"
                    minOccurs="0" maxOccurs="unbounded"/>
                   <xsd:element name="annotation-type"
                    type="kalua:annotation-typeType"
                    minOccurs="0" maxOccurs="unbounded"/>
                   <!--xsd:element name="augment"
                    type="kalua:augmentType"
                    minOccurs="0" maxOccurs="unbounded"/-->
                </xsd:choice>
             </xsd:sequence>
          </xsd:sequence>
          <xsd:attributeGroup
           ref="kalua:NamedElementAttributes"/>
       </xsd:complexType>
    </xsd:element>
    <xsd:group name="Datatype">
       <xsd:choice>
          <xsd:group ref="kalua:simpleTypeReference"/>
          <xsd:element name="simple-type">
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:group
                    ref="kalua:ModelElementProperties"/>
                   <xsd:group
                    ref="kalua:simpleTypeDefinition"/>
                   <xsd:element
                    name="unit" type="xsd:string"
                    minOccurs="0"/>
                   <xsd:group
                    ref="kalua:ConstrainableElementProperties"/>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="structure">
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:group ref="kalua:ModelElementProperties"/>
                   <xsd:group ref="kalua:AttributeContainer"/>
                   <xsd:group
                    ref="kalua:ConstrainableElementProperties"/>
                </xsd:sequence>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="sequence">
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:group ref="kalua:ModelElementProperties"/>
                   <xsd:group ref="kalua:Datatype"/>
```

```
                   <xsd:group ref="kalua:Accessibility"/>
                   <xsd:group
                    ref="kalua:ConstrainableElementProperties"/>
                </xsd:sequence>
                <xsd:attribute name="minLength"
                 type="xsd:nonNegativeInteger"
                 default="0"/>
                <xsd:attribute name="maxLength"
                 default="unbounded">
                   <xsd:simpleType>
                      <xsd:union
                       memberTypes="xsd:nonNegativeInteger">
                         <xsd:simpleType>
                            <xsd:restriction
                             base="xsd:string">
                               <xsd:enumeration
                                value="unbounded"/>
                            </xsd:restriction>
                         </xsd:simpleType>
                      </xsd:union>
                   </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="ordered"
                 type="xsd:boolean" default="true"/>
                <xsd:attribute name="elementName"
                 type="xsd:NCName"/>
             </xsd:complexType>
          </xsd:element>
       </xsd:choice>
    </xsd:group>
    <xsd:simpleType name="ModelElementReference">
       <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType name="AttributeGroup">
       <xsd:sequence>
          <xsd:group ref="kalua:NamedElementProperties"/>
          <xsd:group ref="kalua:AttributeContainer"/>
          <xsd:group ref="kalua:ConstrainableElementProperties"/>
       </xsd:sequence>
       <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
    </xsd:complexType>
    <xsd:group name="Accessibility">
       <xsd:sequence>
          <xsd:element name="max-access"
           default="read-create" minOccurs="0">
             <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                   <xsd:enumeration value="not-accessible"/>
```

```
                   <xsd:enumeration value="accessible-for-notify"/>
                   <xsd:enumeration value="read-only"/>
                   <xsd:enumeration value="read-write"/>
                   <xsd:enumeration value="read-create"/>
                </xsd:restriction>
             </xsd:simpleType>
          </xsd:element>
       </xsd:sequence>
    </xsd:group>
    <xsd:complexType name="Class">
       <xsd:sequence>
          <xsd:group ref="kalua:NamedElementProperties"/>
          <xsd:group ref="kalua:Accessibility"/>
          <xsd:element name="abstract" minOccurs="0">
             <xsd:annotation>
                <xsd:documentation>
    If present it means that this class cannot be
    instantiated.</xsd:documentation>
             </xsd:annotation>
             <xsd:complexType/>
          </xsd:element>
          <xsd:element name="super-class"
           type="kalua:ModelElementReference"
           minOccurs="0">
             <xsd:annotation>
                <xsd:documentation>
    Do we really want a restriction to single
    inheritance?</xsd:documentation>
             </xsd:annotation>
          </xsd:element>
          <xsd:group ref="kalua:AttributeContainer"/>
          <xsd:group ref="kalua:ConstrainableElementProperties"/>
       </xsd:sequence>
       <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
    </xsd:complexType>
    <xsd:complexType name="Attribute">
       <xsd:sequence>
          <xsd:group ref="kalua:NamedElementProperties"/>
          <xsd:choice minOccurs="0">
             <xsd:element name="mandatory">
                <xsd:complexType/>
             </xsd:element>
             <xsd:element name="optional">
                <xsd:complexType/>
             </xsd:element>
          </xsd:choice>
          <xsd:choice minOccurs="0">
             <xsd:element name="read-only">
```

```
              <xsd:complexType/>
           </xsd:element>
           <xsd:element name="unchangeable">
              <xsd:complexType/>
           </xsd:element>
           <xsd:element name="read-write">
              <xsd:complexType/>
           </xsd:element>
        </xsd:choice>
        <xsd:group ref="kalua:Datatype"/>
        <!-- some semantic changes w.r.t. OCoS here ...-->
        <!--xsd:attribute name="initialization"
         type="cmb:InitializationKind"/-->
        <!--xsd:attribute name="unsettable"
         type="xsd:boolean"/-->
        <xsd:element name="defaultValueLiteral"
         type="xsd:string" minOccurs="0"/>
        <xsd:element name="unit"
         type="xsd:string" minOccurs="0"/>
        <xsd:group ref="kalua:ConstrainableElementProperties"/>
     </xsd:sequence>
     <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
  </xsd:complexType>
  <xsd:complexType name="Relationship">
     <xsd:sequence>
        <xsd:group ref="kalua:NamedElementProperties"/>
        <xsd:element name="baseRelationship"
         type="kalua:ModelElementReference"
         minOccurs="0"/>
        <xsd:choice minOccurs="0">
           <xsd:element name="read-only">
              <xsd:complexType/>
           </xsd:element>
           <xsd:element name="read-write">
              <xsd:complexType/>
           </xsd:element>
        </xsd:choice>
        <xsd:element name="kind">
           <xsd:complexType>
              <xsd:choice>
                 <xsd:element name="containment"/>
                 <xsd:element name="reference"/>
                 <xsd:element name="calculated">
                    <xsd:complexType>
                       <xsd:sequence>
                          <xsd:element name="condition">
                             <xsd:complexType>
                                <xsd:simpleContent>
```

```
                                     <xsd:extension
                                      base="xsd:string">
                                        <xsd:attribute
                                         name="language"
                                 type="xsd:normalizedString"/>
                                     </xsd:extension>
                                  </xsd:simpleContent>
                               </xsd:complexType>
                            </xsd:element>
                         </xsd:sequence>
                      </xsd:complexType>
                   </xsd:element>
                </xsd:choice>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="source" type="kalua:RelationshipEnd"/>
          <xsd:element name="target" type="kalua:RelationshipEnd"/>
          <xsd:group ref="kalua:ConstrainableElementProperties"/>
       </xsd:sequence>
       <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
    </xsd:complexType>
    <xsd:complexType name="RelationshipEnd">
       <xsd:sequence>
          <xsd:element name="class"
           type="kalua:ModelElementReference"/>
          <xsd:element name="role" type="kalua:nameType"/>
          <xsd:element name="minCardinality"
           type="xsd:nonNegativeInteger"
           default="0" minOccurs="0"/>
          <xsd:element name="maxCardinality"
           default="unbounded" minOccurs="0">
             <xsd:simpleType>
                <xsd:union memberTypes="xsd:nonNegativeInteger">
                   <xsd:simpleType>
                      <xsd:restriction base="xsd:string">
                         <xsd:enumeration value="unbounded"/>
                      </xsd:restriction>
                   </xsd:simpleType>
                </xsd:union>
             </xsd:simpleType>
          </xsd:element>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="annotation-typeType">
       <xsd:sequence>
          <xsd:group ref="kalua:NamedElementProperties"/>
          <xsd:element name="annotation-property-type"
           type="kalua:annotation-property-typeType"
```

```
          minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="annotable-type"
          type="kalua:annotable-typeType"
          maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup
       ref="kalua:NamedElementAttributes"/>
      <xsd:attribute name="multiple"
       type="xsd:boolean" default="true"/>
   </xsd:complexType>
   <xsd:complexType name="importType">
      <xsd:sequence>
         <xsd:group ref="kalua:ModuleIdentityProperties"/>
         <xsd:group ref="kalua:ModelElementProperties"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="annotation-propertyType">
      <xsd:simpleContent>
         <xsd:extension base="xsd:string">
            <xsd:attributeGroup
             ref="kalua:NamedElementAttributes"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="annotationType">
      <xsd:sequence>
         <xsd:element name="e"
          type="kalua:annotation-propertyType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string"/>
   </xsd:complexType>
   <xsd:complexType name="annotation-property-typeType">
      <xsd:sequence>
         <xsd:group ref="kalua:NamedElementProperties"/>
         <xsd:element name="optional" minOccurs="0"/>
         <xsd:element name="pattern" type="xsd:string"
          minOccurs="0"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
   </xsd:complexType>
   <!--xsd:complexType name="augmentType">
      <xsd:sequence>
         <xsd:group ref="kalua:ModelElementProperties"/>
         <xsd:element name="target"
          type="kalua:ModelElementReference"/>
         <xsd:element name="when" type="xsd:string"
          minOccurs="0"/>
```

```
          <xsd:group ref="kalua:AttributeContainer"/>
       </xsd:sequence>
    </xsd:complexType-->
    <xsd:complexType name="typedefType">
       <xsd:sequence>
          <xsd:group ref="kalua:NamedElementProperties"/>
          <xsd:group ref="kalua:Datatype"/>
       </xsd:sequence>
       <xsd:attributeGroup
        ref="kalua:NamedElementAttributes"/>
    </xsd:complexType>
    <xsd:simpleType name="annotable-typeType">
       <xsd:restriction base="xsd:string">
          <xsd:enumeration value="module"/>
          <xsd:enumeration value="import"/>
          <xsd:enumeration value="attribute"/>
          <xsd:enumeration value="attribute-group"/>
          <xsd:enumeration value="structure"/>
          <xsd:enumeration value="sequence"/>
          <xsd:enumeration value="enum"/>
          <xsd:enumeration value="enum-literal"/>
          <xsd:enumeration value="typedef"/>
          <xsd:enumeration value="use"/>
          <xsd:enumeration value="key"/>
          <xsd:enumeration value="member"/>
          <xsd:enumeration value="constraint"/>
          <xsd:enumeration value="class"/>
          <xsd:enumeration value="relationship"/>
          <xsd:enumeration value="annotation-type"/>
          <xsd:enumeration value="annotation-property-type"/>
       </xsd:restriction>
    </xsd:simpleType>
    <!-- simple type definitions -->
    <xsd:group name="facets">
       <xsd:choice>
          <xsd:element name="minExclusive"
           type="kalua:facet" id="minExclusive"/>
          <xsd:element name="minInclusive"
           type="kalua:facet" id="minInclusive"/>
          <xsd:element name="maxExclusive"
           type="kalua:facet" id="maxExclusive"/>
          <xsd:element name="maxInclusive"
           type="kalua:facet" id="maxInclusive"/>
          <xsd:element name="totalDigits"
           id="totalDigits">
             <xsd:complexType>
                <xsd:complexContent>
                   <xsd:restriction base="kalua:numFacet">
```

```
                      <xsd:attribute name="value"
                       type="xs:positiveInteger"
                       use="required"/>
                   </xsd:restriction>
                </xsd:complexContent>
             </xsd:complexType>
          </xsd:element>
          <xsd:element name="fractionDigits"
           type="kalua:numFacet" id="fractionDigits"/>
          <xsd:element name="length"
           type="kalua:numFacet" id="length"/>
          <xsd:element name="minLength"
           type="kalua:numFacet" id="minLength"/>
          <xsd:element name="maxLength"
           type="kalua:numFacet" id="maxLength"/>
          <xsd:element name="pattern"
           type="kalua:facet" id="pattern"/>
       </xsd:choice>
    </xsd:group>
    <xsd:group name="simpleTypeReference">
       <xsd:sequence>
          <xsd:element name="type"
           type="kalua:ModelElementReference"/>
       </xsd:sequence>
    </xsd:group>
    <xsd:group name="simpleTypeDefinition">
       <xsd:sequence>
          <xsd:choice>
             <xsd:element name="restriction"
              type="kalua:restrictionType"/>
             <xsd:element name="union"
              type="kalua:unionType"/>
             <xsd:element name="enum"
              type="kalua:enumType"/>
             <xsd:group ref="kalua:simpleTypeReference"/>
          </xsd:choice>
       </xsd:sequence>
    </xsd:group>
    <xsd:complexType name="restrictionType">
       <xsd:sequence>
          <xsd:group ref="kalua:simpleTypeDefinition"/>
          <xsd:group ref="kalua:facets" minOccurs="0"
           maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="unionType">
       <xsd:sequence>
          <xsd:group ref="kalua:simpleTypeDefinition"
```

```
           maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="enumType">
       <xsd:sequence>
          <xsd:element name="enum-literal"
           type="kalua:enum-literalType"
           maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="base"
        type="xsd:QName" use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="enum-literalType">
       <xsd:sequence>
          <xsd:group ref="kalua:NamedElementProperties"/>
       </xsd:sequence>
       <xsd:attribute name="value"/>
       <xsd:attributeGroup ref="kalua:NamedElementAttributes"/>
    </xsd:complexType>
    <xsd:complexType name="facet">
       <xsd:sequence>
          <xsd:group ref="kalua:ModelElementProperties"/>
       </xsd:sequence>
       <xsd:attribute name="value" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="numFacet">
       <xsd:complexContent>
          <xsd:restriction base="kalua:facet">
             <xsd:attribute name="value"
              type="xs:nonNegativeInteger" use="required"/>
          </xsd:restriction>
       </xsd:complexContent>
    </xsd:complexType>
 </xsd:schema>
```

Appendix B. Module Example: RFC1213-MIB

The example below shows how the contents of the MIB RFC1213-MIB is represented in Kalua.

```
<?xml version="1.0" encoding="UTF-8"?>
<kalua:module name="rfc1213_mib"
xmlns:kalua="urn:ietf:params:xml:ns:kalua:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:kalua:1
C:\Users\kalua\kalua.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- name must not contain '-': rfc1213-mib -->
<presentation>RFC1213-MIB</presentation>
<description>Extracted from rfc1213.txt</description>
<ns-uri>iso.org.dod.internet.mgmt.mib-2.rfc1213</ns-uri>
<!-- MIBs do not map to OID namespaces: MIBs contain overlapping
definitions, and define multiple namespaces  -->
<ns-prefix>rfc1213</ns-prefix>
<release>1</release>
<organization>RFC</organization>
<import>
   <ns-uri>iso.org.dod.internet.mgmt.mib-2.rfc1155-smi</ns-uri>
   <ns-prefix>rfc1155-smi</ns-prefix>
   <release>1</release>
   <description>Structure of Management Information
   </description>
</import>
<import>
   <ns-uri>iso.org.dod.internet.mgmt.mib-2.rfc1212</ns-uri>
   <ns-prefix>rfc1212</ns-prefix>
   <release>1</release>
   <description>Object type definition macros</description>
</import>
<typedef name="DisplayString">
   <description>This data type is used to model textual
   information taken from the NVT ASCII character set.
   By convention, objects with this syntax are declared
   as having SIZE (0..255)</description>
   <type>kalua:string</type>
</typedef>
<typedef name="PhysAddr">
   <description>This data type is used to model media
   addresses.  For many types of media, this will be in
   a binary representation. For example, an ethernet
   address would be represented as a string of 6 octets.
   </description>
   <type>kalua:string</type>
```

```
   </typedef>
   <class name="system">
      <presentation>System group</presentation>
      <description>
   Implementation of the System group is mandatory
   for all systems.  If an agent is not configured to have a
   value for any of these variables, a string of length 0 is
   returned.</description>
      <attribute name="sysDescr">
      <description>
          A textual description of the entity.
          This value should include the full name and version
          identification of the system's hardware type,
          software operating-system, and networking
          software.  It is mandatory that this only contain
          printable ASCII characters.</description>
         <read-only/>
         <simple-type>
            <restriction>
               <type>DisplayString</type>
   <!-- yet unsolved is how to reference a typedef
   properly. In theory, the name of the typedef could
   collide with the name of an attribute or class.

   Potential solution: allow all named elements to
   have explicit namespaces, which apply recursively
   to all contained named elements? This may also solve
   the issue of MIB - namespace mismatch ...-->
                  <minLength value="0"/>
                  <maxLength value="255"/>
               </restriction>
            </simple-type>
         </attribute>
         <attribute name="sysObjectID">
            <description>
            The vendor's authoritative identification
            of the network management subsystem contained in the
            entity.  This value is allocated within the SMI
            enterprises subtree (1.3.6.1.4.1) and provides an
            easy and unambiguous means for determining `what
            kind of box' is being managed.  For example, if
            vendor `Flintstones, Inc.' was assigned the
            subtree 1.3.6.1.4.1.4242, it could assign the
            identifier 1.3.6.1.4.1.4242.1.1 to its `Fred
            Router'.
            </description>
            <read-only/>
            <type>kalua:string</type>
```

```
   <!-- this could as well be some kind of namespace type ... -->
         </attribute>
         <attribute name="sysUpTime">
            <description>
            The time (in hundredths of a second) since the
            network management portion of the system was last
            re-initialized.
            </description>
            <read-only/>
            <type>rfc1155-smi:TimeTicks</type>
         </attribute>
         <attribute name="sysContact">
            <description>
            The textual identification of the contact person
            for this managed node, together with information
            on how to contact this person.
            </description>
            <simple-type>
               <restriction>
                  <type>DisplayString</type>
                  <minLength value="0"/>
                  <maxLength value="255"/>
               </restriction>
            </simple-type>
         </attribute>
         <attribute name="sysName">
            <description>
            An administratively-assigned name for this
            managed node.  By convention, this is the node's
            fully-qualified domain name.
            </description>
            <simple-type>
               <restriction>
                  <type>DisplayString</type>
                  <minLength value="0"/>
                  <maxLength value="255"/>
               </restriction>
            </simple-type>
         </attribute>
         <attribute name="sysLocation">
            <description>
            The physical location of this node (e.g.,
            'telephone closet, 3rd floor').
            </description>
            <simple-type>
               <restriction>
                  <type>DisplayString</type>
                  <minLength value="0"/>
```

```
                  <maxLength value="255"/>
               </restriction>
            </simple-type>
         </attribute>
         <attribute name="sysServices">
            <description>
            A value which indicates the set of services that
            this entity primarily offers.

            The value is a sum.  This sum initially takes the
            value zero, Then, for each layer, L, in the range
            1 through 7, that this node performs transactions
            for, 2 raised to (L - 1) is added to the sum.  For
            example, a node which performs primarily routing
            functions would have a value of 4 (2^(3-1)).  In
            contrast, a node which is a host offering
            application services would have a value of 72
            (2^(4-1) + 2^(7-1)).  Note that in the context of
            the Internet suite of protocols, values should be
            calculated accordingly:

            layer 1  physical (e.g., repeaters)
            layer 2  datalink/subnetwork (e.g., bridges)
            layer 3  internet (e.g., IP gateways)
            layer 4  end-to-end  (e.g., IP hosts)
            layer 7  applications (e.g., mail relays)

            For systems including OSI protocols, layers 5 and
            6 may also be counted."
            </description>
            <read-only/>
            <simple-type>
               <restriction>
                  <type>kalua:integer</type>
                  <maxInclusive value="127"/>
               </restriction>
            </simple-type>
         </attribute>
          <key scope="global">
            <member>sysName</member>
          </key>
      </class>
      <class name="interfaces">
         <attribute name="ifNumber">
            <description>
            The number of network interfaces
            (regardless of their current state) present on
            this system.
```

```
            </description>
            <read-only/>
            <type>kalua:integer</type>
         </attribute>
      </class>
      <relationship name="ifTable">
         <description>
            A list of interface entries.  The number
            of entries is given by the value of ifNumber.
         </description>
         <read-only/>
         <kind>
            <containment/>
         </kind>
         <source>
            <class>interfaces</class>
            <role>parent</role>
            <minCardinality>1</minCardinality>
            <maxCardinality>1</maxCardinality>
         </source>
         <target>
            <class>ifEntry</class>
            <role>children</role>
            <minCardinality>0</minCardinality>
            <maxCardinality>unbounded</maxCardinality>
         </target>
      </relationship>
      <class name="ifEntry">
         <description>
            An interface entry containing objects at
            the subnetwork layer and below for a particular
            interface.
         </description>
         <attribute name="ifIndex">
            <description>
            A unique value for each interface.  Its value
            ranges between 1 and the value of ifNumber.  The
            value for each interface must remain constant at
            least from one re-initialization of the entity's
            network management system to the next re-
            initialization.
            </description>
            <read-only/>
            <type>kalua:integer</type>
         </attribute>
         <attribute name="ifDescr">
            <description>
            A textual string containing information about the
```

```
            interface.  This string should include the name of
            the manufacturer, the product name and the version
            of the hardware interface.
            </description>
            <read-only/>
            <simple-type>
               <restriction>
                  <type>DisplayString</type>
                  <minLength value="0"/>
                  <maxLength value="255"/>
               </restriction>
            </simple-type>
         </attribute>
         <attribute name="ifType">
            <description>
            The type of interface, distinguished according to
            the physical/link protocol(s) immediately 'below'
            the network layer in the protocol stack.
            </description>
            <read-only/>
            <simple-type>
         <enum>
            <enum-literal value="1" name="other"/>
            <enum-literal value="2" name="regular1822"/>
            <enum-literal value="3" name="hdh1822"/>
            <enum-literal value="4" name="ddn_x25">
               <presentation>ddn-x25</presentation>
            </enum-literal>
            <enum-literal value="5" name="rfc877_x25">
               <presentation>rfc877-x25
               </presentation>
            </enum-literal>
            <enum-literal value="6"
            name="ethernet_csmacd">
               <presentation>ethernet-csmacd
               </presentation>
            </enum-literal>
            <enum-literal value="7"
            name="iso88023_csmacd">
               <presentation>iso88023-csmacd
               </presentation>
            </enum-literal>
            <enum-literal value="8"
            name="iso88024_tokenBus">
               <presentation>iso88024-tokenBus
               </presentation>
            </enum-literal>
            <enum-literal value="9"
```

```
            name="iso88025_tokenRing">
               <presentation>iso88025-tokenRing
               </presentation>
            </enum-literal>
            <enum-literal value="10"
            name="iso88026_man">
               <presentation>iso88026-man
               </presentation>
            </enum-literal>
            <!-- .... -->
            <enum-literal value="32"
            name="frame_relay">
               <presentation>frame-relay
               </presentation>
            </enum-literal>
         </enum>
            </simple-type>
         </attribute>
         <attribute name="ifMtu">
            <description>
            The size of the largest datagram which can be
            sent/received on the interface, specified in
            octets.  For interfaces that are used for
            transmitting network datagrams, this is the size
            of the largest network datagram that can be sent
            on the interface.
            </description>
            <read-only/>
            <type>kalua:integer</type>
            <unit>octets</unit>
         </attribute>
         <attribute name="ifSpeed">
            <description>
            An estimate of the interface's current bandwidth
            in bits per second.  For interfaces which do not
            vary in bandwidth or for those where no accurate
            estimation can be made, this object should contain
            the nominal bandwidth.
            </description>
            <read-only/>
            <type>rfc1155-smi:Gauge</type>
            <unit>bit/sec</unit>
         </attribute>
         <attribute name="ifPhysAddress">
            <description>
            The interface's address at the protocol layer
            immediately `below' the network layer in the
            protocol stack.  For interfaces which do not have
```

```
            such an address (e.g., a serial line), this object
            should contain an octet string of zero length.
            </description>
            <read-only/>
            <type>PhysAddress</type>
         </attribute>
         <attribute name="ifAdminStatus">
            <description>
            The desired state of the interface.  The
            testing(3) state indicates that no operational
            packets can be passed.</description>
            <simple-type>
               <enum base="kalua:int">
                  <enum-literal value="1"
                  name="up"/>
                  <enum-literal value="2"
                  name="down"/>
                  <enum-literal value="3"
                  name="testing"/>
               </enum>
            </simple-type>
         </attribute>
         <attribute name="ifOperStatus">
            <description>
            The current operational state of the interface.
            The testing(3) state indicates that no operational
            packets can be passed.</description>
            <read-only/>
            <simple-type>
               <enum>
                  <enum-literal value="1"
                  name="up"/>
                  <enum-literal value="2"
                  name="down"/>
                  <enum-literal value="3"
                  name="testing"/>
               </enum>
            </simple-type>
         </attribute>
         <attribute name="ifSpecific">
   <!-- this is not easily translated!
   Seems to be a workaround for true inheritance support.
   Then this attribute would be the discriminator. -->
            <description>
            A reference to MIB definitions specific to the
            particular media being used to realize the
            interface.  For example, if the interface is
            realized by an ethernet, then the value of this
```

```
            object refers to a document defining objects
            specific to ethernet.  If this information is not
            present, its value should be set to the OBJECT
            IDENTIFIER { 0 0 }, which is a syntatically valid
            object identifier, and any conformant
            implementation of ASN.1 and BER must be able to
            generate and recognize this value.
            </description>
            <read-only/>
            <type>kalua:string</type>
         </attribute>
         <key scope="local">
            <member>ifIndex</member>
         </key>
      </class>
      <class name="at">
         <description>
            the Address Translation group
            Implementation of the Address Translation group is
            mandatory for all systems.  Note however that this group
            is deprecated by MIB-II. That is, it is being included
            solely for compatibility with MIB-I nodes, and will most
            likely be excluded from MIB-III nodes.  From MIB-II and
            onwards, each network protocol group contains its own
            address translation tables.

            The Address Translation group contains one table which is
            the union across all interfaces of the translation tables
            for converting a NetworkAddress (e.g., an IP address) into
            a subnetwork-specific address.  For lack of a better term,
            this document refers to such a subnetwork-specific address
            as a `physical' address.

            Examples of such translation tables are: for broadcast
            media where ARP is in use, the translation table is
            equivalent to the ARP cache; or, on an X.25 network where
            non-algorithmic translation to X.121 addresses is
            required, the translation table contains the
            NetworkAddress to X.121 address equivalences.
         </description>
      </class>
      <relationship name="atTable">
         <description>
            The Address Translation tables contain the
            NetworkAddress to `physical' address equivalences.
            Some interfaces do not use translation tables for
            determining address equivalences (e.g., DDN-X.25
            has an algorithmic method); if all interfaces are
```

```
            of this type, then the Address Translation table
            is empty, i.e., has zero entries.
         </description>
         <kind>
            <containment/>
         </kind>
         <source>
            <class>at</class>
            <role>parent</role>
            <minCardinality>1</minCardinality>
            <maxCardinality>1</maxCardinality>
         </source>
         <target>
            <class>atEntry</class>
            <role>children</role>
            <minCardinality>0</minCardinality>
            <maxCardinality>unbounded</maxCardinality>
         </target>
      </relationship>
      <class name="atEntry">
         <description>
   Each entry contains one NetworkAddress to
   'physical' address equivalence.
         </description>
         <annotation name="statusAnnotation">
            <e name="deprecated"/>
         </annotation>
         <attribute name="atIfIndex">
            <description>
            The interface on which this entry's equivalence
            is effective.  The interface identified by a
            particular value of this index is the same
            interface as identified by the same value of
            ifIndex.
            </description>
            <annotation name="statusAnnotation">
               <e name="deprecated"/>
            </annotation>
            <type>kalua:int</type>
         </attribute>
         <attribute name="atPhysAddress">
            <description>
            The media-dependent `physical' address.
            Setting this object to a null string (one of zero
            length) has the effect of invaliding the
            corresponding entry in the atTable object.  That
            is, it effectively dissasociates the interface
            identified with said entry from the mapping
```

```
            identified with said entry.  It is an
            implementation-specific matter as to whether the
            agent removes an invalidated entry from the table.
            Accordingly, management stations must be prepared
            to receive tabular information from agents that
            corresponds to entries not currently in use.
            Proper interpretation of such entries requires
            examination of the relevant atPhysAddress object.
            </description>
            <annotation name="statusAnnotation">
               <e name="deprecated"/>
            </annotation>
            <type>PhysAddress</type>
         </attribute>
         <attribute name="atNetAddress">
            <description>
            The NetworkAddress (e.g., the IP address)
            corresponding to the media-dependent `physical'
            address.
            </description>
            <annotation name="statusAnnotation">
               <e name="deprecated"/>
            </annotation>
            <type>NetworkAddress</type>
         </attribute>
         <key scope="local">
            <member>atIfIndex</member>
            <member>atNetAddress</member>
         </key>
      </class>
      <relationship name="addressTranslation">
        <kind>
          <calculated>
           <condition>$source/atIfIndex=
                      $target/ifIndex
           </condition>
          </calculated>
         </kind>
         <source>
            <class>atEntry</class>
            <role>atEntry</role>
            <minCardinality>0</minCardinality>
            <maxCardinality>1</maxCardinality>
         </source>
         <target>
            <class>ifEntry</class>
            <role>ifEntry</role>
            <minCardinality>1</minCardinality>
```

```
            <maxCardinality>1</maxCardinality>
         </target>
      </relationship>
      <class name="ip">
         <attribute name="ipForwarding">
            <description>
            The indication of whether this entity is acting
            as an IP gateway in respect to the forwarding of
            datagrams received by, but not addressed to, this
            entity.  IP gateways forward datagrams.  IP hosts
            do not (except those source-routed via the host).

            Note that for some managed nodes, this object may
            take on only a subset of the values possible.
            Accordingly, it is appropriate for an agent to
            return a 'badValue' response if a management
            station attempts to change this object to an
            inappropriate value.
            </description>
         <simple-type>
            <enum base="kalua:integer">
               <enum-literal value="1"
               name="forwarding">
               <description>acting as a gateway
               </description>
               </enum-literal>
               <enum-literal value="2"
               name="not_forwarding">
                  <presentation>not-forwarding
                  </presentation>
                  <description>
                  NOT acting as a gateway
                  </description>
               </enum-literal>
               </enum>
            </simple-type>
         </attribute>
      </class>
      <relationship name="ipAddrTable">
         <description>
   The table of addressing information relevant to this
         entity's IP addresses.</description>
         <kind>
            <containment/>
         </kind>
         <source>
            <class>ip</class>
            <role>parent</role>
```

```
            <minCardinality>1</minCardinality>
            <maxCardinality>1</maxCardinality>
         </source>
         <target>
            <class>ipAddrEntry</class>
            <role>children</role>
            <minCardinality>0</minCardinality>
            <maxCardinality>unbounded</maxCardinality>
         </target>
      </relationship>
      <class name="ipAddrEntry">
         <attribute name="ipAdEntAddr">
            <description>
   The IP address to which this entry's
   addressing information pertains.
            </description>
            <read-only/>
            <type>rfc1155-smi:IpAddress</type>
         </attribute>
         <attribute name="ipAdEntIfIndex">
            <description>
            The index value which uniquely identifies the
            interface to which this entry is applicable.  The
            interface identified by a particular value of this
            index is the same interface as identified by the
            same value of ifIndex.
            </description>
            <read-only/>
            <type>kalua:integer</type>
         </attribute>
         <key scope="global">
            <member>ipAdEntAddr</member>
         </key>
      </class>
      <relationship name="ip_interface">
       <kind>
        <calculated>
          <condition>$source/ipAdEntIfIndex=
                     $target/ifIndex
          </condition>
        </calculated>
       </kind>
         <source>
            <class>ipAddrEntry</class>
            <role>ipAddress</role>
            <minCardinality>0</minCardinality>
            <maxCardinality>1</maxCardinality>
         </source>
```

```
         <target>
            <class>ifEntry</class>
            <role>interface</role>
            <minCardinality>1</minCardinality>
            <maxCardinality>1</maxCardinality>
         </target>
      </relationship>
      <class name="icmp"/>
      <class name="tcp"/>
      <class name="udp"/>
      <class name="egp"/>
      <class name="transmission"/>
      <class name="snmp"/>
      <annotation-type name="statusAnnotation">
         <annotation-property-type name="current"/>
         <annotation-property-type name="deprecated"/>
         <annotation-property-type name="obsolete"/>
         <annotable-type>annotation-property-type
         </annotable-type>
         <annotable-type>annotation-type</annotable-type>
         <annotable-type>attribute</annotable-type>
         <annotable-type>attribute-group</annotable-type>
         <annotable-type>class</annotable-type>
         <annotable-type>constraint</annotable-type>
         <annotable-type>enum</annotable-type>
         <annotable-type>enum-literal</annotable-type>
         <annotable-type>import</annotable-type>
         <annotable-type>key</annotable-type>
         <annotable-type>member</annotable-type>
         <annotable-type>module</annotable-type>
         <annotable-type>relationship</annotable-type>
         <annotable-type>sequence</annotable-type>
         <annotable-type>structure</annotable-type>
         <annotable-type>typedef</annotable-type>
         <annotable-type>use</annotable-type>
      </annotation-type>

      <class name="linkDown">
         <description>
   A linkDown trap signifies that the SNMP entity, acting in
   an agent role, has detected that the ifOperStatus object for
   one of its communication links is about to enter the down
   state from some other state (but not from the notPresent
   state).  This other state is indicated by the included value
   of ifOperStatus.
         </description>
         <max-access>accessible-for-notify</max-access>
         <attribute name="ifIndex">
```

```
            <type>kalua:int</type>
         </attribute>
         <attribute name="ifAdminStatus">
            <structure>
               <attribute name="ifIndex">
                  <type>kalua:int</type>
               </attribute>
               <attribute name="ifAdminStatus">
                  <description>
   The desired state of the interface.  The testing(3) state
   indicates that no operational packets can be passed.  When a
   managed system initializes, all interfaces start with
   ifAdminStatus in the down(2) state.  As a result of either
   explicit management action or per configuration information
   retained by the managed system, ifAdminStatus is then
   changed to either the up(1) or testing(3) states (or remains
   in the down(2) state).</description>
               <simple-type>
                  <enum>
                     <enum-literal
                     name="up" value="1"/>
                     <enum-literal
                     name="down" value="2"/>
                     <enum-literal
                     name="testing" value="3"/>
                  </enum>
               </simple-type>
               </attribute>
            </structure>
         </attribute>
         <attribute name="ifOperStatus">
            <structure>
               <attribute name="ifIndex">
                  <type>kalua:int</type>
               </attribute>
               <attribute name="ifOperStatus">
                  <description>
   The current operational state of the interface.  The
   testing(3) state indicates that no operational packets can
   be passed.  If ifAdminStatus is down(2) then ifOperStatus
   should be down(2).  If ifAdminStatus is changed to up(1)
   then ifOperStatus should change to up(1) if the interface is
   ready to transmit and receive network traffic; it should
   change to dormant(5) if the interface is waiting for
   external actions (such as a serial line waiting for an
   incoming connection); it should remain in the down(2) state
   if and only if there is a fault that prevents it from going
   to the up(1) state; it should remain in the notPresent(6)
```

```
   state if the interface has missing (typically, hardware)
   components.
                  </description>
            <simple-type>
               <enum>
                  <enum-literal name="up" value="1"/>
                  <enum-literal name="down" value="2"/>
                  <enum-literal name="testing" value="3"/>
                  <enum-literal name="unknown" value="4"/>
                  <enum-literal name="dormant" value="5"/>
                  <enum-literal name="notPresent" value="6"/>
                  <enum-literal name="lowerLayerDown" value="7"/>
               </enum>
            </simple-type>
               </attribute>
            </structure>
         </attribute>
      </class>
   </kalua:module>
```

Appendix C. Netconf Payload Example

The XML example below shows a Netconf payload that is in line with the Kalua module shown in the previous sections:


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    This is an example of an NETCONF instance document
    for the Kalua module rfc1213-mib.xml
-->
<rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data xmlns:rfc1213="iso.org.dod.internet.mgmt.mib-2.rfc1213">
      <rfc1213:system>
         <sysDescr>IP Router</sysDescr>
         <sysObjectID>8.0.1.2.3.5.63.22.3.4</sysObjectID>
         <sysUpTime>646467347</sysUpTime>
         <sysContact>Bob's phone: (352) 465 3746 available 24/7
         </sysContact>
         <sysName>Bob's router</sysName>
         <sysLocation>Bob's garage</sysLocation>
         <sysServices>6</sysServices>
      </rfc1213:system>
      <rfc1213:interfaces>
         <ifNumber>1</ifNumber>
         <rfc1213:ifEntry>  <!-- classes have fully qualified
          namespaces, as they are reusable -->
            <ifIndex>1</ifIndex>
            <ifDescr>Flintstone Inc Ethernet A562</ifDescr>
            <ifType>10</ifType>
            <!-- corresponds to iso88026_man -
            enum-literal's value overrules name -->
            <ifMtu>1500</ifMtu>
            <ifSpeed>10000000</ifSpeed>
            <ifPhysAddress>0:12:3f:7d:b5:8b</ifPhysAddress>
            <ifAdminStatus>1</ifAdminStatus>      <!-- up -->
            <ifOperStatus>1</ifOperStatus>        <!-- up -->
         </rfc1213:ifEntry>
      </rfc1213:interfaces>
      <rfc1213:at>
         <rfc1213:atEntry>
            <atIfIndex>1</atIfIndex>
            <atPhysAddress>0:23:be:8e:00:6a</atPhysAddress>
            <atNetAddress>192.168.2.1</atNetAddress>
         </rfc1213:atEntry>
      </rfc1213:at>
      <rfc1213:ip>
         <ipForwarding>1</ipForwarding>
```

```
            <rfc1213:ipAddrEntry>
               <ipAdEntIfIndex>1</ipAdEntIfIndex>
               <ipAdEntAddr>10.34.120.3</ipAdEntAddr>
            </rfc1213:ipAddrEntry>
            <rfc1213:ipAddrEntry>
               <ipAdEntIfIndex>1</ipAdEntIfIndex>
               <ipAdEntAddr>10.22.255.255</ipAdEntAddr>
            </rfc1213:ipAddrEntry>
         </rfc1213:ip>
         <rfc1213:icmp/>
         <rfc1213:tcp/>
         <rfc1213:udp/>
      </data>
   </rpc-reply>
```

Appendix D. NETCONF Notification Example

The XML example below shows a NETCONF notification for the Kalua module rfc1213-mib.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is an example of a NETCONF notification for the Kalua
module rfc1213-mib.xml -->

<rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data xmlns:rfc1213="iso.org.dod.internet.mgmt.mib-2.rfc1213">

      <notification
        xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
         <eventTime>2007-07-08T00:01:00Z</eventTime>
         <rfc1213:linkDown>
            <ifIndex>1</ifIndex>
            <ifAdminStatus>
            <ifAdminStatus>1</ifAdminStatus>
               <ifIndex>1</ifIndex>
            </ifAdminStatus>
            <ifOperStatus>
               <ifOperStatus>2</ifOperStatus>
               <ifIndex>1</ifIndex>
            </ifOperStatus>
         </rfc1213:linkDown>
      </notification>
   </data>
</rpc-reply>
```

Appendix E. DHCP example from RCDML Requirements Document

The following XML example demonstrates the Kalua variant of the DHCP example in RCDML Requirements Document [RCDML].

```
<kalua:module name="DHCP" xmlns:kalua="urn:ietf:params:xml:ns:kalua:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:kalua:1:\Users\kalua\kalua.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <presentation>DHCP</presentation>
    <description>
       DHCP example, as in draft-presuhn-rcdml-03#appendix-C
    </description>
    <ns-uri>http://example.org/ns/dhcp</ns-uri>
    <ns-prefix>dhcp</ns-prefix>
    <release>1</release>
    <organization>Nokia Siemens Networks</organization>
    <import>
        <ns-uri>urn:ietf:params:xml:ns:netmod:base</ns-uri>
        <ns-prefix>ndl</ns-prefix>
    </import>
    <import>
        <ns-uri>http://example.com/ns/int</ns-uri>
        <ns-prefix>int</ns-prefix>
        <description>interfaces</description>
    </import>
    <class name="dhcp">
        <attribute name="default_lease_time">
            <presentation>default-lease-time</presentation>
            <type>kalua:int</type>
        </attribute>
        <attribute name="max_lease_time">
            <presentation>max-lease-time</presentation>
            <type>kalua:int</type>
        </attribute>
    </class>
    <relationship name="subnets">
        <kind>
            <containment/>
        </kind>
        <source>
            <class>dhcp</class>
            <role>parent</role>
        </source>
        <target>
            <class>subnet</class>
            <role>children</role>
        </target>
```

```
    </relationship>
    <class name="subnet">
        <attribute name="network">
            <type>ndl:ipAddress</type>
        </attribute>
        <attribute name="prefix_length">
            <presentation>prefix-length</presentation>
            <type>kalua:int</type>
        </attribute>
        <attribute name="range">
            <optional/>
            <type>rangeType</type>
        </attribute>
        <attribute name="max_lease_time">
            <presentation>max-lease-time</presentation>
            <type>kalua:int</type>
        </attribute>
        <attribute name="leases">
            <read-only/>
            <sequence elementName="lease">
                <structure>
                    <attribute name="ip_address">
                        <presentation>ip-address</presentation>
                        <type>ndl:ipAddress</type>
                    </attribute>
                    <attribute name="starts">
                        <type>kalua:dateTime</type>
                    </attribute>
                    <attribute name="ends">
                        <type>kalua:dateTime</type>
                    </attribute>
                    <attribute name="mac_address">
                        <presentation>mac-address</presentation>
                        <type>ndl:nsapAddress</type>
                    </attribute>
                    <key scope="local">
                        <member>ip_address</member>
                    </key>
                </structure>
            </sequence>
        </attribute>
        <attribute name="interface_filter">
            <sequence elementName="interface">
                <type>kalua:string</type>
            </sequence>
        </attribute>
        <key scope="global">
            <member>network</member>
```

```
            <member>prefix_length</member>
        </key>
    </class>
    <typedef name="rangeType">
        <structure>
            <attribute name="dynamic_bootp">
                <presentation>dynamic-bootp</presentation>
                <type>kalua:boolean</type>
                <defaultValueLiteral>true</defaultValueLiteral>
            </attribute>
            <attribute name="low">
                <mandatory/>
                <type>ndl:ipAddress</type>
            </attribute>
            <attribute name="high">
                <mandatory/>
                <type>ndl:ipAddress</type>
            </attribute>
        </structure>
    </typedef>
    <relationship name="dhcp_options_Rel">
        <kind>
            <containment/>
        </kind>
        <source>
            <class>dhcp</class>
            <role>dhcp</role>
        </source>
        <target>
            <class>dhcp_options</class>
            <role>dhcp_options</role>
        </target>
    </relationship>
    <class name="dhcp_options">
        <presentation>dhcp-options</presentation>
        <attribute name="router_list">
            <presentation>router-list</presentation>
            <sequence elementName="router">
                <type>ndl:ipAddress</type>
            </sequence>
        </attribute>
        <attribute name="domain_list">
            <presentation>domain-list</presentation>
            <sequence elementName="domain">
                <type>ndl:ipAddress</type>
            </sequence>
        </attribute>
        <attribute name="custom">
```

```
            <structure>
                <attribute name="option">
                    <type>kalua:int</type>
                </attribute>
                <attribute name="ip_address">
                    <presentation>ip-address</presentation>
                    <type>ndl:ipAddress</type>
                </attribute>
                <attribute name="string">
                    <type>kalua:string</type>
                </attribute>
            </structure>
        </attribute>
    </class>
    <relationship name="filtered_interfaces">
        <kind>
            <calculated>
                <condition>
                    $source/interface_filter/interface=$target/ifName
                </condition>
            </calculated>
        </kind>
        <source>
            <class>subnet</class>
            <role>filtering_subnet</role>
            <maxCardinality>unbounded</maxCardinality>
        </source>
        <target>
            <class>int:interface</class>
            <role>filtered_interface</role>
            <maxCardinality>unbounded</maxCardinality>
        </target>
    </relationship>
    <class name="shared_network">
        <attribute name="name">
            <type>kalua:string</type>
        </attribute>
        <key scope="global">
            <member>name</member>
        </key>
    </class>
    <relationship name="shared_network_subnets">
        <kind>
            <containment/>
        </kind>
        <source>
            <class>shared_network</class>
            <role>parent</role>
```

```
        </source>
        <target>
            <class>subnet</class>
            <role>children</role>
        </target>
    </relationship>
</kalua:module>
```

Appendix F. DHCP augmentation example from RCDML Requirements Document

The XML example below shows the Kalua variant of the DHCP augmentation example in RCDML Requirements Document [RCDML].

```
<kalua:module name="DHCP_augment"
xmlns:kalua="urn:ietf:params:xml:ns:kalua:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:kalua:1:\Users\kalua\kalua.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <presentation>DHCP_augment</presentation>
    <description>DHCP augmentation example, as in
      http://tools.ietf.org/html/draft-presuhn-rcdml-03#appendix-C
    </description>
    <ns-uri>http://example.org/ns/cal</ns-uri>
    <ns-prefix>cal</ns-prefix>
    <release>1</release>
    <organization>Nokia Siemens Networks</organization>
    <import>
       <ns-uri>http://example.org/ns/dhcp</ns-uri>
       <ns-prefix>dhcp</ns-prefix>
    </import>
    <class name="extended_dhcp_options">
       <description>
       Inheritance would imply substitution of the element
       name as well, which is not the case here. An additional
       augmentation mechanism would be needed to truly support
       this case.
     </description>
       <super-class>dhcp:dhcp_options</super-class>
       <attribute name="timezone">
          <type>kalua:string</type>
       </attribute>
    </class>
</kalua:module>
```

Appendix G. Example for Partial Lock RPC for NETCONF

The XML example below shows a Kalua example for Partial Lock RPC for NETCONF.

```
<kalua:module name="NCPL" xmlns:kalua="urn:ietf:params:xml:ns:kalua:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:kalua:1:\Users\kalua\kalua.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <presentation>NETCONF partial lock</presentation>
    <description>NETCONF partial lock operations</description>
    <ns-uri>urn:ietf:params:xml:ns:netconf:partial-lock:1.0</ns-uri>
    <ns-prefix>ncpl</ns-prefix>
    <release>1</release>
    <organization>IETF</organization>
    <import>
        <ns-uri>urn:ietf:params:xml:ns:netconf:base:1.0</ns-uri>
        <ns-prefix>nc</ns-prefix>
    </import>
    <attribute-group name="lock_id_attribute">
        <attribute name="lock_id">
            <type>kalua:unsignedInt</type>
        </attribute>
    </attribute-group>
    <operation name="partial_lock">
        <description>
        This operation defines the element for partial-lock RPC
        operation. Positive response to this operation is the
        "lock-id" element.
        </description>
        <input>
            <attribute name="config_name">
                <type>nc:config_name</type>
            </attribute>
            <attribute name="select">
                <sequence>
                    <type>kalua:string</type>
                </sequence>
            </attribute>
        </input>
        <output>
            <use>
                <attribute-group>sadasd</attribute-group>
            </use>
        </output>
    </operation>
    <operation name="partial_unlock">
        <description>
```

```
        This operation defines the element for partial-unlock RPC
        operation. The standard positive response
        (rpc-reply with &lt;nc:ok/&gt;) is sent if the operation
        succeeds.
        </description>
        <input>
            <use>
                <attribute-group>sadasd</attribute-group>
            </use>
        </input>
    </operation>
</kalua:module>
```

Appendix H. Support of RCDML Requirements in Kalua

Following table shows the support of RCDML Requirements in Kalua.

RCDML Requirements	Kalua support	Comments
1. Consequences of NETCONF		
1.1. Notification Definition (Agreed)	Yes	
1.2. Notification Get (NOT Agreed)	Yes (*)	Reuse of config definitions possible, not mandatory
1.3. Locking (Agreed)	Yes	
1.4. All Base Operations (Agreed)	Yes	
1.5. Define new NETCONF Operations (Agreed)	Yes (wo)	
1.6. Separation of Operations and Payload (Agreed)	Yes	
1.7. Error Annotation (Agreed)	Yes (wo)	In parallel to Req. # 1.5
1.8. No Mixed Content (Agreed)	Yes	
2. Model Representation Requirements		
2.1. Human Readable (Agreed)	Yes	
2.2. Machine Readable (Agreed)	Yes	
2.3. Textual Representation (Agreed)	Yes	
2.4. Document Information (Agreed)	Yes	
2.5. Ownership and Change Control (Agreed)	Yes	

2.6. Dependency Risk Reduction (Agreed)	Yes	
2.7. Diff-Friendly (Agreed)	Yes	
2.8. Internationalization and Localization	Yes	
2.8.1. Descriptions using Local Languages (Agreed)	Yes	
2.8.2. UTF-8 Encoding (Agreed)	Yes	
2.8.3. Localization Support (Agreed)	Yes	
2.8.4. Error String Localization (Agreed)	Yes	
2.8.5. Tag Names and Strings in Local Languages (NOT agreed)	No	
3. Reusability Requirements		
3.1. Modularity (Agreed)	Yes	
3.2. Reusable Definitions (Agreed)	Yes	
3.3. Modular extension (Agreed)	Yes	
4. Instance Data Requirements		
4.1. Default Values on the Wire (Agreed)	Yes (wo)	
4.2. Ordering		
4.2.1. Ordered Lists (Agreed)	Yes	
4.2.2. Order within Containers (NOT Agreed)	No	Not for containment relationships.
4.2.3. Interleaving (NOT Agreed)	Yes (*)	Order of contained elements is not defined

4.3. Validation			
4.3.1. Validate Instance Data (Agreed)	Yes	No explicit definition of valid and well-formed	
4.3.2. Tools to Validate Instance Data (NOT Agreed)	No		
4.4. Instance Canonicalization (Agreed)	Yes (wo)		
4.5. Character Set and Encoding (Agreed)	Yes		
4.6. Model Instance Localization (NOT Agreed)	Yes (*)		
5. Semantic Richness Requirements			
5.1. Human-Readable Semantics (Agreed)	Yes		
5.2. Basic Types (Agreed)	Yes		
5.3. Handling Opaque Data (Agreed)	Yes (wo)	kalua:any type can be added easily	
5.4. Keys			
5.4.1. Define Keys (Agreed)	Yes		
5.4.2. Deep Keys (NOT Agreed)	Yes		
5.5. Relationships			
5.5.1. Simple Relationships (Agreed)	Yes		
5.5.2. Many-to-Many Relationships (NOT Agreed)	Yes (*)		
5.5.3. Retrieve Relationships instance (NOT Agreed)	Yes (*)		

5.5.4. Retrieve Relationships - qualified (NOT Agreed)	Yes (*)	
5.6. Hierarchical Data	Yes	
5.7. Referential Integrity		
5.7.1. Referential Integrity (NOT Agreed)	Yes (wo) (*)	Calculated relationships do not imply referential integrity. Reference relationships only cover the key attributes
5.7.2. Extended Referential Integrity (NOT Agreed)	No	Not really clear what this is ...
5.7.3. Referential Integrity Robustness (NOT Agreed)	Yes (*)	
5.8. Characterize Data (Agreed)	Yes	
5.9. Defaults		
5.9.1. Default Values (NOT Agreed)	Yes (*)	
5.9.2. Dynamic Defaults (NOT Agreed)	No	
5.10. Formal Constraints		
5.10.1. Formal Description of Constraints (Agreed)	Yes	
5.10.2. Multi-element Constraints (NOT Agreed)	No	
5.10.3. Non-Key Uniqueness (Agreed)	Yes	
5.11. Units (Agreed)	Yes	
5.12. Define Actions (NOT Agreed)	No	
6. Extensibility Requirements		

6.1. Language Extensibility	Yes	
6.1.1. Language Versioning (Agreed)	Yes	
6.1.2. User Extensions (NOT Agreed)	Yes (*)	
6.1.3. Mandatory Extensions (NOT Agreed)	No	
6.2. Model Extensibility		
6.2.1. Model Version Identification (Agreed)	Yes	
6.2.2. Interaction with defaults (NOT Agreed)	No	
6.2.3. Conformance Interference (NOT Agreed)	Yes (wo) (*)	
6.2.4. Obsolete Portions of a Model (Agreed)	Yes	Solution assumes that the manager will select the correct version of a module (matching the agent)
6.3. Instance Data Extensibility		
6.3.1. Schema Version of Instance (NOT Agreed)	Yes (wo) (*)	
6.3.2. Interaction with default Values (NOT Agreed)	No	
6.3.3. Backwards Compatibility (Agreed)	Partially	Additional inheritance may lead to element names not understood by old clients.

6.3.4. Forwards Compatibility (NOT Agreed)	No	Unclear about implications of this requirement
7. Talking About Conformance		
7.1. Conformance to the Modeling Language (NOT Agreed)	Yes (*)	
7.2. Conformance to a Model (Agreed)	Yes	
8. Techno-Political Constraints		
8.1. Standard Technology (NOT Agreed)	Yes (*)	
8.2. Translate Models to Other Forms (Agreed)	Yes	
8.3. Minimize SMI Translation Pain (NOT Agreed)	No	
8.4. Generate Models from Other Forms (NOT Agreed)	Yes (*)	
8.5. Isolate Models from Protocol (NOT Agreed)	Yes (*)	
8.6. Library Support (NOT Agreed)	Yes (*)	
8.7. RFC 3139 Considerations	Yes	
8.8. RFC 3216 Considerations	Yes	

Table 2: Support of RCDML Requirements in Kalua

(wo): work ongoing

(*): Not agreed RCDML requirement supported by Kalua

Appendix I. Support of Use Cases for SMI and MIB Modules

Following table shows the support of Use cases for SMI and MIB modules in Kalua.

#	Use cases for SMI and MIB modules	Supported by Kalua
1	Agreement on a set of standard knobs (or proprietary knobs in a proprietary MIB module). These knobs can be defined in a clear and unambiguous manner including the restrictions that apply to the knobs, such as range, character set, what gets counted in a counter, and so on. Ideally, Netconf knobs would support all the types of management object properties already supported by MIB modules, unless it can be shown that such properties do not apply to configuration.	Yes
2	clear specification in the schema of what MUST/SHOULD/MAY/MUST NOT/SHOULD NOT be supported to claim compliance to the standard, to support vendor-neutral interoperability	Yes (*)
3	modular documents that can be formally validated using tools such as smilint	Yes
4	enough information for implementers to implement, with internal engineering choices being implementer-dependent, but with vendor-neutral formats on-the-wire	Yes
5	enough human-readable description/qualities that operators can read the raw schema to understand the meaning of a managed object	Yes
6	enough machine-readability that applications can effectively parse (and compare/contrast/utilize) the information across multiple vendor implementations, and across multiple vendor implementation releases.	Yes

7	the document can be used by network management applications to programmatically create corresponding databases of information (i.e. an NMS can IMPORT a MIB module to create corresponding record formats in a database)	Yes
8	each managed object is clearly instance-addressable such that a sender can label the object instance being sent in a message	Yes
9	the receiver of a message can clearly identify the instance of any object contained in a message, and can validate the data types and values (such as range, character set, etc.) of objects being passed in a message	Yes
10	a protocol-dependent message MAY contain a very small subset of the managed objects defined in a schema (e.g., one object from a schema) and still meet the previous requirements	Yes
11	a protocol-dependent message MAY contain a mixture of managed objects defined in different modules/schemas, and still meet the previous requirements	Yes

Table 3: Support of Use cases for SMI and MIB modules in Kalua

(*): Kalua does not define optional elements.

Authors' Addresses

Bernd Linowski
Nokia Siemens Networks
Heltorfer Strasse 1
40472 Duesseldorf
Germany

Email: bernd.linowski@nsn.com

Martin Storch
Nokia Siemens Networks
Heltorfer Strasse 1
40472 Duesseldorf
Germany

Email: martin.storch@nsn.com

Mikko Lahdensivu
Nokia Siemens Networks
Hatanpaeaen valtatie 30
33100 Tampere
Finland

Email: mikko.lahdensivu@nsn.com

Mehmet Ersue
Nokia Siemens Networks
St.-Martin-Strasse 76
81541 Munich
Germany

Email: mehmet.ersue@nsn.com