Information Services Working Group
https://spaces.internet2.edu/display/ISWG/Home

Aaron Brown, Internet2
Martin Swany, UDel
Jason Zurawski, Internet2
September 28, 2009

# An Extensible Schema for Network Topology

**Status of This Document**

This memo provides information for the NTAC community. It does not specify any standards or technical recommendations. Distribution is unlimited.

## 1 Abstract

The identification and description of network topology is a common problem for communities that rely on well defined distributed resources such as the Grid, performance measurement, or the control of static and dynamic network circuits. Work produced in related efforts, such as the Network Measurements Working Group (**NM-WG**) [11], Network Measurement Control Working Group (**NMC-WG**) [10] and Network Service Interface Working Group (**NSI-WG**) [12] have provided extensible mechanisms to describe and control the flow of information from networked components. These groups, to date, have not described a scalable, reusable, and succinct representation for the static and dynamic network elements they measure and control.
The following document describes the topology description language for two of these independent efforts: the **Control Plane** [3]community and **perfSONAR** [15]. This informational document is intended to present the representation and evolution of the topology descriptions used in these two communities.

## Contents

# 2 Introduction

This document presents an extensible encoding standard for *network topology* designed and implemented by specific parties, in particular members of the **Control Plane** [3] and **perfSONAR** [15] projects. In general, communities engaging in the use of dynamic resources (e.g. *Cloud Computing*, *The Grid*, *Dynamic Circuit Networks*) often find the need to describe the underlying network for resource acquisition and management. Network measurements often need to refer to the elements of *network infrastructure* as the *subject* (e.g. "source" or "destination") of their observed results. Initially members of related OGF working groups (e.g. **NM-WG**, **GHPN-RG**) made efforts to define canonical forms of recurring network elements. Subsequently, it was observed that by defining explicit *relationships* between these various elements, a succinct representation of the topology of the network could be achieved. As in the Network Measurement schema [17], we propose defining basic *structural* elements and varying the XML *namespace* to allow for layer- or technology-specific information.

## 2.1 Goals

The goal of this document is to define a *neutral* representation for network topology, currently used by the **Control Plane** community and **perfSONAR**. This work can be easily extended to support new types of networks and complements both the network measurement schema and control protocol in terms of representing the subjects of network measurements and relationship to the rest of the network at large. This topological representation is useful in its own right and can be applied in a variety of ways, including:

- Determining *provenance* of network measurements.
- Representing which elements *share* infrastructure.
- Location of appropriate points from which to measure.

A representation of the topology of the network described in this schemata should allow network measurement clients and servers to discover appropriate peers for measurements and to determine if any available data is relevant to a given path.

# 3 Design Philosophy

All network infrastructures should be described using basic components; any network representation must include appropriate elements to be both succinct and complete. As in the *Network Measurement schema*, one of the high-level goals of this representation is *normalizing* the data representation by removing as much *redundancy* as possible. The base schema is designed to facilitate easy incorporation into other network abstractions that service as extensions — this mirrors *object oriented software* design and is crux to this work. This easy extension can be accomplished through varying the namespace to indicate specific components.

## 3.1 Object Placement

There are two axes used to subdivide this network topology description: specific properties of the *OSI protocol model* [14] vs *technology* specific requirements. These two concerns are orthogonal and intersect cleanly, allowing the model to proceed in a natural manner.

A paramount concern of the schema design is describing network topology at various network *levels*. All specific topologies may be described, albeit with some amount of missing detail, in the most general manner using the "base" schema. As we move to more appropriate confines through the *OSI layers* and finally to the *technology specific* realm we are able to fully describe all necessary nuances but should observe that any of the layers has still provided a fungible representation.

### 3.1.1 Example Placement

As an example of appropriately modeling a network topology we choose an example that can easily be decomposed into atomic units: A *TCP* connection between two hosts. We first recognize that this is a simple relationship that may be described in general terms. After this initial step we will look toward both the *OSI layers* and *TCP* to extract further representations requirements.

#### 3.1.1.1 Base Model

Our first realization is that a TCP connection can be decomposed into basic components:

- Two hosts
- Network interfaces on each host
- Abstract "link" that represents an end to end connection
- Complexity in the middle of "link" that we can ignore

Reasonable first steps include attempting to design entities to represent the hosts and including identifying information about each: e.g. *hostnames*, *IP addresses*, and even items such as *operating system*. This information may be important to *someone*, particularly if we plan to share or otherwise export the description; it is important to be succint and complete when possible. After modeling the hosts we turn our attention to how theses hosts communicate over a network. This is normally done via the *network interface*. We should follow the same pattern for information retrieval as we did with the hosts: *speed* and *name* may be important here along with others things. Be sure to denote **all** interfaces on the machine as there may be several.

With our end systems modeled we now examine the "link" that connects and allows for end to end communication. Keeping the ideas simple, it is sufficient to note the abstraction connects the ports of two end hosts and may contain a simple name (e.g. *TCP connection between HostA and HostB*). The remainder of the "network" in the middle is not important for this example as we are concerned with a *TCP* connection: according to the *OSI layers* this translates to "Layer 4", the *Transport Layer*. In general we do not care about the "middle" for this type of connection and will only focus on the ends — if we were at a lower *layer* it would be important to model components that enable the entire link.

#### 3.1.1.2 Transport Layer Model

To re-model the same example at this layer does not require another definition of some of the key abstractions. Our first step is to take the above model, change the namespace on all elements, then attempt to come closer in representing the initial idea. Here is a modified version of the original list of requirements:

- Two hosts

- Network interfaces on each host including software and hardware details (e.g. *MTU*)
- Abstraction to model the connection between the two including address information, capacity, ports used, etc.

Because the basic building blocks are in place, we do not need to change any of our original assumptions and can add to our existing model. To the hosts we may add additional detail where necessary (e.g. details about *software* and *hardware* capabilities of the hosts). To the ports we will add configuration details such as *MTU* or even the size of the *transmit* and *receive* queues. Finally to the link we can add information we are aware of such as the *bottleneck capacity*, or if we were so inclined references to the sub links along the path if they were available.

If we still desire more detail regarding the connection, the final step we will explore is to define a specific use for the protocol we will be using for communication: *TCP*.

#### 3.1.1.3 TCP Model

A TCP specific schema can include more details that are not captured in the previous layers. Based on our previous modeling there are only minor details we may have missed

- TCP Details: *congestion control* algorithm, *buffer* sizes, *timeout* information
- Host Details: software *versions*
- Port Details: driver *versions*

These may be added to the aforementioned elements after the necessary namespace changes have been made. A further step we will not explore here is addition of other layers (as needed). For example if a specific applications (e.g. file transfer) has other needs beyond what is defined, it may be subdivided from the *TCP* layer. There is an infinite amount of layering that may be achieved via the building blocks of this model.

### 3.2 XML Namespaces

We have adopted XML namespaces to allow reuse of these same basic element names and vary the contents of the basic elements for each describing properties at various "layers": layers in this sense refers to the **OSI protocol model** as described in Section 3.1.1. As in the *Network Measurement schema*, a key design consideration is the observation that elements can be used to describe *any* network topology *layer*; this comes with the understanding that the exact contents may differ slightly between *layers*.

All namespace extensions **MUST** contain the elements defined in the *base* namespace. Each namespace **MAY** *redefine* the meaning of elements or *add* new elements but elements **SHOULD NOT** be removed that were previously defined. When released, each version of a namespace **MUST** specify the versions of their *parent* namespaces. If a new version of a parent namespace is released, the version of the child namespace **MUST** be changed to add any new elements or properties added in the parent. Example Namespaces:

- **Base**: *http://ogf.org/schema/network/topology/base/20070707/*
- **Layer 2 - Base Extension**: *http://ogf.org/schema/network/topology/l2/20070707/*
- **Ethernet - Layer 2 Extension**: *http://ogf.org/schema/network/topology/l2/ethernet/20070707/*

### 3.2.1 Namespace Versioning

The developers of the schemata realized early in the design process that new ideas will quickly depose older practices, particularly when reference implementations implement **RECOMMENDED** practices. This dual development track (e.g. implementations vs the creation of scalable standards) has forced a key change in the creation and use of XML namespaces. First recognized by members of the OGF community at large, a system to define identifying names uniquely and in a uniformly is paramount [7]. This namespace versioning scheme is a benefit for implementers as they can easily plan for backward and forward compatibility of community recommendations; the scheme also allows standards writers the freedom to introduce new and experimental ideas without pollution of the schema space.

This working group has adopted several components from the OGF community practice, but differ on some structural considerations. An example of the namespace format as adopted for the "base" namespace:

```
http://ogf.org/schema/network/topology/base/20070707/
```

Breaking down each portion of the namespace, we are able to decompose into the following components:

- **Scheme/Domain** - http://ogf.org
- **Customs** - /schema
- **Project** - /network/topology
- **Part/Extension** - /base
- **Version** - /20070707

The first key difference between this approach and the community document is a choice to subdivide the domain *later* in the namespace versus as the first item (e.g. instead of "http://schema.ogf.org" we are using "http://ogf.org/schema"). This choice was initially arbitrary, and **SHOULD NOT** change the overall intention of the approaches.

A second difference is the lack of a specific working group name in the namespace; this alteration from the specification is **intentional**. As a group that is working closely with and recommending topology descriptions for several other working groups, it is important not to place a *distinguishing mark* on the final result. Allowing the schema to live in a *common* area allows other groups to easily *incorporate* and *add* to the work performed in this body.

The final difference is the choice to place the version as the *last* entity of the namespace instead of immediately after a group designation. This choice is related to implementation details of software consuming these recommendations. To better support the *object oriented* design of the schema the full "name" of each element (including the namespace) needed to be present, sans version information. The easiest way to accomplish this is to place the version as the last piece of identifying information. This **SHALL NOT** change the meaning of the original community recommendation.

### 3.2.2 Namespace Expansion

Using this versioning scheme it is possible and expected that alternate approaches will manifest. The flexibility offered allows for this, and the working group fully expects strong candidates to emerge from this initial work. Specific expansion beyond the *OSI* layers into technology specific extensions (e.g. *Ethernet*, *SONET*, and *TCP*) are possible and encouraged.

# 4 Identifiers

The network topology identification scheme has been designed to be *globally unique*, *human-readable*, and *extensible*; the construction is in the style of Uniform Resource Names (**URNs**) [8]. The *namespace* for the URN is "ogf" and the *subnamespace* is "network". All identifiers **MUST** begin with "urn:ogf:network:" in this format. The remainder of the identifier consists of a series of name and value pairs. Each of these fields is separated by a ":" and each name and value is separated by a "=". The value **MUST** be encoded using the standard URN encoding scheme with one addition. The "=" character **MUST** be encoded with the value "

The named fields provide a truly unique identification schema as well as a natural hierarchy to the data similar to what is offered by the schemata we will present in Section 5.1. There are six major fields defined for network entities:

- domain
- node
- port
- link
- path
- service

A single "*" **SHOULD** appear instead of a name or value field to connote an identifier for an *unknown* network element of unknown type. If a "*" field is included, that field **MUST** be the last field in the identifier. The name and value pairs **MUST** be ordered from most general network entity to most specific network entity. For example, the domain field **MUST** come before the node field which **MUST** come before the port field. There **MUST NOT** exist repeated fields in the identifier.

## 4.1 Description of Fields

The following sections describe each field in depth. These fields closely match the intention of the XML schemata presented in Section 5.1.

### 4.1.1 Domain

The **domain** field **MUST** be used to describe the administrative domain in which the network element is located, either physically or logically. If this field is included, it **MUST** be positioned first and **MUST NOT** include any trailing spaces. The value **SHOULD** be the globally unique **DNS** name for the domain, setting the value of this field to "*" connotes the domain is *unknown*. While other options exist for this value (e.g. **AS** number), the simplicity and availability of the former motivate use.

**Examples**:

The first example describes a domain:

```
urn:ogf:network:domain=dcn.internet2.edu
```

The second example **MAY** be used when we do not know the domain:

```
urn:ogf:network:domain=*
```

Finally if after the domain field we specify a wildcard:

```
urn:ogf:network:domain=dcn.internet2.edu:*
```

### 4.1.2 Node

The **node** field **SHOULD** be a *host*, *router* or even an abstraction such as a *site*. An identifier for a physical, logical or virtual device in a domain **MUST** include the domain and node fields. The domain field **MUST** be identical to the domain field in the identifier for the domain containing the device. For instance if we have previously defined the domain to be "internet2.edu" we **MUST NOT** refer to this same domain by ASN "237" when talking about *any* item in this domain.

The position of this element will always be second **with respect to the domain**. There are no specific rules governing value of this element which allows for maximum flexibility when encoding. The value **SHOULD** be a human readable description of the device. Setting the value of the node field to "*" connotes the device is unknown.

**Examples**:

The first example uses the machine's unqualified *hostname* as a value. Note this is a natural extension since we already have specified a domain:

```
urn:ogf:network:domain=internet2.edu:node=packrat
```

The second example demonstrates that we **MAY** easily use a hosts address (**IPv4** or **IPv6**):

```
urn:ogf:network:domain=internet2.edu:node=207.75.164.10
```

Using a description instead of a host name (including spaces) is also acceptable:

```
urn:ogf:network:domain=internet2.edu:node=Joe's Machine
```

Finally we **MAY** leave the value unspecified by using the "*" element:

```
urn:ogf:network:domain=internet2.edu:node=*
```

### 4.1.3 Port

The **port** field describes the interface between a *node* and a *link* and commonly describes **Ethernet** interfaces, **IP** interfaces, or listening **TCP** sockets. The structure of the identifier will be decided based on if the interface is:

- *physical*
- logically *inside* of a *single device*
- constructed *across multiple devices*

If the interface is physical or logically inside a single device, the field **MUST** appear *after* the node. If the interface is a logical interface for a domain, the field **MUST** appear after the domain. A prudent choice for the value is the physical interface from a networked device for the first two options, or a logical name for the latter.

**Examples**:

The first example is common, simply using the designated physical interface name:

```
urn:ogf:network:domain=internet2.edu:node=packrat:port=eth0
```

The second example overloads the concept of the port and extends it to a TCP socket (tied to a specific address):

```
urn:ogf:network:domain=internet2.edu:node=packrat:port=207.75.164.10
```

Finally we **MAY** leave it unspecified:

```
urn:ogf:network:domain=internet2.edu:node=packrat:port=*
```

#### 4.1.4 Link

The **link** field **SHOULD** describe *logical* or *physical* links, regardless of direction. Bidirectional links **MUST** appear directly after the domain designation since they **SHOULD** connect multiple nodes. Unidirectional links, regardless of direction, **MUST** appear after a port. As in previous descriptions, there are no constraints placed on the value, but care **SHOULD** be taken to select a name with distinct meaning to the overall infrastructure.

##### 4.1.4.1 Unidirectional Intra-domain/Inter-domain Links

An identifier for a **unidirectional** link **MUST** include values for the *domain*, *node*, *port* and *link* fields. The undirectional link "belongs" to the port which utilizes the link; the fields that are contained in the link identifier **MUST** be identical to the fields that define the individual pieces it contains (e.g. *domain*, *node*, and *port*).

The value of the link field **MAY** be anything, but **MUST** be unique inside the port and **MUST NOT** include any trailing spaces. The value **SHOULD** be a human readable description of the link. Setting the value of the link field to "*" connotes that the actual link is unknown.

**Examples**:

Example designates a link for a given domain, node, and port tupple. Note spaces are allowed in this field:

```
urn:ogf:network:domain=internet2.edu:node=packrat:port=eth0:link=Link Between Packrat And Router1
```

We **MAY** also leave the name of the link unspecified:

```
urn:ogf:network:domain=internet2.edu:node=packrat:port=eth0:link=*
```

##### 4.1.4.2 Bidirectional Domain Links

An identifier for a **bidirectional** link that begins and ends solely in one domain **MUST** include the link field and **SHOULD** include the domain field. If the domain field is included, it **MUST** be identical to the previously specified domain field. The value of the link field **MAY** be anything, but **MUST** be unique inside the domain if the domain field is included or globally unique if not and **MUST NOT** include any trailing spaces.

The value **SHOULD** be a human readable description of the link and **SHOULD** be the same value as the *undirectional* links that make up the link. Setting the value of the link field to "*" connotes the actual link is unknown.

**Example**:

Similar to the **unidirectional** link example, we only need to specify the link in terms of the domain it exists within:

```
urn:ogf:network:domain=internet2.edu:link=Link Between Packrat And Router1
```

##### 4.1.4.3 Bidirectional Inter-domain Links

An identifier for a **bidirectional** link that begins and ends in different domains **MUST** include only a single item: **the link field**. The value of the link field **MAY** be anything, but **MUST** be globally unique and **MUST NOT** include any trailing spaces. The value **SHOULD** be a human readable description of the link

and **SHOULD** be the same value as the **undirectional** links that make up the link. Setting the value of the link field to "*" connotes the link is unknown.

**Examples**:

Similar to the previous examples, we do not specify a specific domain:

```
urn:ogf:network:link=Link Between Internet2 And ESNet urn:ogf:network:link=*
```

#### 4.1.5 Path

The **path** field **SHOULD** be used for circuits or other named paths. It is possible to create an identifier for a path that is completely contained within a single domain or spanning multiple domains. In either case, the path field **MUST** appear *immediately* after the domain field of the domain creating the identifier. The value **MAY** be anything, but **MUST** be unique to each domain.

**Examples**:

This example demonstrates a path within one domain.

```
urn:ogf:network:domain=internet2.edu:path=IN2P3_Circuit
```

#### 4.1.6 Service

The **service** field is used in identifiers for *services*, either hardware or software, offered by network elements. These **MAY** be used to describe services such *Web Services, ICMP responders*, or *optical converters*. This field will appear after the field for the item providing the service. For example, a service offered by a domain would appear *immediately after* the domain field. The value **MAY** be anything, but **SHOULD** be a human-readable description of the service provided.

**Examples**:

```
urn:ogf:network:domain=internet2.edu:node=packrat:service=http
```

```
urn:ogf:network:domain=internet2.edu:node=packrat:service=perfSONAR%20SNMP%20MA
```

## 5 Topology Model

Given the requirements for identifiers described in Section 4, there are several natural topologies elements we will focus on defining immediately for our "base" representation. In addition to the aformentioned concepts (e.g. *domain*, *node*, *port*, *link*, *path* and *service*) we will also introduce three convenience elements (*topology*, *network*, and *endpoint pair*) along with a key concept that allows extensibility: *relation*.

The following sections will go into detail on each element in the "base" schema. After motivating each element, we will explore tooling that allows for the defining of relationships between elements. Finally we will touch on the *extension* mechanism by introducing schemata that conform to the *OSI protocol model*.

### 5.1 Basic Elements

The "base" *schemata* defines the core elements that **SHOULD** be used to describe network topologies. Each element was included in the base to uniquely model a *common* concept — items that are not considered were deemed *inappropriate* for a generic network model due to inability to cleanly map to sub-schemata. The following sections present a collection of the top level *elements* and select detail regarding *sub-elements* and *attributes*.

### 5.1.1 Topology

The **topology** element serves as a "container" for storage and transmission of network topologies. This element does not serve as a topological element itself, but is provided as a conveyance for implementers of this schemata. An example use of this element follows, note that several elements that appear later in this document are included.

```
<topology>

  <domain id="urn:ogf:network:domain=domain.net">
    <node id="urn:ogf:network:domain=domain.net:node=hostA" />
    <link id="urn:ogf:network:domain=domain.net:link=hostA_to_domain2" />
  </domain>

  <node id="urn:ogf:network:domain=domain2.net:node=hostZ" />

</topology>
```

#### 5.1.1.1 Entities In The Domain

A topology **MAY** have any number of *domain*, *node*, *link*, *network* and *path* entities defined immediately underneath it. The elements **MUST** correspond to topological features contained in the domain.

### 5.1.2 Domain

The **domain** element serves as a *top-level* level element designed to enclose other topology elements into a logical administrative boundary. In technical terms, this should be viewed as the "demarcation point" for the enclosed entities. In practice a topology defined for an organization should contain only one domain element unless they control multiple *top-level* domains. Recursively defining domains is also possible (e.g. "internet2.edu" may contain "dcn.internet2.edu") in this system.

To illustrate use of the *domain* element consider this simple example. Note that the included elements represent other topology elements presented on this page.

```
<domain id="urn:ogf:network:domain=domain.net">

  <node id="urn:ogf:network:domain=domain.net:node=hostA" />

  <link id="urn:ogf:network:domain=domain.net:link=hostA_to_domain2" />

</domain>
```

The following represent the *attributes* and *elements* for the *domain* element. Beyond the identification *attributes* the remainder of the element is used to enclose the topology definition. These topological features will be defined in the subsequent sections.

#### 5.1.2.1 id Attribute

Each domain has an **id** attribute that contains the fully-qualified identifier for this domain. See Section 4 for information on constructing identifiers.

#### 5.1.2.2 idRef Attribute

A domain **MAY** contain the **idRef** attribute. If used, the **id** attribute **MUST NOT** be used. The purpose of this construct is to define *partial* information about a domain. When present the domain element **MAY** be interpreted as being "non-authoritative" and **SHOULD** be viewed as offering *additional* information not defined in the original element. See Section 4 for information on constructing identifiers.

#### 5.1.2.3 Entities In The Domain

A domain **MAY** have any number of *domain*, *node*, *link*, *network* and *path* entities defined immediately underneath it. The elements **MUST** correspond to topological features contained in the domain. The links that are defined **MUST** be *bidirectional* links; *unidirectional* links **MUST** be defined inside of ports.

### 5.1.3 Node

The **node** element **MAY** represent a *host*, a *network device*, or an abstract concept such as "site". The node element contains basic properties that apply to all nodes. The following example demonstrates a node with several attributes described.

```
<node id="urn:ogf:network:domain=domain.net:node=hostA">
  <name>hostA.domain.net</name>
  <address>192.168.0.1</address>
  <role>host</role>
  <description>Host in domain.net</description>
  <location>
    <floor>4</floor>
    <room>304</room>
    <cage>1</cage>
    <rack>5</rack>
    <shelf>2</shelf>
  </location>
  <contact>admin@domain.net</contact>
  <port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0" />
</node>
```

#### 5.1.3.1 id Attribute

Each node has an id attribute that contains the *fully-qualified* identifier for this node. See Section 4 for information on constructing identifiers.

#### 5.1.3.2 idRef Attribute

A node **MAY** contain the **idRef** attribute. If used, the **id** attribute **MUST NOT** be used. The purpose of this construct is to define *partial* information about a node. When present the node element **MAY** be interpreted as being "non-authoritative" and **SHOULD** be viewed as offering *additional* information not defined in the original element. See Section 4 for information on constructing identifiers.

#### 5.1.3.3 Name

Node elements **MAY** contain one or more name elements to describe the node. The name element contains a string and **MAY** be any value that describes the node. This **MAY** include the node's *hostname*, default **address**, or a **description** of the node.

##### 5.1.3.3.1 Type

The name element contains an attribute type that **SHOULD** be used to specify the format of the name element. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

#### 5.1.3.4 Address

Each node **MAY** have one or more address elements. This element encloses information that **MAY** be used locate the node. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

#### 5.1.3.5 Relations

One or more relation elements, see Section 5.1.4, **MAY** be defined inside the node.

#### 5.1.3.6 Lifetime

*Lifetime* is an **OPTIONAL** field that describes the time during which the node had the properties being described. In the case of ephemeral nodes this is defined to be the time the node was in existence. The lifetime element **MUST** include the *start* element to describe the beginning of the time range. The start element has one attribute, *type*. Type is **REQUIRED** and describes the format of the time value. Possible values could be "iso" for time formated in the ISO standards or "unix" for time measured in seconds past the epoch.

The lifetime element **MAY** also include either a *duration* or *end* element. Duration contains a string describing the amount of time that the entity persisted. End describes the end point of the time range. Each **MAY** also contain the aforementioned *type* attribute. If the lifetime element does not include an end or duration element, the end is *unspecified* and **SHOULD** be assumed to be infinite.

#### 5.1.3.7 Role

**Role** describes a succinct job or role the network entity maintains. The contents of this element are unspecified, but **SHOULD** correspond to the job of the node. For example if the node is a router, the contents **SHOULD** be "router". If the node is an endpoint, the contents **SHOULD** be "endpoint".

#### 5.1.3.8 Type

The **type** element, and related *type* attributes contained in other elements, is meant to describe the canonical use of the element. In the case of the node element we **MAY** define a particular use for this node (e.g. "host" or "switch"). The potential values of this element are purposely left undefined to allow for easy generalization.

#### 5.1.3.9 Description

Each node **MAY** include a **description** element. This element **SHOULD** include a human readable description of the node.

#### 5.1.3.10 Location Information

Each node **MAY** contain one **location** element which **SHOULD** be used to describe the *physical* location of the node. The purpose of including this element is to record basic information about where this entity is located. The element **MAY** consist of any of: *latitude*, *longitude*, *continent*, *country*, *zipcode*, *state*, *city*, *street address*, *floor*, *room*, *cage*, *rack*, and *shelf*.

The *latitude* and *longitude* elements are floating point values that describe the location on earth where the node is located. The *street address*, *city*, *state*, *zipcode* and *country* **MAY** be included to specify the address where the node is located. The *room*, *cage*, *rack* and *shelf* **MAY** be used to describe precisely where in the building a node is located.

#### 5.1.3.11 Contact Information

Each node **MAY** contain one or more **contact** elements containing information regarding contact information of equipment maintainers. The elements **MAY** have a *priority* attribute set which is a non-negative number. The priority attribute **SHOULD** be used to specify an *ordering* of the contact information with the primary contact being "0". If multiple contact elements share a priority, the ordering is unspecified. The lowest priority group consists of all nodes with no priority attribute set.

The contents of the contact element **MAY** consist of: *administrator*, *email*, *phone number*, and *institution*. The *administrator* element is a string that **SHOULD** contain the name of the administrator. The *institution* element is a string that **SHOULD** contain the name of the institution that has been tasked with administering this machine. The *email* element is a string that **SHOULD** contain the email address of the administrator or a help desk where questions about the node **MAY** be asked. The *phone number* element is a string that **SHOULD** contain the phone number of the administrator or the institution's help desk where questions about the node **MAY** be asked.

#### 5.1.3.12 Comments

Each node **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific node. These **SHOULD NOT** be used to store necessary information about the node.

#### 5.1.3.13 Port

One or more port elements, see Section 5.1.5, **MAY** be defined inside the node.

#### 5.1.3.14 Service

One or more service elements, see Section 5.1.6, **MAY** be defined inside the node.

### 5.1.4 Relation

The **relation** element is a general structure used to form a bond between elements of different types. This bond serves as an extension feature that is able to create complex structures using the small number of "core" elements defined in this schema. For instance *relationships* are used to subdivide a physical *ethernet port* into logical units or could be used to related links that share a common port (e.g. create the concept of a *bidirectional* link from two *unidirectional* links).

This element **MUST** contain a type as well as any number or elements that serve as references to topological elements. Possible elements include: *DomainIdRef*, *NodeIdRef*, *PortIdRef*, *LinkIdRef*, *PathIdRef*, *NetworkIdRef*, *ServiceIdRef*, and *GroupIdRef*. The following is a list of possible values for the *type* element. Note that this list features *essential* relationships that are easily reproduced in practice and included in the base for conveyance. Additional relation types are possible and encouraged in schema extensions.

- **contained-in/contains** - Describes elements that are used to *build* a different concept (e.g. a series of *VLAN links* may be contained in a higher order *end to end link*) or can be used to describe "containment" (e.g. one element physically or logically is within another).
- **incoming-link/outgoing-link** - *Port* elements will use this relationship to establish the direction of a *link* element
- **source/destination** - *Link* elements will use this relationship to establish the direction of transfer to specific ports
- **peer** - Describes elements that share a common container (e.g. *logical* ports that share a common *physical* port are **peers**).

An example of how to use this element in practice follows. This example, using only 3 elements, features several relations of various types. Though verbose, this approach yields maximum flexibility and allows the partial specification of topology.

```
<node id="urn:ogf:network:domain=domain.net:node=hostA">
  <relation type="contains">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0</portIdRef>
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth1</portIdRef>
  </relation>
</node>

<port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0">
  <relation type="contained-in">
    <nodeIdRef>urn:ogf:network:domain=domain.net:node=hostA</nodeIdRef>
  </relation>
  <relation type="peer">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth1</portIdRef>
  </relation>
</port>

<port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth1">
  <relation type="contained-in">
    <nodeIdRef>urn:ogf:network:domain=domain.net:node=hostA</nodeIdRef>
  </relation>
</port>

<port idRef="urn:ogf:network:domain=domain.net:node=hostA:port=eth1">
  <relation type="peer">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0</portIdRef>
  </relation>
</port>
```

The example illustrates the concept of defining a host (*node*) and two interfaces (*ports*) on this host. We use the following relationships:

- The *node* **contains** *ports*
- The *ports* are **contained in** a *node*
- The *ports* are **peers** of each other.

Note that in the case of *port eth1* we have defined the node first, then used an *idRef* to augment the initial information. This simulates partial information that may be defined over time or in some cases in different domains.

### 5.1.5 Port

Nodes contain **ports**, which are the attachment points of *link* elements. These entities have properties which are independent of the *node*. The following example shows a *port* with several sub elements included:

```
<port idRef="urn:ogf:network:domain=domain.net:node=hostA:port=eth1">
  <name>hostA.domain.net:eth1</name>
  <type>ethernet</type>
  <description>2nd Ethernet port on hostA.domain.net</description>
  <capacity>1000000</capacity>
  <mtu>1500</mtu>
  <link id="urn:ogf:network:domain=domain.net:link=hostA_to_domain2" />
</port>
```

#### 5.1.5.1 id Attribute

Each port has an id attribute that contains the *fully-qualified* identifier for this port. See Section 4 for information on constructing identifiers.

#### 5.1.5.2 idRef Attribute

A port **MAY** contain the **idRef** attribute. If used, the **id** attribute **MUST NOT** be used. The purpose of this construct is to define *partial* information about a port. When present the port element **MAY** be interpreted as being "non-authoritative" and **SHOULD** be viewed as offering *additional* information not defined in the original element. See Section 4 for information on constructing identifiers.

#### 5.1.5.3 Name

Port elements **MAY** contain one or more name elements to describe the port. The name element contains a string and **MAY** be any value that describes the node. This **MAY** include the **interface** of a host, the **address** for a Layer 3 port, or a simple **description** of the port.

##### 5.1.5.3.1 Type

The name element contains an attribute type that **SHOULD** be used to specify the format of the name element. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

#### 5.1.5.4 Address

Each port **MAY** have one or more address elements. This element encloses information that **MAY** be used locate the port. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

#### 5.1.5.5 Relation

One or more relation elements, see Section 5.1.4, **MAY** be defined inside the port.

#### 5.1.5.6 Lifetime

*Lifetime* is an **OPTIONAL** field that describes the time during which the port had the properties being described. In the case of ephemeral ports this is defined to be the time the port was in existence. The lifetime element **MUST** include the *start* element to describe the beginning of the time range. The start element has one attribute, *type*. Type is **REQUIRED** and describes the format of the time value. Possible values could be "iso" for time formated in the ISO standards or "unix" for time measured in seconds past the epoch.

The lifetime element **MAY** also include either a *duration* or *end* element. Duration contains a string describing the amount of time that the entity persisted. End describes the end point of the time range. Each **MAY** also contain the aforementioned *type* attribute. If the lifetime element does not include an end or duration element, the end is *unspecified* and **SHOULD** be assumed to be infinite.

#### 5.1.5.7 Type

The **type** element, and related *type* attributes contained in other elements, is meant to describe the canonical use of the element. In the case of the port element we **MAY** define a particular use for this port (e.g. "ethernet" or "sonnet"). The potential values of this element are purposely left undefined to allow for easy generalization.

#### 5.1.5.8 Description

Each port **MAY** include a **description** element. This element **SHOULD** include a human readable description of the port.

#### 5.1.5.9 Capacity

The *capacity* element describes the amount of data that **MAY** flow through the port.

#### 5.1.5.10 MTU

The *mtu* element, contains the maximum size an individual *segment* of data that **MAY** be transmitted with this port. If the port were an ethernet port, this would be the configured *MTU* for the port. If the port were a UDP port, the value would contain the maximum segment size for the port. This field **MUST** contain an integer corresponding to the number of bytes in the maximum size. If the specified value is "0", this means that the particular interface has no maximum segment size.

#### 5.1.5.11 Comments

Each port **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific port. These **SHOULD NOT** be used to store necessary information about the port.

#### 5.1.5.12 Link

One or more link elements, see Section 5.1.7, **MAY** be defined inside the port.

#### 5.1.6 Service

The **service** element **SHOULD** be used to describe services, either hardware of software. This could include high-level services like *Web Services* or low-level services such as a *translator* between optical wavelengths. A simple example of a *service* follows:

```
<service id=" urn:ogf:network:domain=domain.net:node=hostA:service=HTTPD">
  <name>hostA.domain.net:80</name>
  <description>Web Server - hostA.domain.net</description>
</service>
```

##### 5.1.6.1 id Attribute

Each service has an id attribute that contains the *fully-qualified* identifier for this service. See Section 4 for information on constructing identifiers.

##### 5.1.6.2 idRef Attribute

A service **MAY** contain the **idRef** attribute. If used, the **id** attribute **MUST NOT** be used. The purpose of this construct is to define *partial* information about a service. When present the service element **MAY** be interpreted as being "non-authoritative" and **SHOULD** be viewed as offering *additional* information not defined in the original element. See Section 4 for information on constructing identifiers.

##### 5.1.6.3 Name

Service elements **MAY** contain one or more name elements to describe the service. The name element contains a string and **MAY** be any value that describes the service. This **MAY** include the services's *hostname*, default **address**, or a **description** of the service.

###### 5.1.6.3.1 Type

The name element contains an attribute type that **SHOULD** be used to specify the format of the name element. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

##### 5.1.6.4 Type

The **type** element, and related *type* attributes contained in other elements, is meant to describe the canonical use of the element. In the case of the service element we **MAY** define a particular use for this service (e.g. "Web Service" or "transport"). The potential values of this element are purposely left undefined to allow for easy generalization.

##### 5.1.6.5 Description

Each service **MAY** include a **description** element. This element **SHOULD** include a human readable description of the service.

#### 5.1.6.6 Comments

Each service **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific service. These **SHOULD NOT** be used to store necessary information about the service.

#### 5.1.6.7 Relation

One or more relation elements, see Section 5.1.4, **MAY** be defined inside the service.

#### 5.1.6.8 Lifetime

*Lifetime* is an **OPTIONAL** field that describes the time during which the service had the properties being described. In the case of ephemeral services this is defined to be the time the service was in existence. The lifetime element **MUST** include the *start* element to describe the beginning of the time range. The start element has one attribute, *type*. Type is **REQUIRED** and describes the format of the time value. Possible values could be "iso" for time formated in the ISO standards or "unix" for time measured in seconds past the epoch.

The lifetime element **MAY** also include either a *duration* or *end* element. Duration contains a string describing the amount of time that the entity persisted. End describes the end point of the time range. Each **MAY** also contain the aforementioned *type* attribute. If the lifetime element does not include an end or duration element, the end is *unspecified* and **SHOULD** be assumed to be infinite.

### 5.1.7 Link

Entities (e.g.*nodes*, *ports* are connected via **link** elements. These entities **MAY** be viewed as *edges* in graph terminology. The following example shows some of the sub-elements that are possible in a *link*:

```
<link id="urn:ogf:network:domain=domain.net:link=hostA_to_domain2">
  <remoteLinkId>urn:ogf:network:domain=domain2.net:link=hostZ_to_domain</remoteLinkId>
  <type>bidirectional/type>
  <description>Inter-domain link from domain.net to domain2.net</description>
</link>
```

#### 5.1.7.1 id Attribute

Each link has an id attribute that contains the *fully-qualified* identifier for this link. See Section 4 for information on constructing identifiers.

#### 5.1.7.2 idRef Attribute

A link **MAY** contain the **idRef** attribute. If used, the **id** attribute **MUST NOT** be used. The purpose of this construct is to define *partial* information about a link. When present the link element **MAY** be interpreted as being "non-authoritative" and **SHOULD** be viewed as offering *additional* information not defined in the original element. See Section 4 for information on constructing identifiers.

#### 5.1.7.3 Name

Link elements **MAY** contain one or more name elements to describe the link. The name element contains a string and **MAY** be any value that describes the link.

#### 5.1.7.3.1 Type

The name element contains an attribute type that **SHOULD** be used to specify the format of the name element. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

### 5.1.7.4 Relation

One or more relation elements, see Section 5.1.4, **MAY** be defined inside the link.

### 5.1.7.5 Lifetime

*Lifetime* is an **OPTIONAL** field that describes the time during which the link had the properties being described. In the case of ephemeral links this is defined to be the time the link was in existence. The lifetime element **MUST** include the *start* element to describe the beginning of the time range. The start element has one attribute, *type*. Type is **REQUIRED** and describes the format of the time value. Possible values could be "iso" for time formated in the ISO standards or "unix" for time measured in seconds past the epoch.

The lifetime element **MAY** also include either a *duration* or *end* element. Duration contains a string describing the amount of time that the entity persisted. End describes the end point of the time range. Each **MAY** also contain the aforementioned *type* attribute. If the lifetime element does not include an end or duration element, the end is *unspecified* and **SHOULD** be assumed to be infinite.

### 5.1.7.6 RemoteLinkId

The **remoteLinkId** element is used to reference the "far" (e.g. *remote*) end of a link that **MAY** have a different name depending on domain policy. This link **MAY** also be used in cases where two *uni-directional* links are employed to describe a *bi-directional* connection.

### 5.1.7.7 Type

The **type** element, and related *type* attributes contained in other elements, is meant to describe the canonical use of the element. In the case of the link element we **MAY** define a particular use for this link (e.g. "logical" or "physical"). The potential values of this element are purposely left undefined to allow for easy generalization.

### 5.1.7.8 Description

Each link **MAY** include a **description** element. This element **SHOULD** include a human readable description of the node.

### 5.1.7.9 GlobalName

The **globalName** element **SHOULD** be used to give a *globally unique* name that describes a link. This element contains an attribute describing what *type* of name this element contains. The types and contents of the element are unspecified as the element is deprecated.

#### 5.1.7.10 Comments

Each link **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific link. These **SHOULD NOT** be used to store necessary information about the link.

### 5.1.8 Network

The **network** element **MAY** contain any number of *NodeIdRef*s, *LinkIdRef*s, or *PortIdRef*s which are used to define the set of elements contained in the network. If the *type* of the network has been set, all the elements in the network **MUST** be connected using the specified type. A simple example of a network is pictured below:

```
<network id="urn:ogf:network:domain=domain.net:network=network1">
  <description>Network1 in the domain.net domain</description>

  <node id="urn:ogf:network:domain=domain.net:node=hostA" />
  <link id="urn:ogf:network:domain=domain.net:link=hostA_to_domain2" />

</network>
```

#### 5.1.8.1 id Attribute

Each network has an id attribute that contains the *fully-qualified* identifier for this network. See Section 4 for information on constructing identifiers.

#### 5.1.8.2 idRef Attribute

A network **MAY** contain the **idRef** attribute. If used, the **id** attribute **MUST NOT** be used. The purpose of this construct is to define *partial* information about a network. When present the network element **MAY** be interpreted as being "non-authoritative" and **SHOULD** be viewed as offering *additional* information not defined in the original element. See Section 4 for information on constructing identifiers.

#### 5.1.8.3 NodeIdRef

The **NodeIdRef** element is used to reference external *node* elements.

#### 5.1.8.4 PortIdRef

The **PortIdRef** element is used to reference external *port* elements.

#### 5.1.8.5 LinkIdRef

The **LinkIdRef** element is used to reference external *link* elements.

#### 5.1.8.6 Name

Network elements **MAY** contain one or more name elements to describe the network. The name element contains a string and **MAY** be any value that describes the network.

#### 5.1.8.6.1 Type

The name element contains an attribute type that **SHOULD** be used to specify the format of the name element. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

### 5.1.8.7 Relation

One or more relation elements, see Section 5.1.4, **MAY** be defined inside the network.

### 5.1.8.8 Lifetime

*Lifetime* is an **OPTIONAL** field that describes the time during which the network had the properties being described. In the case of ephemeral networks this is defined to be the time the network was in existence. The lifetime element **MUST** include the *start* element to describe the beginning of the time range. The start element has one attribute, *type*. Type is **REQUIRED** and describes the format of the time value. Possible values could be "iso" for time formated in the ISO standards or "unix" for time measured in seconds past the epoch.

The lifetime element **MAY** also include either a *duration* or *end* element. Duration contains a string describing the amount of time that the entity persisted. End describes the end point of the time range. Each **MAY** also contain the aforementioned *type* attribute. If the lifetime element does not include an end or duration element, the end is *unspecified* and **SHOULD** be assumed to be infinite.

### 5.1.8.9 Type

The **type** element, and related *type* attributes contained in other elements, is meant to describe the canonical use of the element. In the case of the network element we **MAY** define a particular use for this node (e.g. "ethernet" or "TCP"). The potential values of this element are purposely left undefined to allow for easy generalization.

### 5.1.8.10 Description

Each network **MAY** include a **description** element. This element **SHOULD** include a human readable description of the network.

### 5.1.8.11 Comments

Each network **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific network. These **SHOULD NOT** be used to store necessary information about the network.

## 5.1.9 Path

A **path** defines a discrete path between two topological points. Each path contains some number of *hop*s that are used to describe portions of the entire path. A simple example of a path element is pictured below:

```
<path id="urn:ogf:network:domain=domain.net:path=Circuit_123">
  <hop id="0">
    <linkIdRef>urn:nml:domain.net:link=segment1</linkIdRef>
    <nextHop>1</nextHop>
  </hop>
  <hop id="1">
    <linkIdRef>urn:nml:domain.net:link=segment2</linkIdRef>
    <nextHop>2</nextHop>
  </hop>
  <hop id="2">
    <linkIdRef>urn:nml:domain.net:link=segment3</linkIdRef>
  </hop>
</path>
```

#### 5.1.9.1 id Attribute

Each path has an id attribute that contains the *fully-qualified* identifier for this path. See Section 4 for information on constructing identifiers.

#### 5.1.9.2 Relation

One or more relation elements, see Section 5.1.4, **MAY** be defined inside the path.

#### 5.1.9.3 Lifetime

*Lifetime* is an **OPTIONAL** field that describes the time during which the path had the properties being described. In the case of ephemeral paths this is defined to be the time the path was in existence. The lifetime element **MUST** include the *start* element to describe the beginning of the time range. The start element has one attribute, *type*. Type is **REQUIRED** and describes the format of the time value. Possible values could be "iso" for time formated in the ISO standards or "unix" for time measured in seconds past the epoch.

The lifetime element **MAY** also include either a *duration* or *end* element. Duration contains a string describing the amount of time that the entity persisted. End describes the end point of the time range. Each **MAY** also contain the aforementioned *type* attribute. If the lifetime element does not include an end or duration element, the end is *unspecified* and **SHOULD** be assumed to be infinite.

#### 5.1.9.4 Comments

Each path **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific path. These **SHOULD NOT** be used to store necessary information about the path.

#### 5.1.9.5 Hop

A **hop** is a decomposed section of a path. A hop **MAY** contain element references that are meant to describe involved entities.

##### 5.1.9.5.1 DomainIdRef

The **DomainIdRef** element is used to reference external *domain* elements.

##### 5.1.9.5.2 NodeIdRef

The **NodeIdRef** element is used to reference external *node* elements.

#### 5.1.9.5.3 PortIdRef

The **PortIdRef** element is used to reference external *port* elements.

#### 5.1.9.5.4 LinkIdRef

The **LinkIdRef** element is used to reference external *link* elements.

#### 5.1.9.5.5 PathIdRef

The **PathIdRef** element is used to reference external *path* elements.

#### 5.1.9.5.6 NetworkIdRef

The **NetworkIdRef** element is used to reference external *network* elements.

### 5.1.10 Endpoint Pairs

The **endPointPair** element is a construct that simulates an end-to-end connection; normally at the higher layers of communication (e.g. "TCP"). The notion of this element **MAY** be created using primitives we have discussed: two *port* elements and a *link* element. As this is a common occurrence, this element provides a shorthand notation. An example of this element is shown below:

```
<endPointPair>
  <src type="ipv4" value="10.0.0.1" />
  <dst type="ipv4" value="10.0.0.2" />
</endPointPair>
```

#### 5.1.10.1 Name

Endpoint Pair elements **MAY** contain one or more name elements to describe the endpoint pair. The name element contains a string and **MAY** be any value that describes the endpoint pair. This **MAY** include the endpoint pair's *hostname*, default **address**, or a **description** of the endpoint pair.

##### 5.1.10.1.1 Type

The name element contains an attribute type that **SHOULD** be used to specify the format of the name element. If the type of the element is unspecified or unknown, the value **SHOULD** be treated as an *opaque* string.

#### 5.1.10.2 Type

The **type** element, and related *type* attributes contained in other elements, is meant to describe the canonical use of the element. In the case of the endpoint pair element we **MAY** define a particular use for this node (e.g. "host" or "switch"). The potential values of this element are purposely left undefined to allow for easy generalization.

#### 5.1.10.3 Description

Each endpoint pair **MAY** include a **description** element. This element **SHOULD** include a human readable description of the endpoint pair.

#### 5.1.10.4 Comments

Each node **MAY** include one or more *comments* elements. These elements **SHOULD** be used to add human-readable comments about a specific node. These **SHOULD NOT** be used to store necessary information about the node.

#### 5.1.10.5 Src

The *source* for an endpoint pair is described using a **src** element. This element borrows the features of the *port* (see Section 5.1.5). It is also possible to use a *portIdRef* element to reference an already defined *port*.

#### 5.1.10.6 Dst

The *destination* for an endpoint pair is described using a **dst** element. This element borrows the features of the *port* (see Section 5.1.5). It is also possible to use a *portIdRef* element to reference an already defined *port*.

### 5.2 Element Relationships

Relationships within the schemata are accomplished in two ways:

- **Explicit** - Elements that share a relationship are often defined together (e.g. a *node* contains *ports*)
- **Implicit** - Relationships of certain types are created via the *relation* element using *idRef*s instead of being explicitly defined together. This allows relationships to be built over time.

The later approache to defining these interactions allows for greater flexibility in element use. It is natural to describe certain relationships such as those that exist between *nodes*, *ports*, and *links*. An example of the former approach of defining element relations a the same time is pictured below:

```
<node id="urn:ogf:network:domain=domain.net:node=hostA">
  <port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0" />
  <port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth1" />
</node>
```

The same relationship is possible by defining the elements in different approaches (or environments) and combining at a later time. Note that we are relaying the same information, just in a fragmented form:

```
<node id="urn:ogf:network:domain=domain.net:node=hostA">
  <relation type="contains">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0</portIdRef>
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth1</portIdRef>
  </relation>
</node>

<port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0">
  <relation type="contained-in">
    <nodeIdRef>urn:ogf:network:domain=domain.net:node=hostA</nodeIdRef>
  </relation>
</port>

<port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth1">
  <relation type="contained-in">
    <nodeIdRef>urn:ogf:network:domain=domain.net:node=hostA</nodeIdRef>
  </relation>
</port>
```

The “easy” interactions were built directly into the schema since they represent common use cases. Non-standard use cases, particularly those that interest specific communities and implementations, are impossible to predict and therefore **MUST** be built in an ad-hoc fashion using the *relation* element. The following represent the “core” relationships in this schema:

- **contained-in/contains**
- **incoming-link/outgoing-link**
- **source/destination**
- **peer**

As described in Section 5.1.4, the *relation* element can exist in any of the basic topology elements and **MAY** be used more than once. Each element contains an extensible *type* that can be filled in with the aforementioned relationships or re-defined to include unexplored options. Once a *type* is selected, *idRef* elements make up the remaining portion of this definition. It is important to note there are no limits on the specific *idRef* elements to be used (e.g. *node*, *port*) or the amount that **MAY** be used. This malleability allows for a general object capable of representing concrete and abstract relationships with ease. The schema authors realize that some cases are more common than others, and it is possible that the existing *types* do not fully cover all possible relationships. The following sections serve as a guide to working within the confines of the defined types.

### 5.2.1 Node Relationships

The *node* element will commonly use the **contained-in/contains** and **peer** relationship types. Some potential use cases:

- A *node* is **contained-in** a *domain*
- A *node* is **contained-in** a *network*
- A *node* is **contained-in** a *path* (e.g. via the “hop” element)
- A *node* **contains** *ports*
- A *node* **contains** *services*
- A *node* is a peer of other *nodes*

### 5.2.2 Port Relationships

The *port* element uses the **incoming-link/outgoing-link**, **contained-in/contains**, and **peer** relationship types. Some potential use cases:

- A *port*’s **incoming-link** is a *link*
- A *port*’s **outgoing-link** is a *link*
- A *port* is **contained-in** a *domain*

- A *port* is **contained-in** a *network*
- A *port* is **contained-in** a *path* (e.g. via the "hop" element)
- A *port* is **contained-in** a *node*
- A *port* **contains** other *ports*
- A *port* is a peer of other *ports*

#### 5.2.3 Link Relationships

The *link* element uses the **source/destination**, **contained-in/contains**, and **peer** relationship types. Some potential use cases:

- A *link*'s **source** is a *port*
- A *link*'s **destination** is a *port*
- A *link* is **contained-in** a *domain*
- A *link* is **contained-in** a *network*
- A *link* is **contained-in** a *path* (e.g. via the "hop" element)
- A *link* is **contained-in** other *links*
- A *link* **contains** *ports*
- A *link* **contains** *paths* (of a "lower" layer).
- A *link* **contains** other *links*
- A *link* is a peer of other *links*

#### 5.2.4 Network Relationships

The *network* element will commonly use the **contained-in/contains** and **peer** relationship types. Some potential use cases:

- A *network* is **contained-in** a *domain*
- A *network* is **contained-in** other *networks*
- A *network* is **contained-in** a *path* (e.g. via the "hop" element)
- A *network* **contains** *ports*
- A *network* **contains** *nodes*
- A *network* **contains** *services*
- A *network* **contains** *links*

- A *network* **contains** *paths*
- A *network* **contains** other *paths*
- A *network* is a peer of other *networks*

#### 5.2.5 Path Relationships

The *path* element will commonly use the **contained-in/contains** and **peer** relationship types. Some potential use cases:

- A *path* is **contained-in** a *domain*
- A *path* is **contained-in** a *network*
- A *path* **contains** *ports* (e.g. via the "hop" element)
- A *path* **contains** *nodes* (e.g. via the "hop" element)
- A *path* **contains** *links* (e.g. via the "hop" element)
- A *path* **contains** *services* (e.g. via the "hop" element)
- A *path* **contains** *networks* (e.g. via the "hop" element)
- A *path* is a peer of other *paths*

#### 5.2.6 Service Relationships

The *service* element will commonly use the **contained-in/contains** and **peer** relationship types. Some potential use cases:

- A *service* is **contained-in** a *domain*
- A *service* is **contained-in** a *network*
- A *service* is **contained-in** a *node*
- A *service* is **contained-in** other *services*
- A *service* **contains** *ports*
- A *service* **contains** other *services*
- A *service* is a peer of other *services*

### 5.3 Time dimension

Network topology may remain "static" for many use cases: backbone, regional, and campus networks will maintain the same configuration for an indefinite amount of time. The alternate situation of a "dynamic" network must also be planned for: networks that change configuration makeup as well as *dynamic circuit network* should still be able to use the same topological descriptions. The addition of "time aware" markup to the existing topology elements allows for the definition of entities along a *time dimension*. This indicates that a given domain may feature elements with the same *identification*, but potentially different *active* periods.

As described in Section 5.1, most of the "core" elements (*node*, *port*, *link*, *network*, *path*, and *service*) feature the *lifetime* specification. *Lifetime* is an **OPTIONAL** field that describes the time during which the element had the properties being described (or in the case of ephemeral elements this is defined to be the time of existence). Consider the following examples:

```
<port id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0">
  <relation type="outgoing-link">
    <linkIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0:link=hostA_to_hostB</linkIdRef>
  </relation>
</port>

<port id="urn:ogf:network:domain=domain.net:node=hostB:port=eth0">
  <relation type="incoming-link">
    <linkIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0:link=hostA_to_hostB</linkIdRef>
  </relation>
</port>

<link id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0:link=hostA_to_hostB">
  <capacity>1000000</capacity>
  <relation type="source">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0</portIdRef>
  </relation>
  <relation type="destination">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostB:port=eth0</portIdRef>
  </relation>
  <lifetime>
    <start type="unix">1142841539</start>
    <end type="unix">1142845139</end>
  </lifetime>
</link>

<link id="urn:ogf:network:domain=domain.net:node=hostA:port=eth0:link=hostA_to_hostB">
  <capacity>5000000</capacity>
  <relation type="source">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostA:port=eth0</portIdRef>
  </relation>
  <relation type="destination">
    <portIdRef>urn:ogf:network:domain=domain.net:node=hostB:port=eth0</portIdRef>
  </relation>
  <lifetime>
    <start type="unix">1142845139</start>
  </lifetime>
</link>
```

This example features two *ports* and a *link* that connects them. Note we have specified the *link* twice: first with a *one hour* lifetime and then with an *indefinite* lifetime. This reason for this dual specification is physical: the *capacity* has changed. This concept may be replicated on other elements, but is most commonly seen with *links* (e.g. **VLAN** creation) and *ports* (e.g. *virtual interface* creation).

### 5.4 Layer 2 Elements

Building on the generic layer established by the elements in the "base" (see Section 5.1) it is possible to more robustly define specific extensions for various layers and technologies. We present the *Layer 2* extensions here that describe topology on the *Data Link Layer*.

#### 5.4.1 Type

There are no structural changes to this element from the "base". When used the type **SHOULD** reflect the *type*s that are available in the *Data Link Layer* (e.g. "ethernet", "sonnet").

#### 5.4.2 Port

There are some minor additions to the *port* element that are discussed below.

##### 5.4.2.1 Address

There are no structural changes to this element from the "base". The *address* element in *Layer 2* port elements **MUST** consist of the *Layer 2* address.

##### 5.4.2.2 ifName

The **ifName** element is a new addition to the *Layer 2* namespace. This element can be used to specify the *SNMP* name of the interface. The ifName element contains a string representation of the *port*s ifName.

##### 5.4.2.3 ifIndex

The **ifIndex** element is a new addition to the *Layer 2* namespace. This element can be used to specify the *SNMP* index of the interface. The ifIndex field **MUST** be filled in with the integer *SNMP* index for the interface.

##### 5.4.2.4 Vlan Tag

The **vlan** element is a new addition to the *Layer 2* namespace. This element can be used to specify the VLAN tag for the specified *port*. The VLAN element **MUST** contain an integer corresponding to the VLAN tag for that *port*.

##### 5.4.2.5 Vlan Range Availability

The **vlanRangeAvailability** element is a new addition to the *Layer 2* namespace. This element can be used to specify which tags are still available for use to create new VLANs on the specified *port*. The element **MUST** contain a string containing a comma-separated list of VLAN tag identifiers. Ranges of tags can be specified by listing two VLAN tags separated by a "-".

#### 5.4.3 Link

There are some minor additions to the *link* element that are discussed below.

##### 5.4.3.1 Vlan Tag

The **vlan** element is a new addition to the *Layer 2* namespace. This element can be used to specify the VLAN tag for the specified *link*. The VLAN element **MUST** contain an integer corresponding to the VLAN tag for that link. If the VLAN tag is specified for the port associated with this link, this value **MUST** be the same as the one in the port.

#### 5.4.4 Network

There are some minor additions to the *network* element that are discussed below.

##### 5.4.4.1 Vlan Tag

The **vlan** element is a new addition to the *Layer 2* namespace. This can be used to specify the VLAN tag for the specified *network*. The VLAN element **MUST** contain an integer corresponding to the VLAN tag for that network. If the VLAN tag is specified, all the elements referenced by the network need to have the same VLAN tag.

### 5.5 Layer 3 Elements

Building on the generic layer established by the elements in the "base" (see Section 5.1) it is possible to more robustly define specific extensions for various layers and technologies. We present the *Layer 3* extensions here that describe topology on the *Network Layer*.

#### 5.5.1 Type

There are no structural changes to this element from the "base". When used the type **SHOULD** reflect the *type*s that are available in the *Network Layer* (e.g. "IPv4", "IPv6").

#### 5.5.2 Port

There are some minor additions to the *port* element that are discussed below.

##### 5.5.2.1 Name

There are no structural changes to this element from the "base". The **name** for the *Layer 3* element **SHOULD** be the *Layer 3* address for that element. The type attribute **MUST** be specified.

##### 5.5.2.2 Address

There are no structural changes to this element from the "base". The **address** element in *Layer 3* ports **MUST** consist of the *Layer 3* address for the port.

##### 5.5.2.3 Netmask

The *Layer 3* port has an **OPTIONAL netmask** element. This element is a string that contains the netmask for the specified address. If the address type is *IPv4*, the contents **MUST** consist of a netmask in *dotted decimal* form. If the address type is *IPv6*, the contents **MUST** consist of an IPv6 netmask in *colon separated hexadecimal* format. If the address type is any other value, the contents of the netmask are unspecified.

#### 5.5.3 Link

There are some minor additions to the *link* element that are discussed below.

#### 5.5.3.1 Netmask

The *Layer 3* link has an **OPTIONAL netmask** element. This element is a string that contains the netmask for the specified address. If the address type is *IPv4*, the contents **MUST** consist of a netmask in *dotted decimal* form. If the address type is *IPv6*, the contents **MUST** consist of an IPv6 netmask in *colon separated hexadecimal* format. If the address type is any other value, the contents of the netmask are unspecified.

### 5.5.4 Network

There are some minor additions to the *network* element that are discussed below.

#### 5.5.4.1 Subnet

The *Layer 3* network element **SHOULD** contain a **subnet** element to describe the range of addresses contained in the network. This element **MUST** contain a *Layer 3 address* and *netmask* element. The *Layer 3* address element **MAY** contain any address in the range contained in the network. The contents of the netmask element depend on the type of the address. If the address type is IPv4, the netmask element **MUST** contain a netmask in *dotted decimal* form. If the address type is ipv6, the netmask element **MUST** contain a number corresponding to the IPv6 netmask.

#### 5.5.4.2 ASN

The *Layer 3* network element **SHOULD** contain an **ASN** element to specify which *autonomous system* the network is describing. The element **MUST** contain the autonomous system number as an integer.

## 5.6 Layer 4 Elements

Building on the generic layer established by the elements in the "base" (see Section 5.1) it is possible to more robustly define specific extensions for various layers and technologies. We present the *Layer 4* extensions here that describe topology on the *Transport Layer*.

### 5.6.1 Type

There are no structural changes to this element from the "base". When used the type **SHOULD** reflect the *type*s that are available in the *Network Layer* (e.g. "TCP", "UDP").

### 5.6.2 Port

There are some minor additions to the *port* element that are discussed below.

#### 5.6.2.1 Name

There are no structural changes to this element from the "base". The **name** element in *Layer 4 ports* **SHOULD** use the type "port". If so, the element **MUST** contain the number for that port. If a *Layer 4* element name has no *type* specified, the format of the element **SHOULD** be of the form "protocol:port". For example, a TCP service listening on port 80 would be described as "tcp:80".

#### 5.6.2.2 Address

There are no structural changes to this element from the "base". The format of the **address** element differs slightly for a *Layer 4* port. If no *type* is specified, the address **MUST** be of the form "IP:port". The IP corresponds to the underlying *Layer 4* address. If the underlying address is an IPv6 address, the convention is "[IP]:port". In this context, the IP **MUST** be the same as the interface underlying the *Layer 4* port.

#### 5.6.2.3 Port Number

The *Layer 4* port has a **REQUIRED** element **portNum**. This element consists of an integer that contains the port number that the protocol is listening on.

# 6 Topology Schema

The following sections show the formal schemata of the **NM-WG** using the elements described in Section 5.1 and concepts presented in this document. Each is written in the RELAX-NG[16] language. Through the use of tools such as Trang[18] and MSV[9] it is possible to convert this to other widely accepted formats such as XSD[19].

We are presenting three major components of the entire schemata:

- **Base** - A representation of the elements in Section 5.1
- **Layer 2** - An extesion to the "base" schema that describes the use of topology elements in "Layer 2" of the *OSI protocol model*.
- **Layer 3** - An extesion to the "base" schema that describes the use of topology elements in "Layer 3" of the *OSI protocol model*.
- **Layer 4** - An extesion to the "base" schema that describes the use of topology elements in "Layer 4" of the *OSI protocol model*.

# 7 Base Topology Schema

The "base" schema is so named because it contains the essential elements of this work. Subsequent extensions and profiles will incorporate the items presented below.

```
# Begin Schema

# ################################################################
#
# File:    nmtopo_base.rnc - Schema to describe network elements
# Version: $Id: nmtopo_base.rnc 347 2008-05-20 15:55:47Z aaron $
#
# ################################################################

# ################################################################
# Namespace definitions
# ################################################################
 default namespace nmtb =
   "http://ogf.org/schema/network/topology/base/20070707/"

# External schema files
 include "nmtypes.rnc"

# generic Topology can be a root element
 start |= Topology
```

```
 Topology = element topology {
     Domain*
     & BaseNode*
     & BaseLink*
     & BasePort*
     & BaseNetwork*
     & BasePath*
   }

 Domain = element domain {
     Identifier?
     & IdReference?
     & BaseNode*
     & BaseLink*
     & BaseNetwork*
     & BasePath*
   }

## ########################
## generic node
## ########################
 BaseNode = element node { BaseNodeContent }
 BaseNodeContent =
   Identifier?
   & IdReference?
   & Name*
   & Address*
   & Relation*
   & Lifetime?
   & element role { xsd:string }?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element location { LocationContent }?
   & element contact { ContactInformationContent }*
   & element comments { xsd:string }*
   & BasePort*
   & BaseService*

 ContactInformationContent =
   attribute priority { xsd:integer }?
   & element email { xsd:string }?
   & element phoneNumber { xsd:string }?
   & element administrator { xsd:string }?
   & element institution { xsd:string }?

 LocationContent =
   element continent { xsd:string }?
   & element country { xsd:string }?
   & element zipcode { xsd:integer }?
   & element state { xsd:string }?
   & element institution { xsd:string }?
   & element city { xsd:string }?
   & element streetAddress { xsd:string }?
   & element floor { xsd:string }?
   & element room { xsd:string }?
   & element cage { xsd:string }?
   & element rack { xsd:string }?
   & element shelf { xsd:string }?
   & element latitude { xsd:float }?
   & element longitude { xsd:float }?

## ########################
## generic port
## ########################
 BasePort = element port { BasePortContent }
 BasePortContent =
   Identifier?
   & IdReference?
   & Name*
   & Address*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element capacity { xsd:string }?
   & element mtu { xsd:string }?
   & element comments { xsd:string }*
   & BaseLink*

## ########################
## generic link
## ########################
 BaseLink = element link { BaseLinkContent }
 BaseLinkContent =
   Identifier?
   & IdReference?
   & Name*
```

```
   & Relation*
   & Lifetime?
   & element remoteLinkId { xsd:string }
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element globalName {
       attribute type { xsd:string }?
       & xsd:string
     }?
   & element comments { xsd:string }*

## ########################
## generic network
## ########################
 BaseNetwork = element network { BaseNetworkContent }
 BaseNetworkContent =
   Identifier?
   & IdReference?
   & NodeIdRef*
   & PortIdRef*
   & LinkIdRef*
   & Name*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*

## ########################
## generic path
## ########################
 BasePath =
   element path { BasePathContent }

# a path consists of a list of hops
 BasePathContent =
   Identifier
   & Relation*
   & Lifetime?
   & element comments { xsd:string }*
   & element hop { HopContent }*

 HopContent =
   Identifier,
   (
     DomainIdRef
     | NodeIdRef
     | PortIdRef
     | LinkIdRef
     | PathIdRef
     | NetworkIdRef
   )

## ########################
## generic endpoint pair
## ########################
 BaseEndPointPair =
   element endPointPair { BaseEndPointPairContent }

# an endpoint pair consists of two endpoints in the form of ports
 BaseEndPointPairContent =
   Name*
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & element src { (BasePortContent | PortIdRef) }
   & element dst { (BasePortContent | PortIdRef) }

## ########################
## generic service
## ########################
 BaseService =
   element service { BaseServiceContent }

 BaseServiceContent =
   Identifier?
   & IdReference?
   & Name*
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & Relation*
   & Lifetime?

# End Schema
```

## 7.1 Layer 2 Topology Schema

The “Layer 2” schema speficially models the “core” elements from the point of view of a *data link* network. The elements are the same as in Section 7, but are namespaced into the extension and contain additional elements relevant to this *layer*.

```
# Begin Schema

# ##############################################################
#
# File:    nmtopo_l2.rnc - Schema to describe L2 network elements
# Version: $Id: nmtopo_l2.rnc 347 2008-05-20 15:55:47Z aaron $
#
# ##############################################################

# ##############################################################
# Namespace definitions
# ##############################################################
 default namespace nmtl2 =
           "http://ogf.org/schema/network/topology/l2/20070707/"

# External schema files
 include "nmtypes.rnc"

## ########################
## layer 2 port
## ########################
 L2Port = element port { L2PortContent }
 L2PortContent =
   Identifier?
   & IdReference?
   & Name*
   & Address*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element capacity { xsd:string }?
   & element mtu { xsd:string }?
   & element comments { xsd:string }*
   & element ifName {xsd:string }?
   & element ifIndex {xsd:integer }?
   & element vlan {xsd:integer }?
   & element vlanRangeAvailability { xsd:string }
   & L2Link*

## ########################
## layer 2 link
## ########################
 L2Link = element link { L2LinkContent }
 L2LinkContent =
   Identifier?
   & IdReference?
   & Name*
   & Relation*
   & Lifetime?
   & element remoteLinkId { xsd:string }
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element globalName {
       attribute type { xsd:string }?
       & xsd:string
     }?
   & element comments { xsd:string }*
   & element vlan {xsd:integer }?

## ########################
## layer 2 network
## ########################
 L2Network = element network { L2NetworkContent }
 L2NetworkContent =
   Identifier?
   & IdReference?
   & NodeIdRef*
   & PortIdRef*
   & LinkIdRef*
   & Name*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & element vlan {xsd:integer }?
```

```
## ########################
## layer 2 endpoint pair
## ########################
 L2EndPointPair =
   element endPointPair { L2EndPointPairContent }

# an endpoint pair consists of two endpoints in the form of ports
 L2EndPointPairContent =
   Name*
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & element src { (L2PortContent | PortIdRef | Address) }
   & element dst { (L2PortContent | PortIdRef | Address) }

# End Schema
```

## 7.2 Layer 3 Topology Schema

The “Layer 3” schema speficially models the “core” elements from the point of view of the *network* layer. The elements are the same as in Section 7, but are namespaced into the extension and contain additional elements relevant to this *layer*.

```
# Begin Schema

# ##############################################################
#
# File:    nmtopo_l3.rnc - Schema to describe L3 network elements
# Version: $Id: nmtopo_l3.rnc 347 2008-05-20 15:55:47Z aaron $
#
# ##############################################################

# ##############################################################
# Namespace definitions
# ##############################################################
 default namespace nmtl3 =
           "http://ogf.org/schema/network/topology/l3/20070707/"

# External schema files
 include "nmtypes.rnc"

## ########################
## layer 3 port
## ########################
 L3Port = element port { L3PortContent }
 L3PortContent =
   Identifier?
   & IdReference?
   & Name*
   & Address*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element capacity { xsd:string }?
   & element mtu { xsd:string }?
   & element comments { xsd:string }*
   & element netmask { xsd:string }?
   & L3Link*

## ########################
## layer 3 link
## ########################
 L3Link = element link { L3LinkContent }
 L3LinkContent =
   Identifier?
   & IdReference?
   & Name*
   & Relation*
   & Lifetime?
   & element remoteLinkId { xsd:string }
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element globalName {
       attribute type { xsd:string }?
       & xsd:string
     }?
   & element comments { xsd:string }*
```

```
   & element netmask { xsd:string }?

## ########################
## layer 3 network
## ########################
 L3Network = element network { L3NetworkContent }
 L3NetworkContent =
   Identifier?
   & IdReference?
   & NodeIdRef*
   & PortIdRef*
   & LinkIdRef*
   & Name*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & element subnet {
        Address
        & element netmask { xsd:string }
     }?
   & element asn { xsd:integer }

## ########################
## layer 3 endpoint pair
## ########################
 L3EndPointPair =
   element endPointPair { L3EndPointPairContent }

# an endpoint pair consists of two endpoints in the form of ports
 L3EndPointPairContent =
   Name*
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & element src { (L3PortContent | PortIdRef | Address) }
   & element dst { (L3PortContent | PortIdRef | Address) }

# End Schema
```

## 7.3 Layer 4 Topology Schema

The “Layer 4” schema speficially models the “core” elements from the point of view of the *transport* layer. The elements are the same as in Section 7, but are namespaced into the extension and contain additional elements relevant to this *layer*.

```
# Begin Schema

# ################################################################
#
# File:    nmtopo_l4.rnc - Schema to describe L4 network elements
# Version: $Id: nmtopo_l4.rnc 347 2008-05-20 15:55:47Z aaron $
#
# ################################################################

# ################################################################
# Namespace definitions
# ################################################################
 default namespace nmtl4 =
           "http://ogf.org/schema/network/topology/l4/20070707/"

# External schema files
 include "nmtypes.rnc"

## ########################
## layer 4 port
## ########################
 L4Port = element port { L4PortContent }
 L4PortContent =
   Identifier?
   & IdReference?
   & Name*
   & Address*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element capacity { xsd:string }?
```

```
   & element mtu { xsd:string }?
   & element comments { xsd:string }*
   & element portNum { xsd:integer }?
   & L4Link*

## ########################
## layer 4 link
## ########################
 L4Link = element link { L4LinkContent }
 L4LinkContent =
   Identifier?
   & IdReference?
   & Name*
   & Relation*
   & Lifetime?
   & element remoteLinkId { xsd:string }
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element globalName {
       attribute type { xsd:string }?
       & xsd:string
     }?
   & element comments { xsd:string }*

## ########################
## layer 4 network
## ########################
 L4Network = element network { L4NetworkContent }
 L4NetworkContent =
   Identifier?
   & IdReference?
   & NodeIdRef*
   & PortIdRef*
   & LinkIdRef*
   & Name*
   & Relation*
   & Lifetime?
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*

## ########################
## layer 4 endpoint pair
## ########################
 L4EndPointPair =
   element endPointPair { L4EndPointPairContent }

# an endpoint pair consists of two endpoints in the form of ports
 L4EndPointPairContent =
   Name*
   & element type { xsd:string }?
   & element description { xsd:string }?
   & element comments { xsd:string }*
   & element src { (L4PortContent | PortIdRef | Address) }
   & element dst { (L4PortContent | PortIdRef | Address) }

# End Schema
```

# 8 Related Work

The network topology schemata has evolved from the work of several groups all striving for a common goal: *interchangeable and agnostic network element descriptions.* Throughout the design and implementation of the representation, several key design principals have remained despite the syntactic changes that **MAY** occur in the draft versions of each schema:

- **Description** of all **REQUIRED** network elements
- **Modular** design that allows for easy reuse and extension
- Ability to **relate** elements to each other
- Ability to **include** element definitions within each other

- Easy **identification** of elements to be used for *references* and *inclusion*

The design goals have carried the network topology representation from a simple schemata focused on a subset of what is possible to describe in a network, to a fully featured and extensible language that is currently used by several diverse parties and implementations. Every community that as adopted portions of this work has respected the above design principals despite the use of different dialects and versions of what is presented in this document. The following represents some of the projects that have contributed to the development of the schemata over time.

- **NM-WG**
- **perfSONAR**
- **Network Control Plane**

The contributions of these communities and caveats regarding use of the topology schema will be briefly examined in the following sections. A history of each will be accompanied by technical documentation of the similarities and differences with respect to this document.

## 8.1 NM-WG

The **NM-WG** has participated in several attempts to characterize network topology within the incarnations of the measurement schemata. Early **NM-WG** efforts, as demonstrated in [6], defined a hierarchy of network measurements and out of necessity started venturing into network topology specification. Each attempt built upon the successes and failures of the previous, but the concern of not having standard ways of describing network topology at either the measurement or supporting systems layer provided a huge obstacle to adoption, particular within groups that plan on re-using alternate and better defined topology definitions beyond what **MAY** be featured in network measurement data.

The "Version 2" schemata [17] is the latest standard of extensible network measurement encoding. A notable absence in this work is the lack of well defined "subject" elements in this base schemata, there are two reasons for this:

- The *base* document defines the elements that **MAY** be used, not **SHOULD** be used. Profile documents for specific measurement types are expected to recommend specific topology elements.
- The **NM-WG** decided some time ago to leave the definition of network elements up to more capable groups within the *OGF*.

All **NM-WG** derivative work includes either topology as defined by external groups (e.g. **DICE** affiliations, or **perfSONAR**) or non-sanctioned topology that is a holdover from older work. While the latter topology definitions **MAY** be un-official, their design is similar to current offerings in features but not in scaling and extensibility.

### 8.1.1 Schema Differences

The **NM-WG** originally offered two major topology elements for use in measurements:

- Interface
- Endpoint Pair

We will briefly explain each and offer some context regarding use.

#### 8.1.1.1 Interface

The interface element models a traditional communication interface connected to a network capable system. This model can be viewed as being a "Layer 3" interface (e.g. the "Network Layer" of communication), and would be traditional used on devices such as routers, switches, and end hosts. A simple example of an interface is shown below:

```
<interface>
  <ipAddress type="v4">192.168.0.1</ipAddress>
  <hostName>routerA.domain.net</hostName>
  <ifName>ge1/1/1</ifName>
  <ifDescription>gigE 1/1/1</ifDescription>
  <ifAddress type="ipv4">10.10.10.1</ifAddress>
  <ifIndex>0</ifIndex>
<interface>
```

The purpose of this element was to capture some identifying information (e.g. "metadata") about the interface that is useful in the context of a measurement and is able to uniquely identify the entity. There are several possible elements that store the necessary information.

##### 8.1.1.1.1 ipAddress

The **ipAddress** element describes an address that **MAY** be used to contact the *device* that contains this interface. This can be viewed as the *management* address instead of the specific address for the interface (see also Section 8.1.1.1.5). This element **MAY** contain a *type* attribute that is used to describe the particular type of address (e.g. **ipv4**, **ipv6**).

##### 8.1.1.1.2 hostName

The **hostName** element describes a hostname that **MAY** be used to contact the *device* that contains this interface. This can be viewed as the *management* hostname instead of specific information for the interface.

##### 8.1.1.1.3 ifName

The **ifName** element describes the specific name of the interface as reported by system software or monitoring tools (e.g. *SNMP*). There are no requirements on the value of this element.

##### 8.1.1.1.4 ifDescription

The **ifDescription** element is a human readable description of the element. This **MAY** be reported by the system software or monitoring tools, but is commonly supplemented with external information when applicable (e.g. describing point to point connections if available, etc.).

##### 8.1.1.1.5 ifAddress

The **ifAddress** element describes the address of a specific interface. See also Section 8.1.1.1.1 for the element that describes a management address.

##### 8.1.1.1.6 ifIndex

The **ifIndex** element describes the "index" or ordered position in the list of interfaces as described by the system software or monitoring tools. This element is normally viewed in conjunction with the *ifName* (see also Section 8.1.1.1.3).

#### 8.1.1.2 Endpoint Pair

The **Endpoint Pair** is a construct that is used to describe a measurement involving two parties, normally viewed as a *source* and a *destination*. This is a common element to use when describing measurements observed at "Layer 4" (e.g. the "Transport Layer" of communication) including bandwidth and latency observations between hosts. A simple example of the construct is shown below:

```
<endPointPair>
  <src type="hostname" value="hostA.domain.net" />
  <dst type="hostname" value="hostB.domain.net" />
</endPointPair>
```

The purpose of this element was to capture some identifying information (e.g. "metadata") about the two hosts that is useful in the context of a measurement and is able to uniquely identify the parties. There are several possible elements that store the necessary information.

##### 8.1.1.2.1 src

The **src** element describes the *source* of the measurement. This element has several potential elements including *value* and *type* which describe contact information and the specific data type this information **MAY** be (e.g. *ipv4* or *hostname* are examples).

##### 8.1.1.2.2 dst

The **dst** element describes the *destination* of the measurement. This element has several potential elements including *value* and *type* which describe contact information and the specific data type this information **MAY** be (e.g. *ipv4* or *hostname* are examples).

### 8.1.2 Schema

The **NM-WG** schema is presented below. Note that portions of the **perfSONAR** archtecture, see Section 8.2, also use this description.

```
# Begin Schema

# ################################################################
#
# File:      nmtopo.rnc - Schema to describe topological
#                         elements.
# Version:   $Id: nmtopo.rnc 341 2008-04-24 21:52:11Z boote $
#
# ################################################################


# ################################################################
# Namespace definitions
# ################################################################
namespace nmwgtopo = "http://ggf.org/ns/nmwg/topology/2.0/"


# ################################################################
# Covers the basic point to point measurement situation.  The two
# points are a source and destination; may contain information
# such as hostname or ip address, and port number when applicable.
#
# Example:
#
# <nmwgtopo:endPointPair
#  xmlns:nmwgtopo="http://ggf.org/ns/nmwg/topology/2.0/">
#
#   <nmwgtopo:src type="REQUIRED_TYPE" value="REQUIRED_VALUE"
#                 port="OPTIONAL_PORT"/>
#
#   <nmwgtopo:dst type="REQUIRED_TYPE" value="REQUIRED_VALUE"
```

```
#                 port="OPTIONAL_PORT"/>
#
# </nmwgtopo:endPointPair>
#
# ############################################################

EndpointPair =
        element nmwgtopo:endPointPair {
                EndpointPairContent
        }

EndpointPairContent =
        element nmwgtopo:src {
                EndpointContent
        } &
        element nmwgtopo:dst {
                EndpointContent
        }


# ############################################################
# Similar to above, from one point only.
#
# Example:
#
# <nmwgtopo:endPoint type="REQUIRED_TYPE" value="REQUIRED_VALUE"
#                 port="OPTIONAL_PORT"/>
#
# ############################################################

Endpoint =
        element nmwgtopo:endPoint {
                EndpointContent
        }

EndpointContent =
        (
                attribute value { xsd:string } |
                text
        ) &
        attribute type { xsd:string } &
        attribute port { xsd:string }?


# ############################################################
# When looking at network utilization numbers (from a router or
# related software) there is a different set of applicable
# information
#
# Example:
#
# <nmwgtopo:interface xmlns:nmwgtopo="http://ggf.org/ns/nmwg/topology/2.0/">
#
#   <nmwgtopo:ipAddress type='REQUIRED_TYPE'> TEXT </nmwgtopo:ipAddress>
#
#   <nmwgtopo:hostName> TEXT </nmwgtopo:hostName>
#
#   <nmwgtopo:ifName> TEXT </nmwgtopo:ifName>
#
#   <nmwgtopo:ifDescription> TEXT </nmwgtopo:ifDescription>
#
#   <nmwgtopo:ifAddress type='REQUIRED_TYPE'> TEXT </nmwgtopo:ifAddress>
#
#   <nmwgtopo:ifHostName> TEXT </nmwgtopo:ifHostName>
#
#   <nmwgtopo:ifIndex> TEXT </nmwgtopo:ifIndex>
#
#   <nmwgtopo:type> TEXT </nmwgtopo:type>
#
#   <nmwgtopo:direction> TEXT </nmwgtopo:direction>
#
#   <nmwgtopo:authRealm> TEXT </nmwgtopo:authRealm>
#
#   <nmwgtopo:classOfService> TEXT </nmwgtopo:classOfService>
#
#   <nmwgtopo:capacity> TEXT </nmwgtopo:capacity>
#
# </nmwgtopo:interface>
#
# ############################################################

Interface =
        element nmwgtopo:interface {
                InterfaceContent
        }

InterfaceContent =
        element nmwgtopo:ipAddress {
```

```
                Address
        }? &
        element nmwgtopo:hostName { xsd:string }? &
        element nmwgtopo:ifName { xsd:string }? &
        element nmwgtopo:ifDescription { xsd:string }? &
        element nmwgtopo:ifAddress {
                Address
        }? &
        element nmwgtopo:ifHostName { xsd:string }? &
        element nmwgtopo:ifIndex { xsd:string }? &
        element nmwgtopo:type { xsd:string }? &
        element nmwgtopo:direction { xsd:string }? &
        element nmwgtopo:authRealm { xsd:string }? &
        element nmwgtopo:classOfService { xsd:string }? &
        element nmwgtopo:capacity { xsd:string }?

Address =
        (
                attribute value { xsd:string } |
                text
        ) &
        attribute type { xsd:string }


# End Schema
```

### 8.1.3 Future Plans

The **NM-WG** will continue to work with external groups to produce measurement structures that use topology created and consumed by these groups respectively. The **NM-WG** will also work to deprecate the "legacy" topology definitions in favor of new topology work offered in part from descriptions in this document and that of other closely aligned working groups.

## 8.2 perfSONAR

The **perfSONAR** project [15] is engaged in the design of software and protocols to aid in the collection, storage, and exchange of network measurements. Beyond the use of the work done within the **NM-WG**, perfSONAR services are also interested in the construction and status of the underlying network that **MAY** be unavailable through simple network measurements. Services such as the *Topology Service* are specifically designed to collect and report the status of network topology. As such members of **perfSONAR** have provided feedback and expertise several **OGF** working groups to aide in the construction of portable network topology representations.

Since inception, the *perfSONAR* project has used schemata for measurements created by the **NM-WG**. Early work in *perfSONAR* was instrumental in driving the **NM-WG** to define some of the rudimentary topology elements discussed in Section 8.1. These elements and several others were necessary to begin classifying *perfSONAR* data and services. We will now present the topology elements used, and in some cases still in use, by *perfSONAR* services.

### 8.2.1 Schema Differences

In addition to the elements described in Section 8.1.1, there are other notable additions to the *perfSONAR* topology:

- Endpoint
- Node
- Router

- Link
- Network
- Path

#### 8.2.1.1 Endpoint

Similar in nature to the Endpoint Pair (see Section 8.1.1.2), this construct is used to describe a single "Layer 4" (e.g. the "Transport Layer" of communication) host. In practice this element was not used often, and was normally seen as an interchangeable element with *Node* (see Section 8.2.1.2) and *Router* (see Section 8.2.1.3). Common use case would be to services that **REQUIRED** a *subscription* such as a service that featured a "push" model of data deliver instead of a "pull". An example of this element is seen below:

```
<endPoint >
  <address  />
</endPoint>
```

##### 8.2.1.1.1 address

The **address** element describes the address of the *endPoint*. Certain implementations **MAY** substitute a *hostName* element here as well (see Section 8.1.1.1.2).

#### 8.2.1.2 Node

The **node** was a concept developed to describe a generic end system. The element itself featured many sub-elements that contained diverse information about the entity in question. This element was used primarily with the *link* construction (see Section 8.2.1.4). An example of the element follows:

```
<node>
  <role />
  <name />
  <type />
  <hostName />
  <description />
  <cpu />
  <operSys />
  <institution />
  <country />
  <city />
  <latitude />
  <longitude />
</node>
```

Many of these elements were seen as duplications of a general "location" concept that could be generalized and included in other topology elements. The following descriptions serve to motivate the old work only.

##### 8.2.1.2.1 role

The **role** element describes the job or assigned *role* the node **MAY** take on. The possible values for this are not defined at the schema level and instead specified by implementations using this construct.

##### 8.2.1.2.2 name

The **name** element describes the name, potentially global in scope, of the node. This element is normally used in conjunction with the *link* element to give context to point to point connections.

#### 8.2.1.2.3 type

The **type** element describes the kind of node in use. Examples include “demarcation point” or similar networking concepts. The possible values for this are not defined at the schema level and instead specified by implementations using this construct.

#### 8.2.1.2.4 hostName

The **hostName** element describes the *DNS* name of the node. See also Section 8.1.1.1.2.

#### 8.2.1.2.5 description

The **description** element is a human readable description of the node. This **SHOULD** be used to supplement the information provided by other elements that describe context of this entity.

#### 8.2.1.2.6 cpu

The **cpu** element describes information regarding the processing power of this node.

#### 8.2.1.2.7 operSys

The **operSys** element describes the installed system software of this node.

#### 8.2.1.2.8 institution

The **institution** element is used to describe the management of the node.

#### 8.2.1.2.9 country

The **country** element describes the physical location of the node.

#### 8.2.1.2.10 city

The **city** element describes the physical location of the node.

#### 8.2.1.2.11 latitude

The **latitude** element describes the physical location of the node taken from *GPS* information.

#### 8.2.1.2.12 longitude

The **longitude** element describes the physical location of the node taken from *GPS* information.

#### 8.2.1.3 Router

The concept of a *router* was developed specifically to describe services that offered *netflow* information. This node predates the *node* (see Section 8.2.1.2) concept but is used to describe a similar bit of information: specifics about a networked router or switch. This element has been marked for depreciation in the *perfSONAR* framework although several legacy services will continue to use and support the construct. An example of a router appears below:

```
<router>
  <name>routerA.domain.net</name>
  <description>Domain border router</description>
  <interface interfaceIdRef="if1" />
  <interface interfaceIdRef="if2" />
</router>
```

##### 8.2.1.3.1 name

The **name** element describes the *name* for the router or switch. It is unspecified in the schema if this **SHOULD** be a *DNS* style name or perhaps something less formal.

##### 8.2.1.3.2 description

The **description** element describes the human readable element that can be used to enter location or other descriptive information for the router.

##### 8.2.1.3.3 interface

The **interface** element describes interfaces of a particular router or switch. There **MAY** be any number if interfaces included in this element. See Section 8.1.1.1 for a definition.

#### 8.2.1.4 Link

The **link** construct was specifically designed to model the behavior of a point to point network connection between other entities such as *interfaces* or *nodes*. This entity was designed to be general and allows the included sub-elements to be included directly inside. An example link is pictured below:

```
<link>
  <index />
  <type />
  <globalName />
  <interface />
  <link />
  <node />
</link>
```

##### 8.2.1.4.1 index

The **index** element describes an arbitrary *index* that can be used to describe the "order" that a link **MAY** be in a larger list of links.

##### 8.2.1.4.2 type

The **type** element describes the *type* of link, possible values include *uni-directional*, or perhaps *inter-domain*. Exact values are not mandated by the schema.

#### 8.2.1.4.3 globalName

The **globalName** element describes a recognized *global* identification for the link.

#### 8.2.1.4.4 interface

The **interface** element describes interfaces that are attached to the link (normally 2). See Section 8.1.1.1 for a definition.

#### 8.2.1.4.5 link

The **link** element recursively defines links within a link.

#### 8.2.1.4.6 node

The **node** element describes nodes that are attached to the link. See Section 8.2.1.2 for a definition.

### 8.2.1.5 Network

The *network* element includes the same items as a *link* (see Section 8.2.1.4 for a definition) but tries to describe the concept of an entire network instead of a single connection. This construct is normally used to describe an entire domain's topology in one attempt.

```
<network>
  <name />
  <type />
  <interface />
  <link />
  <node />
</network>
```

#### 8.2.1.5.1 name

The **name** element describes the *name* for the network. It is unspecified in the schema but **MAY** be a *DNS* style name or perhaps something less formal.

#### 8.2.1.5.2 type

The **type** element describes the *type* of network, possible values include *ethernet*, or perhaps *sonnet*. Exact values are not mandated by the schema.

#### 8.2.1.5.3 interface

The **interface** element describes interfaces that are included in a network. See Section 8.1.1.1 for a definition.

#### 8.2.1.5.4 link

The **link** element describes links that are included in a network. See Section 8.2.1.4 for a definition.

#### 8.2.1.5.5 node

The **node** element describes nodes that are included in a network. See Section 8.2.1.2 for a definition.

### 8.2.1.6 Path

The **path** element is a series of *links* that describes a complete communication path. The *path* takes advantage of the *index* attribute in each link to order links when applicable.

```
  <path>
    <link />
    <link />
  </path>
```

#### 8.2.1.6.1 link

The **link** element describes links that are included in a path. See Section 8.2.1.4 for a definition.

## 8.2.2 Schema

The following schema was developed jointly between members of the **Control Plane** community, the **NM-WG**, and **perfSONAR** and features many of precursor ideas that are featured in this document. We first present the "base" schema, of which layers were constructed to further specify complex concepts.

```
# Begin Schema

# ############################################################
#
# File:         nmtopo_ver3.rnc - Schema to describe network elements
# Version:      $Id: nmtopo_ver3.rnc 189 2007-01-31 20:46:21Z boote $
# Purpose:      This file lays out some major network topologies
#               used in measurement.
# Reference:    http://books.xmlschemata.org/relaxng/page2.html
#
# ############################################################


# ############################################################
# Namespace definitions
# ############################################################
namespace nmwgtopo3 = "http://ggf.org/ns/nmwg/topology/base/3.0/"
namespace nmtl4 = "http://ggf.org/ns/nmwg/topology/l4/3.0/"
namespace nmtl3 = "http://ggf.org/ns/nmwg/topology/l3/3.0/"
namespace nmtl2 = "http://ggf.org/ns/nmwg/topology/l2/3.0/"


# ############################################################
# Include additional functionality from other files
# ############################################################
include "nmtopo-l4.rnc"
include "nmtopo-l3.rnc"
include "nmtopo-l2.rnc"


## ########################
## generic interface
## ########################

BaseInterface =
        element nmwgtopo3:interface {
                BaseInterfaceContent
        }

BaseInterfaceContent =
        Identifier? &
        BaseInterfaceIdRef? &
        BaseNodeIdRef? &
        BaseName? &
        element nmwgtopo3:type { xsd:string }? &
        element nmwgtopo3:hostName { xsd:string }? &
        element nmwgtopo3:ifName { xsd:string }? &
        element nmwgtopo3:ifDescription { xsd:string }? &
```

```
        element nmwgtopo3:ifIndex { xsd:string }? &
        element nmwgtopo3:capacity { xsd:string }?


## ########################
## generic link
## ########################

BaseLink =
        element nmwgtopo3:link {
                BaseLinkContent
        }

BaseLinkContent =
        Identifier? &
        BaseLinkIdRef? &
        BaseName? &
        element nmwgtopo3:index { xsd:string }? &
        element nmwgtopo3:type { xsd:string }? &
        element nmwgtopo3:globalName {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        (
                BaseInterface |
                L2Interface |
                L3Interface
        )* &
        (
                BaseLink |
                L2Link |
                L3Link
        )* &
        Node*


## ########################
## generic network
## ########################

BaseNetwork =
        element nmwgtopo3:network{
                BaseNetworkContent
        }

BaseNetworkContent =
        Identifier? &
        BaseNetworkIdRef? &
        BaseName? &
        element nmwgtopo3:type { xsd:string }? &
        (
                BaseInterface |
                L2Interface |
                L3Interface
        )* &
        (
                BaseLink |
                L2Link |
                L3Link
        )* &
        Node*


## ########################
## generic node
## ########################

Node =
        element nmwgtopo3:node {
                NodeContent
        }

NodeContent =
        Identifier? &
        BaseNodeIdRef? &
        BaseRole? &
        BaseName? &
        element nmwgtopo3:type { xsd:string }? &
        element nmwgtopo3:hostName { xsd:string }? &
        element nmwgtopo3:description { xsd:string }? &
        element nmwgtopo3:cpu { xsd:string }? &
        element nmwgtopo3:operSys { xsd:string }? &
        element nmwgtopo3:location {
                LocationContent
        }? &
        element nmwgtopo3:institution { xsd:string }? &
        element nmwgtopo3:country { xsd:string }? &
        element nmwgtopo3:city { xsd:string }? &
```

```
        element nmwgtopo3:latitude { xsd:float }? &
        element nmwgtopo3:longitude { xsd:float }? &
        (
                BaseInterface |
                L2Interface |
                L3Interface
        )*

LocationContent =
        element nmwgtopo3:institution { xsd:string }? &
        element nmwgtopo3:country { xsd:string }? &
        element nmwgtopo3:zipcode { xsd:integer }? &
        element nmwgtopo3:state { xsd:string }? &
        element nmwgtopo3:city { xsd:string }? &
        element nmwgtopo3:streetAddress { xsd:string }? &
        element nmwgtopo3:floor { xsd:string }? &
        element nmwgtopo3:room { xsd:string }? &
        element nmwgtopo3:cage { xsd:string }? &
        element nmwgtopo3:rack { xsd:string }? &
        element nmwgtopo3:shelf { xsd:string }? &
        element nmwgtopo3:latitude { xsd:float }? &
        element nmwgtopo3:longitude { xsd:float }?


## ########################
## generic path
## ########################

BasePath =
        element nmwgtopo3:path {
                BasePathContent
        }

BasePathContent =
        Identifier &
        BasePathIdRef? &
        (
                BaseLink |
                L2Link |
                L3Link
        )*


## ########################
## misc
## ########################

BaseAddress =
        (
                attribute value { xsd:string } |
                text
        ) &
        attribute type { xsd:string }

BaseRole =
        (
                attribute role { xsd:string } |
                element nmwgtopo3:role { xsd:string }
        )

BaseName =
        element nmwgtopo3:name {
                attribute type { xsd:string }? &
                xsd:string
        }

BaseNodeIdRef =
        attribute nodeIdRef { xsd:string }

BaseInterfaceIdRef =
        attribute interfaceIdRef { xsd:string }

BaseLinkIdRef =
        attribute linkIdRef { xsd:string }

BasePathIdRef =
        attribute pathIdRef { xsd:string }

BaseNetworkIdRef =
        attribute networkIdRef { xsd:string }

# End Schema
```

The second schema is specific to “Layer 4” connections:

```
# Begin Schema

# ############################################################
#
# File:         nmtopo_l4.rnc - Layer 4 Network entities
#
# Version:      $Id: nmtopo-l4.rnc 209 2007-02-15 17:20:25Z zurawski $
# Purpose:      The schema represents entities at the Layer 4 or
#               Transport layer
#
# Reference:    http://books.xmlschemata.org/relaxng/page2.html
#
# ############################################################


# ############################################################
# Namespace definitions
# ############################################################
namespace nmtl4 = "http://ggf.org/ns/nmwg/topology/l4/3.0/"
namespace nmtl3 = "http://ggf.org/ns/nmwg/topology/l3/3.0/"
namespace nmwgtopo3 = "http://ggf.org/ns/nmwg/topology/base/3.0/"


## ########################
## endPoint pair stuff
## ########################

L4EndpointPair =
        element nmtl4:endPointPair {
                L4EndpointPairContent
        }

L4EndpointPairContent =
        (
                element nmtl4:endPoint {
                        attribute port { xsd:string }? &
                        attribute protocol { xsd:string }? &
                        (
                                attribute role { "src" } |
                                element nmwgtopo3:role { "src" }
                        )? &
                        (
                                element nmtl4:address { L4Address } |
                                element nmtl3:interface { anyThing }
                        )?
                },
                element nmtl4:endPoint {
                        attribute port { xsd:string }? &
                        attribute protocol { xsd:string }? &
                        (
                                attribute role { "dst" } |
                                element nmwgtopo3:role { "dst" }
                        )? &
                        (
                                element nmtl4:address { L4Address } |
                                element nmtl3:interface { anyThing }
                        )
                }
        )


## ########################
## endPoint stuff
## ########################

L4Endpoint =
        element nmtl4:endPoint {
                L4EndpointContent
        }

L4EndpointContent =
        attribute port { xsd:string }? &
        attribute protocol { xsd:string }? &
        attribute index { xsd:string }? &
        L4Role? &
        (
                element nmtl4:address {
                        L4Address
                } |
                element nmtl3:interface {
                        anyThing
                }
        )


## ########################
## generic path
```

```
## ########################

Path =
        element nmtl4:path {
                PathContent
        }

PathContent =
        Identifier &
        PathIdRef? &
        L4Endpoint*


## ########################
## misc stuff
## ########################

L4Address =
        (
                attribute value { xsd:string } |
                text
        ) &
        attribute type { xsd:string }

L4Role =
        (
                attribute role { xsd:string } |
                element nmwgtopo3:role { xsd:string }
        )

PathIdRef =
        attribute pathIdRef { xsd:string }


# End Schema
```

The second schema is specific to “Layer 3” connections:

```
# Begin Schema

# ############################################################
#
# File:         nmtopo_l3.rnc - Schema for Layer 3 entities
# Version:      $Id: nmtopo-l3.rnc 189 2007-01-31 20:46:21Z boote $
# Purpose:      This schema represents entities at Layer 3, or the
#               Network Layer.
# Reference:    http://books.xmlschemata.org/relaxng/page2.html
#
# ############################################################


# ############################################################
# Namespace definitions
# ############################################################
namespace nmtl3 = "http://ggf.org/ns/nmwg/topology/l3/3.0/"
namespace nmtl2 = "http://ggf.org/ns/nmwg/topology/l2/3.0/"
namespace nmwgtopo3 = "http://ggf.org/ns/nmwg/topology/base/3.0/"


## ########################
## l3 interface
## ########################

L3Interface =
        element nmtl3:interface {
                L3InterfaceContent
        }

L3InterfaceContent =
        Identifier? &
        L3InterfaceIdRef? &
        element nmtl3:ipAddress {
                L3Address
        }? &
        element nmtl3:netmask { xsd:string }? &
        element nmtl3:ifName { xsd:string }? &
        element nmtl3:ifDescription { xsd:string }? &
        element nmtl3:ifAddress {
                L3Address
        }? &
        element nmtl3:ifHostName { xsd:string }? &
        element nmtl3:ifIndex { xsd:string }? &
        element nmtl3:type { xsd:string }? &
```

```
        element nmtl3:capacity { xsd:string }?

L3Address =
        (
                attribute value { xsd:string } |
                text
        ) &
        attribute type { xsd:string }


## ########################
## l3 link
## ########################

L3Link =
        element nmtl3:link {
                L3LinkContent
        }

L3LinkContent =
        Identifier? &
        L3LinkIdRef? &
        element nmtl3:index { xsd:string }? &
        element nmtl3:type { xsd:string }? &
        element nmtl3:name {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        element nmtl3:globalName {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        (
                L3Interface |
                L2Interface
        )* &
        (
                L3Link |
                L2Link
        )* &
        Node*


## ########################
## l3 network
## ########################

L3Network =
        element nmtl3:network{
                L3NetworkContent
        }

L3NetworkContent =
        Identifier? &
        L3NetworkIdRef? &
        element nmtl3:name {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        element nmtl3:type { xsd:string }? &
        element nmtl3:subnet { xsd:string }? &
        element nmtl3:netmask { xsd:string }? &
        element nmtl3:asn { xsd:string }? &
        (
                L3Interface |
                L2Interface
        )* &
        (
                L3Link |
                L2Link
        )* &
        Node*


## ########################
## l3 path
## ########################

L3Path =
        element nmtl3:path {
                L3PathContent
        }

L3PathContent =
        Identifier &
        L3PathIdRef? &
        (
                L2Link |
```

```
                L3Link
        )*


## ########################
## l3 misc
## ########################

L3Role =
        (
                attribute role { xsd:string } |
                element nmtl3:role { xsd:string }
        )

L3InterfaceIdRef =
            attribute interfaceIdRef { xsd:string }

L3LinkIdRef =
            attribute linkIdRef { xsd:string }

L3NetworkIdRef =
            attribute networkIdRef { xsd:string }

L3PathIdRef =
            attribute pathIdRef { xsd:string }


# End Schema
```

The second schema is specific to “Layer 2” connections:

```
# Begin Schema

# ############################################################
#
# File:         nmtopo_l2.rnc - Schema to describe topological
#                               features to be used in subject
#                               elements.
# Version:      $Id: nmtopo-l2.rnc 189 2007-01-31 20:46:21Z boote $
# Purpose:      This file lays out some major network topologies
#               used in measurement.
# Reference:    http://books.xmlschemata.org/relaxng/page2.html
#
# ############################################################


# ############################################################
# Namespace definitions
# ############################################################
namespace nmtl2 = "http://ggf.org/ns/nmwg/topology/l2/3.0/"
namespace nmwgtopo3 = "http://ggf.org/ns/nmwg/topology/base/3.0/"


## ########################
## l2 interface
## ########################

L2Interface =
        element nmtl2:interface {
                L2InterfaceContent
        }

L2InterfaceContent =
        Identifier? &
        L2InterfaceIdRef? &
        L2Role? &
        element nmtl2:type { xsd:string }? &
        element nmtl2:address {
                L2Address
        }? &
        element nmtl2:name {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        element nmtl2:description { xsd:string }? &
        element nmtl2:ifHostName { xsd:string }? &
        element nmtl2:ifIndex { xsd:string }? &
        element nmtl2:capacity { xsd:string }?

L2Address =
        (
                attribute value { xsd:string } |
                text
```

```
        ) &
        attribute type { xsd:string }


## ########################
## l2 link
## ########################

L2Link =
        element nmtl2:link {
                L2LinkContent
        }

L2LinkContent =
        Identifier? &
        L2LinkIdRef? &
        element nmtl2:index { xsd:string }? &
        element nmtl2:type { xsd:string }? &
        element nmtl2:name {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        element nmtl2:globalName {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        L2Interface* &
        L2Link* &
        Node*


## ########################
## l2 network
## ########################

L2Network =
        element nmtl2:network {
                L2NetworkContent
        }

L2NetworkContent =
        Identifier? &
        L2NetworkIdRef? &
        element nmtl2:name {
                attribute type { xsd:string }? &
                xsd:string
        }? &
        element nmtl2:type { xsd:string }? &
        element nmtl2:vlan { xsd:string }? &
        L2Interface* &
        L2Link* &
        Node*


## ########################
## l2 path
## ########################

L2Path =
        element nmtl2:path {
                L2PathContent
        }

L2PathContent =
        Identifier &
        L2PathIdRef? &
        L2Link*


## ########################
## l2 misc
## ########################

L2Role =
        (
                attribute role { xsd:string } |
                element nmtl2:role { xsd:string }
        )

L2InterfaceIdRef =
            attribute interfaceIdRef { xsd:string }

L2LinkIdRef =
            attribute linkIdRef { xsd:string }

L2NetworkIdRef =
            attribute networkIdRef { xsd:string }
```

```
L2PathIdRef =
           attribute pathIdRef { xsd:string }


# End Schema
```

#### 8.2.3 Future Plans

Members of the *perfSONAR* effort are active in the **NM-WG**, **NML-WG**, **NMC-WG**, and **NSI-WG**. It is anticipated that future iterations of *perfSONAR* will continue to adopt the recommendations of these groups when they become available and slowly depreciate the outdated constructs. Services that do not adapt will face the risk of being discontinued from the *perfSONAR* framework, but this step has not been necessary to date.

### 8.3 Network Control Plane

The *Network Control Plane* activities are driven by several development efforts including ESnet's **OSCARS** project [13], GÉANT2's **AutoBAHN** [1], and Internet2's **DCN** [4]. The groups are striving for protocol interoperability within the *DICE* [5] collaboration and are interested in a portable and fully featured definition of network topology. Early work was not aware of the *OGF* working group efforts and focused on designs that were precise, but not as scalable to other related concepts such as network measurement and monitoring.

The various groups have adopted topology that is similar to what the *perfSONAR* consortium had initially adopted and championed. Various design requirements have caused the flavor of topology to become more specific than was available at the time. The following section describes the current use of topology for these communities. Note that this community does use the concept of an identifier similar to what was described in Section 4 but with the specific elements described below.

#### 8.3.1 Schema Differences

The *Network Control Plane* uses elements similar to what was seen in Section 8.1.1 and Section 8.2.1, but minor modifications were **REQUIRED** to fit this particular enterprise. The following represent the basic elements in this schema:

- Topology
- Domain
- Node
- Port
- Link

##### 8.3.1.1 Topology

The **topology** element is used to enclose the definition of a *domain* as well as the specific information that an **IDC** (**I**nter **D**omain **C**ontroller) wishes to publish. Neighbors on the *control plane* share topology to perform signaling operations.

```
<topology>
  <idcId />
  <domain />
</topology>
```

#### 8.3.1.1.1 idcId

The **idcId** element contains identification information for the **IDC** (**I**nter **D**omain **C**ontroller) of a specific domain. This information contained **SHOULD** be unique.

#### 8.3.1.1.2 domain

The **domain** element contains a complete definition of network topology for a specific domain. See also Section 8.3.1.2.

### 8.3.1.2 Domain

The **domain** element contains a complete definition of all topological entities for a domain. These normally consist of *nodes*, *ports*, and *links*.

```
<domain>
  <node />
  <port />
  <link />
</domain>
```

#### 8.3.1.2.1 node

The **node** element describes network entities that contain *ports*. See also Section 8.3.1.3.

#### 8.3.1.2.2 port

The **port** element describes *interfaces* that connect *links*. See also Section 8.3.1.4.

#### 8.3.1.2.3 link

The **link** element directly connects *ports* that are contained in *nodes*. See also Section 8.3.1.5.

### 8.3.1.3 Node

The **node** element is designed to describe a network capable device such as an endpoint, router, or switch. This element normally contains a contact *address* and a definition of all available *ports*.

```
<node>
  <address />
  <port />
  <port />
</node>
```

#### 8.3.1.3.1 address

The **address** element contains contact information in the form of *DNS* hostnames or *IP* addresses.

#### 8.3.1.3.2 port

The **port** element describes a network *interface* for communicating on a specific node. See also Section 8.3.1.4.

#### 8.3.1.4 Port

The **port** element describes a network *interface* that provides plumbing support for the *nodes* of a *domain*. The port itself contains definitions on capacity as well as a list of *link* elements that connect the node to the outside world.

```
<port>
  <capacity />
  <maximumReservableCapacity />
  <minimumReservableCapacity />
  <granularity />
  <link />
  <link />
</port>
```

##### 8.3.1.4.1 capacity

The **capacity** element describes the maximum *capacity* of the port in bits per second.

##### 8.3.1.4.2 maximumReservableCapacity

The **maximumReservableCapacity** element describes the maximum amount of capacity that can be reserved for a port in bits per second. This **MUST** be less than or equal to the *capacity* (see Section 8.3.1.4.1).

##### 8.3.1.4.3 minimumReservableCapacity

The **minimumReservableCapacity** element describes the minimum amount of capacity that can be reserved for a port in bits per second. This **MUST** be less than or equal to the *maximumReservableCapacity* (see Section 8.3.1.4.2).

##### 8.3.1.4.4 granularity

The **granularity** element describes the minimum increment in which bandwidth can be reserved.

##### 8.3.1.4.5 link

The element

#### 8.3.1.5 Link

The **link** element describes a connection between ports. This element tries to describe the underlying technology in use as best as possible as the information is used by the *IDC* for communication purposes.

```
<link>
  <remoteLinkId />
  <trafficEngineeringMetric />
  <capacity />
  <maximumReservableCapacity />
  <minimumReservableCapacity />
  <granularity />
  <SwitchingCapabilityDescriptors>
    <switchingcapType />
    <encodingType />
    <switchingCapabilitySpecificInfo>
      <interfaceMTU  />
      <vlanRangeAvailability />
    </switchingCapabilitySpecificInfo>
  </SwitchingCapabilityDescriptors>
</link>
```

#### 8.3.1.5.1 remoteLinkId

The **remoteLinkId** element is the remote link to which the link is connected. Values **MAY** all be "*" if connected to an end-host or domain without a topology description.

#### 8.3.1.5.2 trafficEngineeringMetric

The **trafficEngineeringMetric** element is a metric associated with this link. Use is usually reserved for specific implementations and no schema guidance is provided for value.

#### 8.3.1.5.3 capacity

The **capacity** element describes the maximum *capacity* of the link in bits per second.

#### 8.3.1.5.4 maximumReservableCapacity

The **maximumReservableCapacity** element describes the maximum amount of capacity that can be reserved for a link in bits per second. This **MUST** be less than or equal to the *capacity* (see Section 8.3.1.4.1).

#### 8.3.1.5.5 minimumReservableCapacity

The **minimumReservableCapacity** element describes the minimum amount of capacity that can be reserved for a link in bits per second. This **MUST** be less than or equal to the *maximumReservableCapacity* (see Section 8.3.1.4.2).

#### 8.3.1.5.6 granularity

The **granularity** element describes the minimum increment in which bandwidth can be reserved.

#### 8.3.1.5.7 SwitchingCapabilityDescriptors

The **SwitchingCapabilityDescriptors** element describes the parents switching capability. There are several possible sub elements which we will briefly describe here:

- **switchingcapType** - Describes the "type" of switching available on a link (e.g. *ethernet*)
- **encodingType** - Describes the "type" of encoding for the link (e.g. *SONET*).
- **switchingCapabilitySpecificInfo** - Contains **interfaceMTU** and **vlanRangeAvailability**.
- **interfaceMTU** - The maximum transmission unit (**MTU**) of this link
- **vlanRangeAvailability** - The range of **VLAN**s available for provisioning on a link.

### 8.3.2 Schema

The **Control Plane** community constructed this schemata, with assistance from members of the **NM-WG** and **perfSONAR**.

```
# Begin Schema

# ##############################################################
#
# File:  nmtopo_ctrlplane.rnc
# Version: $Id: nmtopo_ctrlplane.rnc 378 2009-02-02 17:25:37Z aaron $
#
# ##############################################################

default namespace CtrlPlane =
  "http://ogf.org/schema/network/topology/ctrlPlane/20080828/"

include "nmtypes.rnc"

## Definition of the topology element
start |= element topology { CtrlPlaneTopologyContent }


CtrlPlaneTopologyContent =
 # & Parameters
  Identifier
  & element idcId { xsd:string }
  & (
    CtrlPlaneDomain |
    element domainSignature {
      CtrlPlaneDomainSignatureContent
    }
  )*

## this a placeholder until we discuss and experiment with signatures
CtrlPlaneDomainSignatureContent =
  attribute domainId { xsd:string }
  & anyElement

CtrlPlaneDomain =
  element domain {
    Identifier
    & Lifetime?
    & (
      CtrlPlaneNode*
      & CtrlPlanePort*
      & CtrlPlaneLink*
    )
  }

CtrlPlaneNode =
  element node {
    Identifier
    & Address?
    & CtrlPlanePort*
  }

CtrlPlanePort =
  element port {
    Identifier
    & CtrlPlaneCapacityContent
    & CtrlPlaneLink*
  }

CtrlPlaneLink =
  element link {
    Identifier
    & element remoteLinkId { xsd:string }?
    & element trafficEngineeringMetric { xsd:string }
    & CtrlPlaneCapacityContent?
    & element linkProtectionTypes { xsd:string }*
    & element administrativeGroups { CtrlPlaneAdministrativeGroup }*
    & element SwitchingCapabilityDescriptors { CtrlPlaneSwitchingCapabilityDescriptor+ }
  }

# Begin path and endpoint additions
CtrlPlanePath =
  element path {
    Identifier
    & Lifetime?
    & attribute direction { "upstream" | "downstream" }?
    & CtrlPlanePathContent
  }
```

```
# a path consists of a list of hops, and/or links
CtrlPlanePathContent =
  element hop { CtrlPlaneHopContent }*


CtrlPlaneHopContent =
  Identifier
  & (
    DomainIdRef |
    NodeIdRef |
    PortIdRef |
    LinkIdRef |
    CtrlPlaneDomain |
    CtrlPlaneNode |
    CtrlPlanePort |
    CtrlPlaneLink
  )
  & element nextHop { CtrlPlaneNextHopContent }*

CtrlPlaneNextHopContent =
  attribute weight { xsd:integer }?
  & attribute optional { "true" | "false" }?
  & xsd:string

CtrlPlaneBidirectionalPath =
  element bidirectionalPath { CtrlPlaneBidirectionalPathContent }

CtrlPlaneBidirectionalPathContent =
  Identifier
  & Lifetime?
  # We have to do this so that it can be validated using trang/jing and still
  # have the paths appear in an arbitrary order
  & (
        (CtrlPlaneDownstreamPathContent, CtrlPlaneUpstreamPathContent)
        | (CtrlPlaneUpstreamPathContent, CtrlPlaneDownstreamPathContent)
    )

CtrlPlaneDownstreamPathContent =
  element path {
    Identifier?
    & attribute direction { "downstream" }
    & CtrlPlanePathContent
  }

CtrlPlaneUpstreamPathContent =
  element path {
    Identifier?
    & attribute direction { "upstream" }
    & CtrlPlanePathContent
  }

# End path and endpoint

CtrlPlaneAdministrativeGroup =
  element group { xsd:int }
  & element groupID { xsd:string }?

CtrlPlaneSwitchingCapabilityDescriptor =
  element switchingcapType {
    "psc-1"
    | "psc-2"
    | "psc-3"
    | "psc-4"
    | "l2sc"
    | "tdm"
    | "lsc"
    | "fsc"
  }
  & element encodingType {
      "packet"
      | "ethernet"
      | "pdh"
      | "sdh/sonet"
      | "digital wrapper"
      | "lambda"
      | "fiber"
      | "fiberchannel"
      | xsd:string
    }
  & element switchingCapabilitySpecificInfo {
      CtrlPlaneSwitchingCapabilitySpecificInfo
    }+

CtrlPlaneSwitchingCapabilitySpecificInfo =
  CtrlPlaneSwitchingCapabilitySpecificInfo_psc
  | CtrlPlaneSwitchingCapabilitySpecificInfo_l2sc
  | CtrlPlaneSwitchingCapabilitySpecificInfo_tdm
  | CtrlPlaneSwitchingCapabilitySpecificInfo_lsc
```

```
  | CtrlPlaneSwitchingCapabilitySpecificInfo_fsc

CtrlPlaneSwitchingCapabilitySpecificInfo_psc =
  element capability { xsd:string }

CtrlPlaneSwitchingCapabilitySpecificInfo_tdm =
  element capability { xsd:string }

CtrlPlaneSwitchingCapabilitySpecificInfo_lsc =
  element capability { xsd:string }

CtrlPlaneSwitchingCapabilitySpecificInfo_fsc =
  element capability { xsd:string }

CtrlPlaneSwitchingCapabilitySpecificInfo_l2sc =
  element interfaceMTU { xsd:int }
  & element vlanRangeAvailability { xsd:string }
  & element suggestedVLANRange { xsd:string }?
  & element vlanTranslation { "true" | "false" }?

## Capacity Description Pattern
CtrlPlaneCapacityContent =
  element capacity { xsd:string }?
  & element maximumReservableCapacity { xsd:string }?
  & element minimumReservableCapacity { xsd:string }?
  & element granularity { xsd:string }?
  & element unreservedCapacity { xsd:string }?


# End Schema
```

### 8.3.3 Future Plans

The *Network Control Plane* will continue to use the current topology schema as represented above through the next several releases of software unless there is a compelling reason to shift to new work produced by the **NML-WG** or **NSI-WG**.

# 9 Notational Conventions

The key words “MUST” “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC 2119 [2]

# 10 Security Considerations

There are important security concerns associated with the generation and distribution of network topology information. We do not address those concerns in this document, but implementers are encouraged to consider the security implications of generating and distributing topology information. While distribution of end-to-end application-level topology is generally accepted, descriptions that identify individual components should be taken carefully, and the distribution of information to other sites that cannot be obtained readily by other users at those sites should be considered carefully.

# 11 Contributors

**Aaron Brown**
Internet2
1000 Oakbrook Drive

Suite 300
Ann Arbor MI 48104

**D. Martin Swany**
University of Delaware
Department of Computer and Information Sciences
Newark, DE 19716

**Jason Zurawski**
Internet2
1150 18th Street, NW
Suite 1020
Washington, DC 20036

## References

[1] AutoBAHN - Bandwidth on Demand. http://www.geant2.net/server/show/ConWebDoc.2544.

[2] S. Bradner. Key Words for Use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[3] DICE InterDomain Controller Protocol (IDCP). http://www.controlplane.net/.

[4] The Internet2 Dynamic Circuit Network (DCN). http://www.internet2.edu/network/dc/.

[5] DICE (DANTE-Internet2-CANARIE-ESnet) Collaboration. http://www.geant2.net/server/show/conWebDoc.1308.

[6] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. A Hierarchy of Network Performance Characteristics for Grid Applications and Services. Community practice, Global Grid Forum, June 2003. http://www.ogf.org/documents/GFD.23.pdf.

[7] A. Anjomshoaa M. Drescher. Standardised Namespaces for XML infosets in OGF. Open grid forum community document, Open Grid Forum, October 2006.

[8] R. Moats. URN Syntax. RFC 2141, May 1997.

[9] Sun Multi-Schema XML Validator (MSV). https://msv.dev.java.net/.

[10] Network Measurement Control Working Group (NMC-WG). https://forge.gridforum.org/projects/nmc-wg.

[11] Network Measurements Working Group (NM-WG). http://nmwg.internet2.edu.

[12] Network Service Interface Working Group (NSI-WG). https://forge.gridforum.org/projects/nsi-wg.

[13] ESnet On-demand Secure Circuits and Advance Reservation System (OSCARS). http://www.es.net/OSCARS/index.html.

[14] OSI Protocol Model. http://en.wikipedia.org/wiki/OSI_model.

[15] perfSONAR - Infrastructure for Network Performance Monitoring. http://www.perfsonar.net.

[16] RELAX-NG Schema Language. http://relaxng.org/.

[17] M. Swany. An Extensible Schema for Network Measurement and Performance Data. Open grid forum working group document, Open Grid Forum, February 2008.

[18] Multi-format schema converter based on RELAX NG. http://www.thaiopensource.com/relaxng/trang.html.

[19] XML Schema). http://www.w3.org/XML/Schema.