GWD-R
Network Measurements Working Group
https://forge.gridforum.org/projects/nm-wg

Martin Swany, UDel
May 27, 2009

An Extensible Schema for Network Measurement and Performance Data

Status of This Document

This document provides information to the Grid community regarding the design formats used in the storage and exchange of network measurements. Distribution is unlimited.

Copyright Notice

Contents

1 Introduction

This document presents an *extensible encoding standard* for **network measurement** and **performance data**. Uniform encoding of this class of information is a key problem for federated network management, and multi-domain dynamic provisioning of network circuits, as well as in advanced distributed computing environments such as the Grid.

This work is born of the need for a common mechanism for the *exchange* and *storage* of network measurement and performance data. In the case of research-oriented networks, parties often want to exchange network performance data with neighbors for debugging purposes. In general, however, there is no single monitoring system that is in use. In the Grid community, the need to exchange network metrics of various sorts is often highlighted. In short, it is highly desirable to have an extensible schema for network performance information that gives a common, general framework for representation and exchange.

This document builds on previous versions of the network measurement schemata. This document describes *Version 2* of the Grid Forum's Network Measurement Working Group (**NM-WG**) schema.

1.1 Goals

The goal is to define a *neutral* representation for network measurements that can be easily extended to support new types of data. This representation **SHOULD** identify *forms* of network performance data as well as to create standardized *mechanism* to both *describe* and *publish* these metrics.

2 Design Philosophy

One of the high-level goals of this representation is to *normalize* the data representation by removing as much redundancy as possible. The basic schema design is based on the observation that network measurement data can be divided into two major classes. The first class is the **Metadata**, which describes the *type* of measurement data, the *entity* or *entities* being measured and the particular *parameters* of the measurement. The second class is the **Data** itself, which is, at its simplest, a *timestamp* and a *value*, or *vector of values*. This division of Metadata and Data is present throughout the system. This structure is present both in the *Messages* sent between various data elements and in data *Stores* – persistent storage of XML documents representing system state.

3 Basic Elements

This schema defines the basic elements that can be used to represent performance data. The first distinction is between the *Metadata*, the relatively static information regarding the data, and *Data* itself, which generally changes over time. The key idea is that, for repeated measurements, which is a common case for performance data in networks and Grids, the Metadata *need not* be repeated with each measurement, saving space and effort.

Each top-level element in this schema has an Identifier attribute called **Id**. There are many cases in which one element needs to *reference* another. For these cases an *Id Reference* (**idRef**) is used. Note that we have not used the **XML Schema** *ID* and *IdRef* types, although they **MAY** be appropriate. The reason is that with an IdRef, the corresponding ID **MUST** appear in the same document. We envision remote Id references (with e.g., a generic URL or a WS-Addressing EPR) and thus find this limitation overly restrictive. (Note that another possibility is the use of a simple element to resolve the local ID and point to remote Metadata.)

3.1 Metadata

The Metadata **MUST** describe the Data *unambiguously*. To accomplish this, the Metadata **MUST** include key elements:

- **Subject** — The *Subject* identifies the entity being measured. This could include the network path between a pair of hosts, an interface on a router, or a specific location on the network from which flow or packet data is captured.
- **EventType** — The *EventType* identifies exactly what sort of measurement Event occurred.
- **Parameters** — The *Parameters* describe the details of the measurement.
- **Key** — The *Key* identifies a complete *Metadata* triplet (*Subject*, *EventType* and/or *Parameters*) that was previously retrieved.

3.1.1 Subject

The *Subject* identifies the measured entity. For networks, this **MAY** represent a path between two hosts or an interface on a network device. In most cases we rely on topological elements to fill in the subject of a measurement. Minimal topological elements have been defined as part of this base schema. However, more extensive definitions are currently being explored by other working groups including the Network Markup Language Working Group (**NML-WG**)[5].

3.1.2 EventType

The *EventType* is the canonical name of the aspect of the subject being measured, or the actual event (i.e. *characteristic*) being reported. The expected value of this element is a *URI string*, similar to a namespace (see Section 4 for information on namespaces).

Using a structured value, such a *URI*, allows greater control in defining the eventType. This format allows for behavior of an implementation to be more directly controlled (e.g. a specific *version* in the URI may indicate a change in operations for a data type; alternatively we may choose to reject data that is too "new" or too "old" based on the encoding).

3.1.3 Parameters

The *Parameters* describe the exact way in which a particular measurement was gathered. These can include parameters to active measurement tools. Essentially, anything needed to determine which measurements are fungible **SHOULD** be included here. Parameters take the form of name, value pairs stored in *Parameter* elements. The value of a parameter can itself be a complex XML element.

3.1.4 Parameter

The *Parameter* is a name and value pair used to describe a single aspect of the entire *Parameters* set of a measurement.

3.1.5 Key

The *Key* element holds the results of a previous query for a *Subject*, *EventType* and/or *Parameters* triplet. The key can be used to *replay* this query to have faster access to the underlying data set.

3.2 Data

The *Data* element has an identifier (**id**) and an identifier reference (**metadataIdRef**) that refers to the *Metadata* that describes it. It contains some number of *Datum* or *Key* elements.

3.2.1 Datum

The *Datum* elements hold the *timestamp* and *value* of the measurement or event. For many network measurement data sources, this can be a time-series of timestamp, value pairs. For other measurement types, the result value **MAY** be a vector.

3.3 Container Elements

The above-named elements are currently contained in two types of outer elements, *Message* and *Store*. They have exactly the same structure, i.e. containing *Metadata* and *Data* elements. Each **MAY** have an attribute called *type* to indicate its type. Each **MAY** also contain one *Parameters* element to indicate Message- or Store-level parameters and options.

3.3.1 Message

The *Message* element is meant to serve as a transient container for the transport of performance information. This element contains *Metadata* and *Data* pairs and could also contain a *Parameters* element.

3.3.2 Store

The *Store* element is a stationary container for the long term storage of performance information. This element contains *Metadata* and *Data* pairs and could also contain a *Parameters* element.

4 XML Namespaces

A key facet in this schema is the observation that any of the "core" elements can be used to describe any network measurement, but the exact content of the each **SHOULD** vary with the measurement type. We have adopted XML *namespaces* to allow reuse of these same elements, but to facilitate variation in the contents for each different type of data. In this way, some superficial examination of the structure of a message or information store can take place without looking at the details of the contents. Most processing functionality **SHOULD** be able to consume new data types with no modification.

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://ggf.org/ns/nmwg/base/2.0/*
- **Achievable Bandwidth - Base Extension**: *http://ggf.org/ns/nmwg/characteristics/bandwidth/acheiveable/2.0/*
- **Iperf - Base Extension**: *http://ggf.org/ns/nmwg/tools/iperf/2.0/*

We envision there being two major classes of namespace URIs. The first is a canonical name based on the *Hierarchy of Network Measurements* from this working group [2] . The second is based on an organization's domain name and allows for autonomous extension in much the same way as the Enterprise branch of the OID space [10] allows. Finally, as this specification does not address the embedding of this schema into other systems, we note that the relevant parts of the namespace can be appended to another namespace if one is already in use.

4.0.3 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 [3]. 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 that the **NM-WG** has adopted for the "base" namespace:

```
http://ggf.org/ns/nmwg/base/2.0/
```

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

- **Scheme/Domain** - http://ggf.org
- **Customs** - /ns
- **Project** - /nmwg
- **Part/Extension** - /base
- **Version** - /2.0

The first key difference between the **NM-WG** approach and the community document is q 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://ggf.org/ns"). This choice was initially arbitrary, and **SHOULD NOT** change the overall intention of the approaches. As early work was based before the *GGF* to *OGF* name change we have kept the legacy domain for historic reasons. We fully anticipate that new versions of the schema will adopt the proper domain name.

A second 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.

The final difference is the reliance on incremental version numbers (e.g. *2.0*) versus using the *date* (e.g. *20070707*). We feel this difference is superficial and **SHOULD** the community decide that using the date is more appropriate the switch is easily managed.

4.0.4 Namespace Expansion

The namespace-based approach alluded to in Section 4 provides extensibility by re-defining the "core" elements in a tool- or characteristic-specific namespace. To motivate the example presented in Section 8, we will briefly describe the procedure used to define a measurement in an expansion namespace. Note that a new namespace **SHOULD** be generated when encoding a form of measurement that cannot be adequately represented with the "base" namespace or any existing extensions.

4.0.4.1 Defining the Namespace

The namespace **SHOULD** be similar to the "base" namespace but feature a different **Part/Extension** and potentially a different **Version**. Alternatively the **Domain** and **Customs MAY** change if a group that is not affiliated with the *OGF* becomes involved in the definition process. The values for the **Part/Extension** will vary depending on if a new *characteristic* is being defined, or if data specific to a *tool* is being encoded. We will be defining the *Ping* tool which also has the characteristic of being a *Round Trip Delay* measurement. The following represent the proposed namespaces:

- **Round Trip Delay - Base Extension**: *http://ggf.org/ns/nmwg/characteristic/delay/roundTrip/2.0/*
- **Ping - Base Extension**: *http://ggf.org/ns/nmwg/tools/ping/2.0/*

For consistency we have kept the **Version** the same, this can be altered when desired. With the namespace carved out, we can move on to the next task of laying out the model of a *ping* measurement.

4.0.4.2 Modeling Information

It is paramount that the "core" elements be reused in schema extension. This implies that we **MAY** define new namespaces for select elements in the base schema, but **MAY** choose to keep several within the base; for some of the enclosing elements (e.g. *metadata*, *data*) this is **REQUIRED**. In general any element **MAY** be redefined but to allow for implementation flexibility it often makes sense to leave as many as possible within the base to avoid complexity in the XML parsing libraries. The following is a list of elements from "base" that will remain in the original namespace:

- **Message**
- **Store**
- **Metadata**
- **EventType**
- **Parameter**

- **Key**
- **Data**

These items **SHOULD** be re-purposed for the new measurement type into the new namespace:

- **Subject**
- **Parameters**
- **Datum**

The following sections will suggest modifications necessary to enable the *ping* datatype in the **tools** namespace.

4.0.4.2.1 Ping Subject

The *ping subject* **SHOULD** be specific to the "subject" of a *Ping* measurement. We learn through observing the tool that it involves two entities: a *source* and *destination* host. These hosts can be described as being "Layer 4" beings in the *OSI protocol model* [6].

A subject for this measurement **SHOULD** consist of a description of both of these hosts, preferably in an accepted format such as the topology descriptions being produced by the **NML-WG**. We **SHOULD** ensure that some basic information about each host is captured:

- Host IP Address
- Address Type (e.g. **IPv4**, **IPv6**)
- Hostname
- TCP Ports used

4.0.4.2.2 Ping Parameters

The *ping parameters* can loosely be translated as the parameters that were used in the running of the measurement tool. For example if there are options to vary the *number of packets sen*t or the *size of each packet*, this information **SHOULD** be encoded in the parameters definition. The following represents some good choices for parameters in the ping tool, this list is not exhaustive:

- Number of packets to send
- Interval between packets
- Size of each packet
- Time to live (*TTL*) of each packet

4.0.4.2.3 Ping Datum

The final behavior to model is the *ping datum*: the results of the measurement. The *Ping* tool reports the results of sending packet "probes" from the source to the destination in a sequence. The results in this case are a measurement of how long the operation tool as well as reporting the *sequence number* of a given probe and the individual *TTL* values of each. While not explicitly reported, we also desire the *time* that each measurement has either started or ended. The following represents the measurement data we wish to capture form the ping tool, this list is not exhaustive any **MAY** vary depending on specific instances of *Ping* in use:

- Measurement Time (round trip delay)
- Measurement units (e.g. *seconds*)
- "Wall clock" - when the measurement started or ended
- Packet TTL
- Ping probe sequence number (in relation to the ping measurement as whole).

4.0.4.3 Create Schema and Examples

Using the information presented in Section 4.0.4.2 we have produced a simple ping extension consisting of both a schema and example instance. These examples appear at the end of this document in Section 8.

5 Merge Metadata

While a complete Metadata block can be used to unambiguously describe a Data block, it is often desirable to combine multiple, partial Metadata blocks together. The main reason for this is reuse of information. Using the "metadataIdRef" attribute of a Metadata block allows us to form a "chain" of Metadata blocks.

This section presents the major uses of *merge chaining*; note that individual implementations **MAY** choose to strictly or loosely interpret these guidelines for the sake of performance or protection. The schema offers no specific guidance on these issues in favor of simply describing the structural composition of both the input data and the resulting output.

5.1 Merge Operation

As the name implies, we intend to "merge" or *combine* metadata elements through this structure. There are many things we **MAY** consider when describing this operation:

- Which elements are *mergeable*?
- How much *recursion* is needed for merge-able elements?
- When **SHOULD** we *duplicate* elements?
- When **SHOULD** we *replace* elements in the course of merging?

As stated previously, the schemata itself does not offer any suggestions as to what is a *good merge* vs. a *bad merge*. There are no rules regarding which *types* of data **SHOULD** and **SHOULD NOT** be merged. There is no guidance on when we **SHOULD** duplicate or replace elements.

We **RECOMMEND** some very simple and succinct guidelines to this operation. There **SHALL** always be exceptions to rules, therefore the reader is encouraged to think carefully about what **MAY** be needed when implementing this recommendation.

5.2 Mergeable Elements and Recursion

When performing the merge operation we **SHOULD** first look at the *top-level* elements; namely subject, eventType, and parameters. When faced with two metadata blocks to be merged, we only wish to combine:

- *Like* Elements (e.g. sharing the same localname)
- Elements in the same namespace
- Elements sharing the same (or "similar") eventType

When this first criteria is met, we **SHOULD** recurse downward and keep trying to merge until we reach the bottom of the structure. How far should we venture into the XML structure looking for similarities or differences? This question does not have a definite answer such as *stop at the grandchild of the current element*. While this **MAY** be frustrating, domain knowledge **MAY** help you make a passable decision especially with regards to topology based elements.

Like elements that do not share a common namespace or eventType **SHALL** require special rules that **MAY** differ between implementations. Depending on the level of protection or speed we wish to attain, these rules **MAY** vary.

5.3 Duplication, Augmentation, and Replacement

When are faced with *like* elements that **MAY NOT** share a common namespace or eventType, we **SHOULD NOT** combine. We **MAY** try to find the *least significant* namespace or eventType and work from there. Additionally we **MAY** run into items that are *exactly* the same (such as certain *parameters*, or *eventTypes*). In some cases we **SHOULD** take care to *add* all of these together to make duplicates; other cases **MAY** dictate total replacement.

As an example of extreme cases, consider taking a very safe approach to the combining of elements (i.e. not merging *like* elements with different namespaces). This approach will ensure that we protect the schema differences but **MAY** result in many more *wrong* answers. The converse is a very dangerous approach where we merge items that could be different on the inside. This **MAY** result in an approach similar to *I know what you meant* and could yield a more robust result data set.

5.4 Merge Examples

A simple example of merge chaining is to partially specify a metadata (leaving out perhaps one unspecified element) and then constructing new elements from this original. This example does not feature any *overwriting* of duplicate elements.

Take for example a physical *Layer 3* interface used to measure SNMP data. If we wanted to specify the two common *directions* (*in* and *out*) we could construct a chain similar to the below example.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="m1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s2">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </netutil:subject>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m3" metadataIdRef="m1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s3">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:direction>out</nmwgt:direction>
    </nmwgt:interface>
  </netutil:subject>
</nmwg:metadata>
```

Note that the merging is performed via the use of the *metadataIdRef* tag in the metadata element. This is a “red flag” used to indicate a chain should be resolved. The Figure 1 demonstrates the linking between the metadata elements. The resulting XML structure after merge chaining is also listed below.

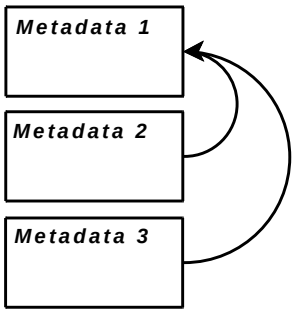

Figure 1: Graphical representation of merge chaining.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="m1">
```

```
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m3" metadataIdRef="m1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>out</nmwgt:direction>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

For continuity, this example has not attempted to modify the *metadataIdRef* attribute. Implementations **MAY** choose to do so if they feel the need. Because eventTypes **MAY** be repeated (either as the *eventType* element or as *parameters*) we **MUST** take special care when merging them. The next example features multiple eventType merge chaining. This example also features a so called *double merge chaining* where the results of the first merging operation **SHALL** feed into the process for the second.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="m1">
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m3" metadataIdRef="m2">
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

The resulting output and cartoon are pictured below. We did take two major issues into consideration: multiple *parameters* and *eventType* elements that did conflict, and the double merge chaining. Implemen-

tations that do not support multiple eventTypes (or simply wish to not implement a naive form of merge chaining) **SHOULD NOT** worry about special cases such as Figure 2.

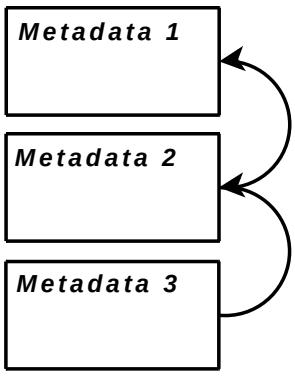

Figure 2: Alternate graphical representation of merge chaining.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m3" metadataIdRef="m2">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:parameter>
  </nmwg:parameters>
```

```
</nmwg:metadata>
```

Implementations **MAY** treat particular elements (such as eventTypes and parameters with certain *name* attributes) in a special way. The implementation is careful not to overwrite or lose any information and will only *add* these items together. This is not the case for all elements though, consider the following example.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifName>eth1</nmwgt:ifName>
      <nmwgt:direction>out</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
</nmwg:metadata>
```

Note that we *probably* wanted to change the *direction* for this particular interface, not necessarily the *ifName* element. The output of this chain is shown below.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>in</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth1</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:direction>out</nmwgt:direction>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

This example shows that it is very easy to introduce semantic errors when designing a chaining instance. It also shows that the implementation **MAY NOT** be interested in protecting a poorly designed chain from being accepted. It is possible to build in different rules instead of *last seen value* such as *first seen*,

original, or other combinations. It is imperative that implementations describe nuances of merge chaining, particularly when interoperability becomes an issue.

A final example comes when we deal with items with the same *localname*, but perhaps a different *namespace*. There are several approaches that can be taken to dealing with this type of situation. The SNMP example follows a safe approach of simply adding all of the elements in question and not attempting to internally merge at all. This causes *unreadable* metadata in many cases, but does not permit *data pollution*.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s2">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:direction>in</nmwgt:direction>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

There are three approaches that I will illustrate here: *safe yet stupid*, *dangerous yet intelligent*, and finally *slow and steady*. The last approach is sometimes used in practice; finding the proper balance will require some thought (depending on how sensing or accurate an implementation wishes to become. Approach one yields output similar to the below example.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s2">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:direction>in</nmwgt:direction>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

Note that this is not schema valid, and presumably would not return results from the backend storage. This is rather ironic given that we are trying to preserve validity on the schema side, yet still generate a clearly invalid result. The other end of the spectrum gives a result such as the example below.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s2">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:direction>in</nmwgt:direction>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

The so called “stupid” part of this comes from not caring about *namespaces*, and only merging based on *localname*. Because the source metadata featured the *netutil* namespace it remains and all other items are added to it.

The approach taken by some implementations is to have a little *domain* knowledge before making a quick judgement. Knowing full well that *nmwg* is a more general namespace than *netutil*, the implementation tries to guess the intent and goes with the most general namespace in order to support a richer query set. Internally anything that utilizes the *nmwg* namespace receives a wild card when performing searches. When we are faced with a choice between specific and general, the implementation errs on the side of general. An example of this merge is below.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="1">
  <nmwg:subject id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:direction>in</nmwgt:direction>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
```

```
  </nmwg:parameters>
</nmwg:metadata>
```

A final question remains: what happens if you are dealing with two very specific namespaces such as this example.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <neterr:subject xmlns:neterr="http://ggf.org/ns/nmwg/characteristic/errors/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
    </nmwgt:interface>
  </neterr:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s2">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:hostName>localhost</nmwgt:hostName>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

Some implementations will still guess “general” and convert to the *nmwg* namespace. The resulting data set will take on an interesting look:

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <neterr:subject xmlns:neterr="http://ggf.org/ns/nmwg/characteristic/errors/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
    </nmwgt:interface>
  </neterr:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2" metadataIdRef="1">
  <nmwg:subject id="s2">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
    </nmwgt:interface>
  </nmwg:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/errors/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>
```

Clearly the two eventTypes (for utilization and errors) **MAY NOT** appear in the same metadata description, but again the implementation can try to help out a bit. eventType descriptions are interpreted as *or* operations when performing a query. Therefore even if our chain was constructed poorly, our final results will be rather robust (perhaps a bit more robust than needed). The implementation designers will no doubt settle on an approach that fits well for the data they are exposing.

6 Operation Metadata

In addition to describing sets of raw data, Metadata blocks can also be used to describe *transformation operations* performed on a set of data. Thus, a list of Metadata blocks, including origin and transformations, can be used to *unambiguously* describe the provenance of any network data. This can also be used in cases where the transformation is *internal* and the original data is not available. This can be thought of as describing set operations on the original set as it passes through a list of operators. This operation has been termed "Operation" chaining.

This section presents the major uses of this operation; note that individual implementations **MAY** choose to strictly or loosely interpret these guidelines for the sake of performance or protection. The protocol itself offers no specific guidance on these issues in favor of simply describing the structural composition of both the input data and the resulting output.

6.1 Operation Chaining

Operation chaining involves the application of a *operator* (or function) to the underlying dataset that a particular metadata describes. We can think of this much like a database operation, where the first metadata is used to select a broad range of data, and subsequent metadata elements that are chained in this manner are used to slowly whittle down the dataset to a very specific range. Consider Figure 3 as an example of the internal process of resolving an operation chain.

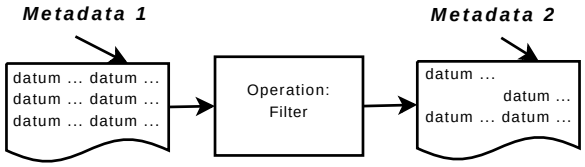

Figure 3: Graphical results of an operation step on a dataset.

An alternate use case is using the *operator* to perform a functional translation on *all* of the data contained in the first set when creating the second set. Figure 4 illustrates this "filter" operation by showing the contents of a data set before and after the operation. The first metadata is used to get the "base" dataset. After applying the second metadata are are left with the final data set: a subset of the first.

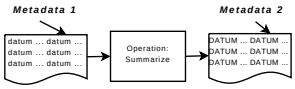

Figure 4: Graphical results of an operation step on a dataset.

It is important to note that even though we are manipulating the data through this form of chaining, we **SHOULD NOT** be harming it, or the related metadata elements. Chaining in general is a non-destructive operation, although it is very possible that when implemented poorly response data corruption **MAY** occur.

Operations can vary from time range selection to aggregations such as performing a cumulative distribution function (CDF). Describing all possible operators is well beyond the scope of this work. Current experience has named most statistical and database operations as candidates for this form of chaining, although new uses being devised.

6.1.1 Operator Chaining Examples

Operation chaining is an easier concept to manage than merge chaining, partially because there are less rules and nuances to grasp. As stated above, it is easy to think of the dataset for the source metadata to be *input* to a function that is named by the metadata utilizing the operation chain.

The syntax of operation chaining is similar to that of merge chaining (by using *metadataIdRef* attributes) but the placement is a bit different. Consider this example.

```
<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m1">
  <netutil:subject xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" id="s1">
    <nmwgt:interface xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/">
      <nmwgt:ifAddress type="ipv4">127.0.0.1</nmwgt:ifAddress>
      <nmwgt:hostName>localhost</nmwgt:hostName>
      <nmwgt:ifName>eth0</nmwgt:ifName>
      <nmwgt:ifIndex>2</nmwgt:ifIndex>
      <nmwgt:direction>in</nmwgt:direction>
      <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
  </netutil:subject>
  <nmwg:eventType>http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:eventType>
  <nmwg:eventType>http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:eventType>
  <nmwg:parameters id="p1">
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/tools/snmp/2.0</nmwg:parameter>
    <nmwg:parameter name="supportedEventType">http://ggf.org/ns/nmwg/characteristic/utilization/2.0</nmwg:parameter>
  </nmwg:parameters>
</nmwg:metadata>

<nmwg:metadata xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" id="m2">
  <select:subject id="s2" metadataIdRef="m1" xmlns:select="http://ggf.org/ns/nmwg/ops/select/2.0/"/>
  <select:parameters id="param2c" xmlns:select="http://ggf.org/ns/nmwg/ops/select/2.0/">
    <nmwg:parameter name="startTime">1121472000</nmwg:parameter>
    <nmwg:parameter name="endTime">1121904000</nmwg:parameter>
    <nmwg:parameter name="consolidationFunction">AVERAGE</nmwg:parameter>
    <nmwg:parameter name="resolution">60</nmwg:parameter>
  </select:parameters>
  <nmwg:eventType>http://ggf.org/ns/nmwg/ops/select/2.0</nmwg:eventType>
</nmwg:metadata>
```

The reference is placed in the *subject* element in this case, as in merge chaining this is a signal to the implementation that operation chaining **SHALL** be required. This indicates that the *input* is the data pointed to by the first metadata and the *output* will be a subset of this. For the sake of these examples we **SHALL** be dealing with the *select* namespace as our operator of choice due to an abundance of examples and its common goal of operation chaining based on time. Other operation examples **SHOULD** work in the same manner.

Because the operations of a operation chain are essentially *internal* we do not present what resultant XML should look like. Currently implementations ignore many of the steps that may go into reforming the XML for response messages in favor of simply returning the *backend* representation of metadata. While quick and easy, this does lead to information loss (specifically when dealing with the various ways to implement merge chaining). Client applications may have no reason to see the original operation information, and therefore are built not to need it.

7 Schema

The following sections show the formal schemata of the **NM-WG** using the elements described in Section 3 and concepts presented in this document. Each is written in the RELAX-NG[7] language. Through the use of tools such as Trang[8] and MSV[4] it is possible to convert this to other widely accepted formats such as XSD[9].

We are presenting three major components of the entire schemata:

- **Base** - A representation of the elements in Section 3
- **Time** - A simple schema that describes representation of time values in network measurements
- **Topology** - A basic representation of network topology; this work will be superseded by the efforts of more relevant working groups (such as the **NML-WG**)

7.1 Base 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:      nmbase.rnc - Main schema definition
# Version:   $Id: nmbase.rnc 341 2008-04-24 21:52:11Z boote $
# Purpose:   This is the main schema file, it defines the
#             general structure of an NMWG message or store
#
# ################################################################


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


# ################################################################
# Include additional functionality from other files
# ################################################################
include "nmtime.rnc"
include "filter.rnc"

# ################################################################
# Every NMWG document should begin with either a 'store' or
# 'message' element
# Patterns are defined for the content of each element.
#
# Example (using message):
#
# <nmwg:message id="OPTIONAL_ID"
#               messageIdRef="OPTIONAL_REFERENCE_ID"
#               type="REQUIRED_TYPE"
#               xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- OPTIONAL PARAMETERS -->
#
#   <!-- OPTIONAL (MULTIPLE) METADATA -->
#
#   <!-- OPTIONAL (MULTIPLE) DATA -->
#
# </nmwg:message>
#
# ################################################################

start =
        (
                element nmwg:message {
                        MessageContent
                } |
```

```
                element nmwg:store {
                        StoreContent
                }
        )

MessageContent =
        Identifier? &
        MessageIdentifierRef? &
        Type &
        Parameters? &
        (
                Metadata |
                Data
        )+


StoreContent =
        Identifier? &
        MessageIdentifierRef? &
        Type &
        Parameters? &
        (
                Metadata |
                Data
        )+


# ################################################################
# Metadata is the information that describes data.  This
# information doesn't change over time
#
#
# Example:
#
# <nmwg:metadata id="REQUIRED_ID"
#                metadataIdRef="OPTIONAL_REFERENCE_ID"
#                xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- TBD OPTIONAL SUBJECT -->
#
#   <!-- TBD OPTIONAL PARAMETERS -->
#
#   <!-- TBD OPTIONAL EVENTTYPE -->
#
#   <!-- TBD OPTIONAL KEY -->
#
#   <!-- ANY OPTIONAL (MULTIPLE) ELEMENT IN ANY NAMESPACE -->
#
# </nmwg:metadata>
#
# ################################################################

Metadata =
        element nmwg:metadata {
                (
                        Identifier &
                        MetadataIdentifierRef? &
                        MetadataContent
                ),
                anyElement*
        }

MetadataBlock =
        Subject? &
        Parameters?

MetadataContent =
        (
                MetadataBlock |
                FilterMetadataBlock
        ) &
        EventType? &
        Key?


# ################################################################
# Subject identifies an endPoint (or points), perhaps the name of
# a service or some other form of physical location. For the
# purpose of the general case, we make no assumptions on potential
# elements and allow all elements, in any namespace. Verification
# can be handled in subsequent schema files.
#
# Example:
#
# <nmwg:subject id="REQUIRED_ID"
#               metadataIdRef="OPTIONAL_REFERENCE_ID"
#               xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
```

```
#   <!-- ANY ELEMENT IN ANY NAMESPACE -->
#
# </nmwg:subject>
#
# ################################################################

Subject =
        element nmwg:subject {
                SubjectContent
        }

SubjectContent =
        (
                Identifier &
                MetadataIdentifierRef?
        ),
        anyElement*


# ################################################################
# Parameters and Parameter elements can be used in a number of
# ways in: 1) the message to signify items such as time stamp
# or authorization or 2) metadata or data to specify filters or
# special cases for the information. A 'parameters' block
# has an id and encloses one to many 'parameter' elements.
# These elements have a required 'name', and may contain
# an attribute, element, or text value (only one please;
# software using this should consider complex elements, then
# text, and finally the value attribute; exceptions should
# be thrown on duplicates).
#
# Example:
#
# <nmwg:parameters id="REQUIRED_ID"
#                  xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <nmwg:parameter name="REQUIRED_NAME" value="OPTIONAL_VALUE"
#                   xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#     <!-- ANY TEXT, OR ANY ELEMENT ANY NAMESPACE (IF YOU DID NOT
#          USE THE VALUE ATTRIBUTE) -->
#
#   </nmwg:parameter>
#
#   <!-- MORE PARAMETERS -->
#
# </nmwg:parameters>
#
# The namespaces can of course be different.
#
# ################################################################

Parameters =
        element nmwg:parameters {
                ParametersContent
        }

ParametersContent =
        Identifier &
        Parameter+

Parameter =
        element nmwg:parameter {
                attribute name { xsd:string } &
                (
                        attribute value { xsd:string } |
                        (
                                anyElement |
                                text
                        )
                )
        }


# ################################################################
# Event type is a simple text element used to describe the
# characteristic or event of the data.
#
# Example:
#
# <nmwg:eventType xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- TEXT -->
#
# </nmwg:eventType>
#
# ################################################################
```

```
EventType =
        element nmwg:eventType { xsd:string }


# ###############################################################
# The key is used to return a 'pointer' or otherwise special piece
# of identifying information in response to a request. For now,
# this information is enclosed only within a parameters block.
# The optional ID can be used to track past searches.
#
# Example:
#
# <nmwg:key id="OPTIONAL_ID"
#           xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- OPTIONAL PARAMETERS -->
#
# </nmwg:key>
#
# ###############################################################

Key =
        element nmwg:key {
                Identifier? &
                (
                        Parameters |
                        FilterParameters
                )
        }


# ###############################################################
# The data block is complex and has the potential to contain
# many things. The data block can be used to return a metadata
# block from a request, commonTime or datum elements, keys,
# or something that we have perhaps not defined as of yet.
#
# Example:
#
# <nmwg:data id="REQUIRED_ID"
#            metadataIdRef="OPTIONAL_REFERENCE_ID"
#            xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- OPTIONAL (MULTIPLE) METADATA -->
#
#           <!-- OR -->
#
#   <!-- TBD OPTIONAL (MULTIPLE) COMMON TIME ELEMENTS AND
#        OPTIONAL (MULTIPLE) DATUM ELEMENTS-->
#
#           <!-- OR -->
#
#   <!-- TBD OPTIONAL (MULTIPLE) DATUM ELEMENTS -->
#
#           <!-- OR -->
#
#   <!-- OPTIONAL (MULTIPLE) KEY ELEMENTS -->
#
#           <!-- OR -->
#
#   <!-- ANY OPTIONAL (MULTIPLE) ELEMENT IN ANY NAMESPACE -->
#
# </nmwg:data>
#
# ###############################################################

Data =
        element nmwg:data {
                (
                        Identifier &
                        MetadataIdentifierRef? &
                        (
                                Metadata* |
                                (
                                        commonTime+ &
                                        Datum*
                                ) |
                                Datum* |
                                Key*
                        )
                ),
                anyElement*
        }

# ###############################################################
# CommonTime is used as a shortcut that is able to 'factor out'
# a frequently occurring time range that a group of datum (or
# other) elements might share, thus reducing the verbosity of the
```

```
# XML representation. CommonTime is similar to the other NMWG time
# stamps (from nmtime.rnc) in its potential time representations.
#
# It is unfortunate that it needs to be in this file and not
# nmtime.rnc, but as it occurs outside the datum, it is here.
#
# Example:
#
# <nmwg:commonTime type="REQUIRED_TYPE" value="OPTIONAL_VALUE"
#                  duration="OPTIONAL_DURATION"
#                  inclusive="OPTIONAL_INCLUSIVE_FLAG"
#                  xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- TBD OPTIONAL START TIME ELEMENT (USE END TIME OR
#        DURATION) -->
#
#   <!-- TBD OPTIONAL END TIME ELEMENT (ONLY WITH START TIME) -->
#
#   <!-- TBD OPTIONAL TIME VALUE ELEMENT (USE IF NO VALUE
#        ATTRIBUTE) -->
#
#   <!-- TBD OPTIONAL (MULTIPLE) DATUM ELEMENTS -->
#
#   <!-- ANY OPTIONAL (MULTIPLE) ELEMENT IN ANY NAMESPACE -->
# </nmwg:commonTime>
#
# ###############################################################

commonTime =
        element nmwg:commonTime {
                (
                        Type &
                        (
                                TimeStamp |
                                (
                                        StartTime &
                                        (
                                                EndTime |
                                                Duration
                                        )
                                )
                        ) &
                        Datum*
                ),
                anyElement*
        }


# ###############################################################
# The datum is meant to be generic in this case because specific
# namespace declarations should be used to better define what
# format that datum should have.
#
# Example:
#
# <nmwg:datum ANY_ATTRIBUTE
#             xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- ANY ELEMENT IN ANY NAMESPACE OR ANY TEXT -->
#
# </nmwg:datum>
#
# ###############################################################

Datum =
        element nmwg:datum {
                anyThing
        }


# ###############################################################
# Common elements are defined as named patterns as they are re-
# used several times.
# ###############################################################

Identifier =
        attribute id { xsd:string }

MetadataIdentifierRef =
        attribute metadataIdRef { xsd:string }

MessageIdentifierRef =
        attribute messageIdRef { xsd:string }

Type =
        attribute type { xsd:string }
```

```
# ##############################################################
# This sequence allows any element, attribute, or text (regardless
# of name or namespace) into the document when invoked.
# ##############################################################

anyElement =
        element * {
                anyThing
        }

anyAttribute =
        attribute * { text }

anyThing =
        (
                anyElement |
                anyAttribute |
                text
        )*


# ##############################################################
# This sequence allows any element, attribute, or text (only in the
# NMWG namespace) into the document when invoked.
# ##############################################################

anyNMWGElement =
        element nmwg:* {
                anyNMWGThing
        }

anyNMWGAttribute =
        attribute * { text }

anyNMWGThing =
        (
                anyNMWGElement |
                anyNMWGAttribute |
                text
        )*

# End Schema
```

7.2 Time Schema

The time schema features a simple schema capable of representing time in the data portion of **NM-WG** encoded information. This schema does not a complex time definition, and may fall short of what is required for many high performance measurement tools. The members of this group fully expect this work to be augmented and eventually replaced by stronger representations that will come from members of the network measurement community.

```
# Begin Schema

# ##############################################################
#
# File:      nmtime.rnc - NMWG Time definitions
# Version:   $Id: nmtime.rnc 358 2008-06-05 15:14:11Z swany $
# Purpose:   This describes a set of time formats for
#             representing measurements.
#
# ##############################################################


# ##############################################################
# Namespace definitions
# ##############################################################
namespace nmtm = "http://ggf.org/ns/nmwg/time/20070914/"


# ##############################################################
# Regular time is attached to a specific datum instance; it is
# essentially the same as before, but cannot have anything
# 'inside' of it. The type can be simple, like UNIX, or it
# could be something like timeRange or timeInterval. If this is
# the case, we would then see the two extra time designators for
# the start and end (or duration)
#
```

```
# Example:
#
# <nmtm:time type="REQUIRED_TYPE" value="OPTIONAL_VALUE"
#            duration="OPTIONAL_DURATION"
#            inclusive="OPTIONAL_INCLUSIVE_FLAG"
#            xmlns:nmtm="http://ggf.org/ns/nmwg/time/2.0/">
#
#   <!-- TBD OPTIONAL START TIME ELEMENT (USE END TIME OR
#        DURATION) -->
#
#   <!-- TBD OPTIONAL END TIME ELEMENT (ONLY WITH START TIME) -->
#
#   <!-- TBD OPTIONAL TIME VALUE ELEMENT (USE IF NO VALUE
#        ATTRIBUTE) -->
#
# </nmtm:time>
#
#
# Time types are enumerated as follows:
#
# * unix: integral seconds since Jan 1, 1970 (UTC)
#
# * iso9601/rfc3339:
#
#       full date/time representation. Examples from RFC-339:
#
#        Here are some examples of Internet date/time format.
#
#           1985-04-12T23:20:50.52Z
#
#        This represents 20 minutes and 50.52 seconds after the 23rd hour of
#        April 12th, 1985 in UTC.
#
#           1996-12-19T16:39:57-08:00
#
#        This represents 39 minutes and 57 seconds after the 16th hour of
#        December 19th, 1996 with an offset of -08:00 from UTC (Pacific
#        Standard Time).  Note that this is equivalent to 1996-12-20T00:39:57Z
#        in UTC.
#
#           1990-12-31T23:59:60Z
#
#        This represents the leap second inserted at the end of 1990.
#
#           1990-12-31T15:59:60-08:00
#
#        This represents the same leap second in Pacific Standard Time, 8
#        hours behind UTC.
#
#           1937-01-01T12:00:27.87+00:20
#
#        This represents the same instant of time as noon, January 1, 1937,
#        Netherlands time.  Standard time in the Netherlands was exactly 19
#        minutes and 32.13 seconds ahead of UTC by law from 1909-05-01 through
#        1937-06-30.  This time zone cannot be represented exactly using the
#        HH:MM format, and this timestamp uses the closest representable UTC
#        offset.
#
# ################################################################

Time =
        element nmtm:time {
                attribute type { xsd:string } &
                (
                        TimeStamp |
                        (
                               StartTime &
                               (
                                       EndTime |
                                       Duration
                               )
                        )
                )
        }

# precisionUnits must be one of the recognized SI units
# The most complete list I could currently find is at:
# http://en.wikipedia.org/wiki/Orders_of_magnitude_%28time%29
#
# synchronized SHOULD be set true if the party generating the timestamp
# has a clock that is synchronized to UTC using an external source
# (e.g., the attribute should be set true if GPS hardware is used and it
# indicates that it has acquired current position and time or if NTP is
# used and it indicates that it has synchronized to an external source,
# which includes stratum 0 source, etc.).  If there is no notion of
# external synchronization for the time source, the attribute SHOULD be set
# to false. If the attribute is not set at all, the synchronization
# status of the timestamp can not be determined except through external
```

```
# knowledge.
Precision =
        (
            (
                attribute precision { xsd:unsignedInt } &
                attribute precisionUnits { xsd:string } &
                attribute synchronized { xsd:boolean }?
            ) |
            element nmtm:precision {
                attribute precisionUnits { xsd:string } &
                attribute synchronized { xsd:boolean }? &
                xsd:unsignedInt
            }
        )


TimeStamp =
        (
                (
                        attribute value { xsd:string } |
                        element nmtm:value { xsd:string }
                ) &
                Precision?
        )

Duration =
        attribute duration { xsd:string }

TimeContent =
        attribute type { text } &
        attribute inclusive { text }? &
        TimeStamp

StartTime =
        element nmtm:start {
                TimeContent
        }

EndTime =
        element nmtm:end {
                TimeContent
        }

# End Schema
```

7.3 Topology Schema

The topology schema contains elements that define simple aspects of network topology. The elements in this work are by no means exhaustive and cover only common measurement cases such as measurements originating from the 3rd and 4th "Layers" of the *OSI protocol model*.

As noted above there has been significant work in the **NML-WG** to define network topology that is both scalable and sharable; the **NM-WG** expects to depreciate this topology schema in time when there concrete standards published. A migration plan and information document from the **NM-WG** is fully expected in this time of transition.

```
# 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 Examples

This section includes examples of network measurements rendered in our schema. These examples are not intended to be normative, although at this time of this writing, they are in use.

8.1 Schema for ping

```
# Begin Schema

# ############################################################
#
# File:         ping.rnc - Specialized schema for the ping
#                          tool
# Version:      $Id: ping.rnc 341 2008-04-24 21:52:11Z boote $
# Purpose:      Defines elements to be used in the representation
#               of ping measurements.
#
# ############################################################


# ############################################################
# Namespace definitions
# ############################################################
namespace nmwg = "http://ggf.org/ns/nmwg/base/2.0/"
namespace ping = "http://ggf.org/ns/nmwg/tools/ping/2.0/"
namespace nmwgr = "http://ggf.org/ns/nmwg/result/2.0/"


# ############################################################
# Include additional functionality from other files
# ############################################################
include "nmtopo.rnc"
include "nmtopo_ver3.rnc"
include "result.rnc"
include "nmbase.rnc" {
        Metadata |= PingMetadata
        Data |= PingData
}
```

```
# ############################################################
# Metadata
# ############################################################

PingMetadata =
        element nmwg:metadata {
                Identifier &
                MetadataIdentifierRef? &
                PingMetadataContent
        }

PingMetadataBlock =
        PingSubject? &
        (
                PingParameters |
                Parameters
        )?

PingMetadataContent =
        (
                PingMetadataBlock |
                FilterMetadataBlock
        ) &
        EventType? &
        Key?


# ############################################################
# Redefined ping subject allows only an endPointPair, and the
# two id attributes.
#
# Example:
#
# <ping:subject id="REQUIRED_ID"
#               metadataIdRef="OPTIONAL_REFERENCE_ID"
#               xmlns:nmwg="http://ggf.org/ns/nmwg/tools/ping/2.0/">
#
#   <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>
#
# </ping:subject>
#
# ############################################################

PingSubject =
        element ping:subject {
                Identifier &
                MetadataIdentifierRef? &
                (
                        EndpointPair |
                        L4EndpointPair
                )
        }


# ############################################################
# This is simply the regular method of doing parameters with an
# enumeration to limit what 'names' are accepted and an outer
# ping: namespace for the parameters.
#
# Example:
#
# <ping:parameters id="REQUIRED_ID"
#                  xmlns:nmwg="http://ggf.org/ns/nmwg/tools/ping/2.0/">
#
#   <nmwg:parameter name="REQUIRED_ENUM_NAME" value="OPTIONAL_VALUE"
#                   xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#     <!-- ANY TEXT, (IF YOU DID NOT USE THE VALUE ATTRIBUTE) -->
#
#   </nmwg:parameter>
#
#   <!-- MORE PARAMETERS -->
#
# </ping:parameters>
#
# ############################################################

PingParameters =
        element ping:parameters {
                Identifier &
```

```
                PingParameter+
        }

PingParameter =
        element nmwg:parameter {
                attribute name { "count" | "interval" | "deadline" |
                                 "packetSize" | "ttl" | "arguments" |
                                 "valueUnits" | "numBytes" |
                                 "numBytesUnits" } &
                (
                        attribute value { text } |
                        text
                )
        }


# ###############################################################
# The data block is complex, and has the potential to contain
# many things.  The data block can be used to return a metadata
# block from a request, commonTime or datum elements, keys,
# or something that we have perhaps not defined as of yet.
#
# Example:
#
# <nmwg:data id="REQUIRED_ID"
#            metadataIdRef="OPTIONAL_REFERENCE_ID"
#            xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
#
#   <!-- OPTIONAL (MULTIPLE) METADATA -->
#
#          <!-- OR -->
#
#   <!-- TBD OPTIONAL (MULTIPLE) COMMON TIME ELEMENTS AND
#        OPTIONAL (MULTIPLE) DATUM ELEMENTS-->
#
#          <!-- OR -->
#
#   <!-- TBD OPTIONAL (MULTIPLE) DATUM ELEMENTS -->
#
#          <!-- OR -->
#
#   <!-- OPTIONAL (MULTIPLE) KEY ELEMENTS -->
#
#          <!-- OR -->
#
#   <!-- ANY OPTIONAL (MULTIPLE) ELEMENT IN ANY NAMESPACE -->
#
# </nmwg:data>
#
# ###############################################################

PingData =
        element nmwg:data {
                Identifier &
                MetadataIdentifierRef? &
                (
                        (
                                Metadata* |
                                PingMetadata*
                        ) |
                        (
                                PingCommonTime+ &
                                (
                                        PingDatum* |
                                        ResultDatum*
                                )
                        ) |
                        (
                                PingDatum* |
                                ResultDatum*
                        ) |
                        Key*
                )
        }


# ###############################################################
# CommonTime
# ###############################################################

PingCommonTime =
        element nmwg:commonTime {
                Type &
                (
                        TimeStamp |
                        (
                                StartTime &
                                (
```

```
                                        EndTime |
                                        Duration
                                )
                        )
                ) &
                (
                        PingDatum* |
                        ResultDatum*
                )
        }


# ############################################################
# These are the basic elements we would expect to see in the
# specific ping datum.
#
# Example:
#
# <ping:datum value="REQUIRED_VALUE"
#              valueUnits="OPTIONAL_VALUE_UNITS"
#              numBytes="OPTIONAL_NUM_BYTES"
#              numBytesUnits="OPTIONAL_NUM_BYTES_UNITS"
#              seqNum="OPTIONAL_SEQ_NUM"
#              ttl="OPTIONAL_TTL"
#              timeType="OPTIONAL_TIME_TYPE"
#              timeValue="OPTIONAL_TIME_VALUE"
#              xmlns:nmwg="http://ggf.org/ns/nmwg/tools/ping/2.0/">
#
#   <!-- TIME ELEMENT (IF ATTRIBUTES NOT USED) -->
#
# </ping:datum>
#
# ############################################################

PingDatum =
        element ping:datum {
                attribute value { xsd:float } &
                attribute valueUnits { xsd:string }? &
                attribute numBytes { xsd:int }? &
                attribute numBytesUnits { xsd:string }? &
                attribute seqNum { xsd:int }? &
                attribute ttl { xsd:int }? &
                (
                        (
                                attribute timeType { xsd:string } &
                                attribute timeValue { xsd:string }
                        ) |
                        Time
                )?
        }

# End Schema
```

8.2 Instance document for ping

```
<!-- Begin XML -->

<?xml version="1.0" encoding="UTF-8"?>
<nmwg:message type="store"
              xmlns="http://ggf.org/ns/nmwg/base/2.0/"
              xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
              xmlns:ping="http://ggf.org/ns/nmwg/tools/ping/2.0/"
              xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/"
              xmlns:nmtm="http://ggf.org/ns/nmwg/time/2.0/"
              xmlns:nmtl4="http://ggf.org/ns/nmwg/topology/l4/3.0/"
              xmlns:nmtl3="http://ggf.org/ns/nmwg/topology/l3/3.0/"
              xmlns:select="http://ggf.org/ns/nmwg/ops/select/2.0/"
              xmlns:average="http://ggf.org/ns/nmwg/ops/average/2.0/"
              xmlns:nmwgr="http://ggf.org/ns/nmwg/result/2.0/">

  <!-- Metadata using original topology schema -->

  <nmwg:metadata id="pingmeta1">
    <ping:subject id="pingsub1">
      <nmwgt:endPointPair>
        <nmwgt:src type="hostname" value="dreadnought.cis.udel.edu" port="4543"/>
        <nmwgt:dst type="hostname" value="alderaan.cse.psu.edu" port="34343"/>
      </nmwgt:endPointPair>
    </ping:subject>
    <ping:parameters id="pingparam1">
      <nmwg:parameter name="count">2</nmwg:parameter>
      <nmwg:parameter name="interval">3</nmwg:parameter>
```

```
    <nmwg:parameter name="deadline">10</nmwg:parameter>
  </ping:parameters>
</nmwg:metadata>


<!-- Metadata(s) using new topology schema -->

<nmwg:metadata id="pingmeta2">
  <ping:subject id="pingsub2">
    <nmtl4:endPointPair>
      <nmtl4:endPoint role="src" port="4543" protocol="icpm">
        <nmtl4:address value="dreadnought.cis.udel.edu" type="hostname"/>
      </nmtl4:endPoint>
      <nmtl4:endPoint role="dst" port="34343" protocol="icpm">
        <nmtl4:address value="alderaan.cse.psu.edu" type="hostname"/>
      </nmtl4:endPoint>
    </nmtl4:endPointPair>
  </ping:subject>
  <ping:parameters id="pingparam2">
    <nmwg:parameter name="count">2</nmwg:parameter>
    <nmwg:parameter name="interval">3</nmwg:parameter>
    <nmwg:parameter name="deadline">10</nmwg:parameter>
  </ping:parameters>
</nmwg:metadata>

<nmwg:metadata id="pingmeta3">
  <ping:subject id="pingsub3">
    <nmtl4:endPointPair>
      <nmtl4:endPoint role="src" port="4543" protocol="icpm">
        <nmtl3:interface id="d1">
          <nmtl3:ipAddress value="128.4.133.200" type="ipv4"/>
          <nmtl3:netmask>255.255.255.0</nmtl3:netmask>
          <nmtl3:ifName>eth0</nmtl3:ifName>
          <nmtl3:ifDescription>External Connection</nmtl3:ifDescription>
          <nmtl3:ifAddress value="128.4.133.200" type="ipv4"/>
          <nmtl3:ifHostName>dreadnought.cis.udel.edu</nmtl3:ifHostName>
          <nmtl3:ifIndex>0</nmtl3:ifIndex>
          <nmtl3:type>1000BaseT Ethernet</nmtl3:type>
          <nmtl3:capacity>1000000000</nmtl3:capacity>
        </nmtl3:interface>
      </nmtl4:endPoint>
      <nmtl4:endPoint role="dst" port="34343" protocol="icpm">
        <nmtl3:interface id="a1">
          <nmtl3:ipAddress value="130.203.16.20" type="ipv4"/>
          <nmtl3:netmask>255.255.255.0</nmtl3:netmask>
          <nmtl3:ifName>eth0</nmtl3:ifName>
          <nmtl3:ifDescription>External Connection</nmtl3:ifDescription>
          <nmtl3:ifAddress value="130.203.16.20" type="ipv4"/>
          <nmtl3:ifHostName>alderaan.cse.psu.edu</nmtl3:ifHostName>
          <nmtl3:ifIndex>0</nmtl3:ifIndex>
          <nmtl3:type>1000BaseT Ethernet</nmtl3:type>
          <nmtl3:capacity>1000000000</nmtl3:capacity>
        </nmtl3:interface>
      </nmtl4:endPoint>
    </nmtl4:endPointPair>
  </ping:subject>
  <ping:parameters id="pingparam3">
    <nmwg:parameter name="count">2</nmwg:parameter>
    <nmwg:parameter name="interval">3</nmwg:parameter>
    <nmwg:parameter name="deadline">10</nmwg:parameter>
  </ping:parameters>
</nmwg:metadata>

<!-- metadata(s) with operation metadata -->

<nmwg:metadata id="pingmeta4">
  <ping:subject id="pingsub4">
    <nmwgt:endPointPair>
      <nmwgt:src type="hostname" value="dreadnought.cis.udel.edu" port="4543"/>
      <nmwgt:dst type="hostname" value="alderaan.cse.psu.edu" port="34343"/>
    </nmwgt:endPointPair>
  </ping:subject>
  <ping:parameters id="pingparam4">
    <nmwg:parameter name="count">2</nmwg:parameter>
    <nmwg:parameter name="interval">3</nmwg:parameter>
    <nmwg:parameter name="deadline">10</nmwg:parameter>
  </ping:parameters>
</nmwg:metadata>

<nmwg:metadata id="pingmeta5">
  <select:subject id="pingsub5" metadataIdRef="pingmeta4" />
  <select:parameters id="pingparam5">
    <nmwg:parameter name="timeValue">
      <nmwg:parameter name="greaterThan">1107492199</nmwg:parameter>
    </nmwg:parameter>
  </select:parameters>
</nmwg:metadata>
```

```
<nmwg:metadata id="pingmeta6">
  <select:subject id="pingsub6" metadataIdRef="pingmeta5" />
  <select:parameters id="pingparam6">
    <nmwg:parameter name="timeValue">
      <nmwg:parameter name="lessThan">1107492207</nmwg:parameter>
    </nmwg:parameter>
  </select:parameters>
</nmwg:metadata>

<nmwg:metadata id="pingmeta7">
  <average:subject id="pingsub7" metadataIdRef="pingmeta6" />
  <average:parameters id="pingparam7">
    <nmwg:parameter name="value" />
  </average:parameters>
</nmwg:metadata>


<!-- Data block, with a time block, with multiple datum blocks -->
<nmwg:data id="data1" metadataIdRef="pingmeta1">
  <nmwg:commonTime type="unix" value="1107492095">
    <ping:datum seqNum="0" value="19.1" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="1" value="19.2" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
  </nmwg:commonTime>
</nmwg:data>

<!-- Data block, with a time block, with multiple datum blocks (other way to show time) -->
<nmwg:data id="data2" metadataIdRef="pingmeta1">
  <nmwg:commonTime type="unix">
    <nmtm:value>1107492096</nmtm:value>
    <ping:datum seqNum="0" value="19.3" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="1" value="19.4" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
  </nmwg:commonTime>
</nmwg:data>

<!-- Data block, with a time block (range), with multiple datum blocks -->
<nmwg:data id="data3" metadataIdRef="pingmeta1">
  <nmwg:commonTime type="range">
    <nmtm:start type="unix" value="1107492097"/>
    <nmtm:end type="unix" value="1107492395"/>
    <ping:datum seqNum="0" value="19.2" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="1" value="17.3" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="3" value="45.4" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="88" value="21.9" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
  </nmwg:commonTime>
</nmwg:data>

<!-- Data block, with a time block (durration), with multiple datum blocks -->
<nmwg:data id="data4" metadataIdRef="pingmeta1">
  <nmwg:commonTime type="durration" duration="300">
    <nmtm:start type="unix" value="1107492097"/>
    <ping:datum seqNum="0" value="19.2" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="1" value="17.3" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="3" value="45.4" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
    <ping:datum seqNum="88" value="21.9" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" />
  </nmwg:commonTime>
</nmwg:data>

<!-- data with datum blocks, time is inline (two ways to represent time) -->
<nmwg:data id="data5" metadataIdRef="pingmeta1">
  <ping:datum seqNum="0" value="14.3" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" timeType="unix" timeValue="1107492199" />
  <ping:datum seqNum="1" value="17.4" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes" timeType="unix" timeValue="1107492201" />
</nmwg:data>

<!-- data with datum blocks, time is a sub element -->
<nmwg:data id="data6" metadataIdRef="pingmeta1">
  <ping:datum seqNum="0" value="19.6" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
    <nmtm:time type="unix" value="1107493095" />
  </ping:datum>
  <ping:datum seqNum="0" value="18.5" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
    <nmtm:time type="unix" value="1107493095" />
  </ping:datum>
</nmwg:data>

<!-- data with datum blocks, time is a sub element -->
<nmwg:data id="data7" metadataIdRef="pingmeta1">
  <ping:datum seqNum="0" value="19.6" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
    <nmtm:time type="unix">
      <nmtm:value>1107493095</nmtm:value>
    </nmtm:time>
  </ping:datum>
  <ping:datum seqNum="0" value="18.5" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
    <nmtm:time type="unix">
      <nmtm:value>1107493095</nmtm:value>
    </nmtm:time>
  </ping:datum>
</nmwg:data>

<!-- data with datum blocks, time is a sub element (other way to show time) -->
```

```
  <nmwg:data id="data8" metadataIdRef="pingmeta1">
    <ping:datum seqNum="0" value="19.6" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
      <nmtm:time type="range">
        <nmtm:start type="unix" value="1107492095"/>
        <nmtm:end type="unix" value="1107492395"/>
      </nmtm:time>
    </ping:datum>
    <ping:datum seqNum="0" value="18.5" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
      <nmtm:time type="range">
        <nmtm:start type="unix" value="1107492095"/>
        <nmtm:end type="unix" value="1107492395"/>
      </nmtm:time>
    </ping:datum>
  </nmwg:data>

  <nmwg:data id="data9" metadataIdRef="pingmeta1">
    <ping:datum seqNum="0" value="19.6" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
      <nmtm:time type="durration" duration="300">
        <nmtm:start type="unix" value="1107492095"/>
      </nmtm:time>
    </ping:datum>
    <ping:datum seqNum="0" value="18.5" valueUnits="ms" ttl="241" numBytes="64" numBytesUnits="bytes">
      <nmtm:time type="durration" duration="300">
        <nmtm:start type="unix" value="1107492095"/>
      </nmtm:time>
    </ping:datum>
  </nmwg:data>

  <!-- result datum elements -->
  <nmwg:data id="data10" metadataIdRef="pingmeta1">
    <nmwgr:datum type="error.ping.mp">From lager (192.168.0.200) icmp_seq=1 Destination Host Unreachable</nmwgr:datum>
  </nmwg:data>

</nmwg:message>

<!-- End XML -->
```

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 [1]

10 Security Considerations

There are important security concerns associated with the generation and distribution of network measurement information. For example, ISPs frequently consider network configuration and performance information to be proprietary. Furthermore, observing traffic, and, in particular, collecting packet headers, is frequently considered a violation of the presumption of privacy on the network. Systems that collect the measurements described here are sometimes regarded as invasive, and, indeed, poorly designed or configured monitoring tools can consume a disproportionate amount of network bandwidth. Port blocking, protocol blocking, and traffic shaping can impact many measurement tools. Tools, such as traceroute, that send UDP probes to increasing port numbers can appear to be port scans and raise security alerts.

We do not address those concerns in this document, but implementers are encouraged to consider the security implications of generating and distributing measurement information. While distribution of end-to-end application-level measurements is generally accepted, measurements that identify individual users or consume noticeable amounts of resources 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

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

12 Acknowledgements

We gratefully acknowledge the contributions of: Jeff Boote, Eric Boyd, Mark Leese, Dan Gunter, Richard Hughes-Jones, Jason Zurawski and the other members of the Network Measurements Working Group.

13 Glossary

...

14 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

15 Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

16 Full Copyright Notice

References

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

[2] 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://nmwg.internet2.edu.

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

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

[5] Network Markup Language Working Group (NML-WG). `https://forge.gridforum.org/projects/nml-wg`.

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

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

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

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

[10] J. Zurawski, M. Swany, and D. Gunter. A scalable framework for representation and exchange of network measurements. In *IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, Barcelona, Spain, March 2006.