GGF-NMWG-??
Network Measurements WG
http://www-didc.lbl.gov/NMWG/

Dan Gunter
Lawrence Berkeley National Lab
February 17, 2005

How-to write your own schema

Status of this memo

This memo provides informtaion to the Grid community. It does not define any standards or technical recommendations. Distribution is unlimited

Copyright Notice

Dan Gunter is supported by the U.S. Dept. of Energy, Office of Sciences, Office of Computational and Technology Research, Mathematical Information and Computational Sciences Division, under contract DE-AC03-76SF00098 with the University of California.

Abstract

How-to guide for writing new NM-WG schemas.

Contents

1 Introduction

This document is intended as a guide for creating your own NM-WG-compatible measurement schemas. This guide does not discuss how to make these schemas into a Web Service and deploy them. More detailed instructions for those tasks are in the NMWG Developer's Guide. Also in the developer's guide is a description of the base NM-WG schemas.

2 Recommended reading

This guide uses the RELAX-NG compact syntax as its schema language. Basic familiarity with this syntax is assumed. See the RELAX-NG homepage at `http://www.relaxng.org/` for details.

3 Step 1: Get base schemas

The base schema is shown in Appendix I, and should also be available from the NMWG home page. Place it in a file called `nmbase.rnc`.

4 Step 2: Write inherited schema

After you have the base schemas and have installed Trang, create a file for your inherited schema. In the examples below, the file will be called myschema.rnc and the name of all the pieces will be prefixed with "myschema", with the words "my" and "schema" capitalized and separated appropriately.

4.1 Choose Schema name

Schema names should be hierarchical. In our example, there is a two-level hierarchy: "my" and its child "schema". Schemas which are to be part of the standard NMWG namespaces should also be identified as either "tools" or "characteristics". Basically, a characteristic is a generalization of the metrics provided by one or more tools, whereas a tool is something like iperf or ping.

This example will assume that "myschema" is a characteristic, but place it in a non-NMWG namespace, in an attempt to illustrate both the use of URLs outside NM-WG, and how NM-WG arranges it.

4.2 Schema header

The first part of the schema declares namespaces and includes other external pieces. Your header should look roughly like this.

```
default namespace = "http://my.own.org/characteristic/my/schema/"
namespace nmwg = "http://www.ggf.org/nmwg/"
include "nmbase.rnc"
```

The more general form of the namespace is: *unique url*/characteristic/*schema name*, where the hierarchical schema name is split by forward slashes. For example, the NM-WG characteristic "delay.roundTrip" becomes "http://www.ggf.org/nmwg/characteristic/delay/roundTrip".

4.3 Exend base schema

The next part is a mechanical extension of the base schema you included in the header. The only change needed is to replace MySchema with the name of your schema.

```
Metadata |=
   MySchemaMetadata

Results |=
   MySchemaResults

MySchemaMetadata =
   element metadata {
      attribute id { nmwg:Identifier },
      element subject { MySchemaSubject },
      element parameters { MySchemaParameters }
   }
```

4.4 Define subject

The "subject" is the thing being measured, e.g., a network link. Some common topology schemas are defined in the base schema, so you may be able to just use this. For example, to set the subject as a host pair (src,dst) or host pair with wildcards, add the following:

```
MySchemaSubject =
   nmwg:Subject,
   ( nmwg:HostPair | nmwg:HostPairQuery )
```

4.5 Define parameters

The schema for the parameters depends entirely on the particular measurement tool or characteristic you are developing. What follows is an abbreviated example from an "iperf" definition.

```
IperfParameters =
   element frmt { xsd:string }?,
   element transtime { xsd:int }?,
   element intrvl { xsd:int }?,
   element windowsz { xsd:float }?
   # etc..
```

4.6 Define results

The results section is similar to the parameters section. By convention, multiple results are represented by making them children of a parent "results" element. Again, an example from "iperf" is shown.

```
IperfResults =
   element results {
      element probe {
         attribute num { xsd:int },
         element interval { string },
         element transfer { xsd:float },
         element bandwidth { xsd:float }
      }*
   }
```

5 Conclusion

Congratulations, you've written a schema. In order to make an XML Schema that you can include in Web Services (WSDL) documents, you need to run a conversion tool. Further details are in the NMWG Developer's Guide.

6 Appendix I: NM-WG Base Schema

```
default namespace = "http://www.ggf.org/nmwg/"
namespace nmwg = "http://www.ggf.org/nmwg/"

start =
   element request { Request } |
   element response { Response }

Request =
  ## Properties of the request
  element properties { Properties }?,
  ## Request metadata (body of request)
  ## The request metadata could include an attribute for each element
  ## describing whether it requires and exact match, or whatever...
  Metadata*

Response =
  ## Properties of the response
  element properties { Properties }?,
   ## Metadata / data sections
   ( Metadata | Data ) *

## Metadata section for a request or response.
## This generic element will be replaced by
## a <metadata> element in some other namespace that
## identifies the actual type of metadata and
## constrains the type of the enclosed subject and
## parameters.
Metadata =
   element metadata {
      attribute id { Identifier },
      element subject { Subject },
      element parameters { Parameters }
   }

## Metadata subject
Subject =
  attribute id { Identifier }

## Metadata parameters
Parameters =
  attribute id { Identifier }

## Data section.
Data =
```

```
   element data {
      ## Data section identifier
      attribute id { Identifier },
      ## Reference to associated metadata section
      attribute metadataId { Identifier },
      ## Initial or "base" timestamp
      element time { Time }?,
      ## Units
      element units { token }?,
      ## Result data, subclasses will extend
      ## for their purposes
      Results
   }

## Base result type
Results =
   element results { text }

## Identifier for part of a request
## or response
Identifier = xsd:string

## Topology section.
HostPair =
   element hostPair {
      element src { Endpoint },
      element dst { Endpoint }
   }

HostPairQuery =
   ## Many different src/dst endpoints may match these patterns
   element hostPairQuery {
      element src { EndpointPattern }+,
      element dst { EndpointPattern }+
   }

Endpoint =
   ## 'hostname', 'ipv4', 'ipv6', other..
   attribute type { token },
   element address { string },
   element port { xsd:int }?

EndpointPattern =
   ## types for Endpoint as well as
   ## 'hostnameWildcard','ip{v4,v6}Wildcard', 'ip{v4,v6}Mask'
   ## e.g. '*.lbl.gov', 131.243.2.*, 131.243.0.0/16
   attribute type { token },
   element address { string },
   ## missing port implies 'any'
   element port { xsd:int }?
```

```
## Timestamp
Time =
   ## Format and semantics of value
   attribute type { token },
   ## Timestamp value
   attribute value { text }

TimeRange =
   element start { Time },
   element end { Time | empty }
```