

Schemas for Grid Performance Events

Grid Performance Working Group

Dan Gunter, Warren Smith

October 3, 2000

1 Introduction

This document describes a proposal for standard schemas to represent grid performance events and information related to these events. Our motivation for this work is to provide a common set of definitions so that information can be communicated between different grid monitoring systems. In this document, we first describe the information we wish to represent in general terms. We can then define one or more representations of this information that meet our requirements for readability, size, or any other requirements. Second, we describe a representation of this information using the extensible Markup Language (XML). We choose to propose an XML representation because such a representation is textual, easily readable by eye, and there are many tools available to parse XML documents. We suspect that more efficient representations may be defined in the future.

The basic piece of information that we wish to represent is an event. An event contains, typically, a relatively small set of information that is communicated from the process that produces it to the processes that wish to consume it. The other major piece of information that we wish to represent is an event type. An event type describes what information must and may be in an event of that type. We also define several other types of information we wish to represent.

We begin in Section 2 with an overview of the components of a monitoring system and other background information. Section 2 contains a general description of the information we need to represent. Section 4 contains descriptions of the XML schemas used to represent various pieces of information.

2 Background

This section provides background information to describe the framework we are working in and the information that needs to be exchanged. A simple view of monitoring in a computational grid is that there are three types of components:

1. Event producers that produce events. An event producer is a component such as a host monitor that is gathering information about its host and will provide this information to appropriate consumers.
2. Event consumers that receive events from producers. An event consumer is a component such as a scheduler for a user that wants to monitor hosts as the user's applications execute on the host.
3. Information services that provide other types of information. An information service can be used to contain information about the location of event producers and the type of information the producers can provide, among other things.

In this document, we define several different pieces of information that are used in this architecture. First, we define the events that are sent from consumers to producers. Second, we want these events to be typed so that they can be easily understood so we define how to define event types. Third, consumers request events from producers so we define how consumers can specify event descriptions to describe

the events they want to receive. Fourth, we must address some smaller issues such as how do we represent units and how do we create dictionaries of event types.

3 Information Models

This section provides abstract models of the information that we need to represent.

3.1 Event

A grid performance event contains one or more measurements which pertain to the performance of an entity in a computational grid. For example, an event can contain information about the load on a computer system, the available bandwidth over a network, the status of a HTTP server, or the results of a scientific application. Every performance event must adhere to the constraints described by one or more event types (as described in Section 2.2). This requirement is most useful if the events are generated, transmitted, and stored in such a way that the associated event type(s) are not lost, otherwise the consumer of the events may have difficulty reliably retrieving the measurements and measurement context contained within.

We are considering two approaches to describing an event.

3.1.1 First Approach

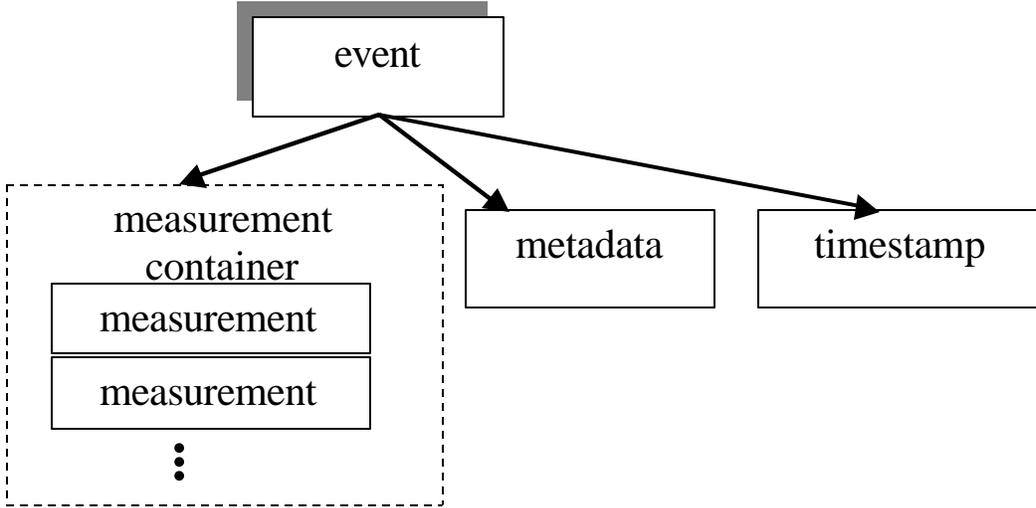
In this approach, each event consists of a type and a set of generic elements:

Type. The type of an event identifies a particular set of information that describes what information may and must be part of an event (we describe event types in Section 3.2).

Element. The elements in an event contain the information. In it's most basic form, an element is a name and a value. The name is a string and the value can be of any basic type (int, long, float, ...) or a string. At the current time, we do not see a need for more complex types, but this is something to discuss (lists, arrays, and structures come to mind). For example, a CPU load event might have an element <Load5, 4.5> that indicates that the 5 minute load on the machine is 4.5. There is other information that may be needed for an element. The other information we think should be supported are units and accuracy. In many cases, execution htime for example, a value is meaningless without a units associated with it. The same is true for accuracy: a user may want to know how accurate a measurement is.

3.1.2 Second Approach

In this approach, an event has a "measurement container" which has one or more measurements, some metadata which provides context for the measurements, and a timestamp.



3.2 Event Type

The information carried in an event will be structured according to the schema of the *event type* to which it belongs. An event type defines which elements must be present in an event of that type and which elements may be present in an event of that type.

We present here two possibilities for the rules governing construction of event types. These correspond to the first approach and second approach in the Event section above.

3.2.1 First Approach

For each element that may or must be in an event, the event type must contain:

- The name of the element
- The type of the data in the element
- A text description of the element

And may contain:

- The default units for the value
- The default accuracy of the value

In this approach, each event type also contains a unique name to identify it, the name of its parent type, and a text description of the event type.

There is a basic set of information that is included in every event type because every event may or must have this information. This basic information can be included in every event type in an implied way or it can be included by inheritance from a base event type. The following elements are required in all events:

Name	Type of Value	Description
Type	String	A string that identifies which event type the event is associated with.
Time	Timestamp	The time that the event was created.
Source	String	A URL that identifies the original source of the event.

The following elements are optional in all events:

Name	Type of Value	Description
SequenceNumber	Integer	What exactly can a sequence number mean?

3.2.2 Second Approach

Event types can be divided into two sections: elements which must be present in the event (required), and elements which may be present in the event (optional). Below is a list of these, followed by a description of each. Required elements should be decided on and stay relatively static, as any changes will cause previous event types to become obsolete. The optional elements may expand over time, as it becomes apparent that interoperability would benefit if a commonly-used attribute were consistently named across systems and implementations.

Required in all event types

- o name of event
- o timestamp
- o measurement container

Optional in all event types

- o target
- o source
- o description
- o expiration
- o sequence-number

name of event. string which is “standardized” in some way, whether by common agreement or by a more formal mechanism such as a mapping to an ASN.1 OID. For easy serialization into different representations, the event name should probably not have embedded whitespace or punctuation.

timestamp. the moment in time to which the measurements in the measurement container apply. Certain measurements may have additional timing information contained within them; it is recommended that where possible this is represented as an offset from the top-level timestamp.

measurement container. a grouping of one or more measurements, and possibly nested containers. The simplest container is the “null container”, i.e. simply one measurement. Other possible containers include: vector, list, 2-D array, N-D array, and binary tree.

target. the variable part of the thing which is described by the event, for instance the two hostnames and packet size in a “ping” event. If the event is thought of as a sample in an experiment, the target is equivalent to the experimental parameters.

source. the location of the entity which is sending this event. It is expected that the source would normally be described as an IP address, but more abstract identifiers, such as a URL or the name of a service, are also possible.

description. string, with embedded whitespace, which provides a more verbose description of the event, including such things as known inaccuracies, references to standards documents, etc. Due to the performance implications, it is expected that the description will only be carried in events intended for immediate human consumption.

expiration. a timestamp which shows the last valid “moment” of the data contained within this event. The interpretation of the relevance of stale data is, of course, application-dependent.

sequence-number. integer value pertaining to the order of this event relative to some set of other events.

3.3 Event Description

When a consumer asks for specific events from a producer, it needs some way to describe those events. We call this an event description. An event description consist of two parts: event parameters and an event filter. There will be cases where a consumer must supply information when asking for events. For example, if a consumer wishes to inquire about the state of a process, it must specify the process identifier. This process identifier is an event parameter. A consumer also may want to provide a filter that constrains when events are sent. For example, if a consumer only wishes to receive an event about process state when the process is complete, the consumer would specify a filter that contains the expression: “process does not exist”. Parameters differ from a filter because the parameters act as input to the producer instructing the producer under what conditions to generate events while the filter is then used to determine if each of these events should be sent to the consumer. These two different types of information could be specified in the same expression but this could lead to confusion about which parts of the expression are assignments and which are comparisons.

Parameter types are associated with an event type and either must or may be specified when requesting events of that type from a producer. An event parameter type must contain:

- The name of the parameter
- The type of the data in the parameter
- A text description of the parameter

And may contain:

- The default units for the parameter

An event filter is a logical expression of valid event element names, event parameter names, and values for the events the filter will be applied to.

3.4 Unit Definitions

One difficulty that we could address is defining the meanings of different units. For example, we need some technique for determining that km = kilometer = 1000 meters. A set of unit definitions provides this functionality.

The definition for units is based upon the *Système International d'Unités* (SI), the modern form of the metric system (for more information see [1], [2] and [7]). Before going into more detail, it is important to understand that underlying every unit of measurement is an implied "dimensionality", that is, an orthogonal dimension of physical quantity, which the unit simply subdivides in a uniform manner. Thus, SI has only seven "base units", each corresponding to a separate dimensionality. All other units, such as weight, resistivity, area, and bandwidth, can be expressed as combinations of these base units. The final row in italics is an area of dimensionality which is not conveniently expressed with the other units, and which is crucial to computer performance measurement: the bit, which represents a single boolean value or, alternately, the smallest piece of digital information.

Dimension = Physical quantity	Base Unit	Symbol
-------------------------------	-----------	--------

length	metre	m
time	second	s
mass	kilogram	k
electric current	ampere	A
thermodynamic temperature	kelvin	K
luminous intensity	candela	cd
amount of substance	mole	mol
<i>digital information</i>	<i>bit</i>	<i>b</i>

The actual list of unit symbols, e.g. “meter” or “second”, and their abbreviations can be extended; the commonly recognized types could be stored in a Unit Dictionary, as described below. The name and symbol of a number of derived units can also be included. In computer performance, these would include:

Base unit combination	Derived Unit	Symbol
8*bit	Byte	B
1/second	Hertz	Hz
mole/6.022E22	Count	ct

The symbol used for the unit itself, e.g. for megabytes the “MB”, can be naturally divided into two parts: a *unit prefix* indicating quantity, and a *unit symbol*. There are a standard set of metric prefixes which account for powers of ten from +24 to -24. Each of these prefixes has a standard abbreviation, as in “k” for “kilo”. For reference, the prefixes, abbreviations, and corresponding power of ten (10) are listed in the table below.

Prefix	Abbrev.	Power of ten	Prefix	Abbrev.	Power of ten
Yotta	Y	24	deci	d	-1
Zetta	Z	21	centi	c	-2
Exa	E	18	milli	m	-3
Peta	P	15	micro	u	-6
Tera	T	12	nano	n	-9
Giga	G	9	pico	p	-12
Mega	M	6	femto	f	-15
Kilo	k	3	atto	a	-18
Hecto	h	2	zepto	z	-21
Deka	da	1	yocto	y	-24

However, the observant reader will have noticed that the term “MB” means 10^6 bytes, but is commonly used in computer measurements to mean 2^{20} bytes -- these are not the same! In order to resolve this confusion, the SI defines some separate units for powers of two:

Prefix	Abbreviation	Power of two
kibi	Ki	10
mebi	Mi	20
gebi	Gi	30
tebi	Ti	40
pebi	Pi	50
exbi	Ei	60

Therefore, if what is really intended is 2^{20} bytes, the proper abbreviation is MiB/s.

The general units schema will define the desired unit in terms of the above base/derived units, and the operators for multiplication and division. This will be presented in more detail in the section on wire protocols.

3.5 Unit Dictionary

A general and formal description of the units contained in every measurement would, in most circumstances, be superfluous and burdensome. The units used for measurements usually change infrequently, and are relatively simple. Instead, a “unit dictionary” could be used to store the commonly used units in the common scientific string representation (see Section 3.4) and, if necessary, associate these with numeric identifiers. In addition, a human readable description, in as many languages as possible, should be an integral part of each entry in the dictionary. Thus the unit dictionary could make implementer’s lives simpler by providing a compact common representation for common units, and it could make user’s lives easier by providing descriptive material in a well-known location.

For example, measurements for network bandwidth are frequently shown in megabits per second. Given the unit definitions from Section 2.1, megabits per second could be broken down into “(prefix=Mi, unit=b) divided-by (prefix=1, unit=s)” and then mapped to the chosen representation. However, with an entry in the unit dictionary for “Mib/s”, this simple string could be mapped instead into the chosen representation, with no chance of ambiguity on the receiving side.

3.6 Event Dictionary

An event dictionary is simply a database containing the event types that provide us a common language for the exchange of events between systems. An event dictionary could be stored in a file, on a Web page, or a LDAP database, the emerging standard for grid information services. Each entry in an event dictionary is an event type which is described in general in Section 3.2 with a representation proposed in Section 4.1.

An initial list of event types to place in this dictionary is:

- Network
 - latency (ping)
 - throughput (netest, iperf, netperf)
 - path (traceroute)
- Disk statistics (iostat)
 - reads/writes per sec

- utilization
- total size, available size, mount point
- Memory statistics (vmstat)
 - swap space
 - free memory
 - page faults
- CPU statistics (vmstat)
 - User
 - System
 - Idle
- Process statistics (ps, top)
 - Process state, priority, running time
- Heartbeats
- File statistics
 - Owner, permissions, size, modification time

4 XML Representations

The concepts presented in the preceding section could be mapped to a number of different concrete representations. We have chosen to show the mapping to XML[5], and in particular XML using XML Schema[6], for several reasons. First, XML has a straightforward syntax that makes it a good choice for presenting new ideas. Second, as of this writing XML is gaining momentum in the commercial sector, and therefore a number of high-quality tools for creating and using XML-based data are becoming available. Finally, a repository of XML schemas is as easy to create as a web page, and with little effort can be made easy to browse and visually effective. As an application-layer protocol for transport of events between applications, developers must balance the advantages cited above with the performance penalty when compared with a more compact encoding such as BER or PER for ASN.1 [ref].

A single-inheritance model will be adopted in which the “base type” outlined in Section 2.2 is specialized by adding new required attributes. This base type will be called “GridEvent”.

A minor point to address is style guidelines. In this document the following style has been followed:

- Names of event types are mixed-case with the first letter capitalized, e.g. MyEventType
- Elements of event types are mixed-case with the first letter lowercase, e.g. myAttribute
- To simplify syntax, XML attributes have been mostly unused; instead we used sub-elements
- All schemas omit the standard XML Schema header, which provides the namespace of the schema using a URL, and a declaration of the “grid” namespace. The following lines are implicit in every schema definition:

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:grid="http://www.gridforum.org/PerfWG/schemas">
```

- Unless otherwise noted, the default namespace for all schema elements is “grid:”

4.1 Events

We do not need to define a schema or language for representing events. We can use XML Schema to define event types (see Section 4.1) and this defines the schemas for events.

We provide two examples for CPU load, based on the two alternative representations in the next section.

4.1.1 Using schema from Section 4.2.1

```
<CPULoadEvent>
  <description>1, 5, and 15 minute CPU load averages</description>
  <source>x-event://myhost.gridforum.org:1234</source>
  <time>2000-06-30T13:00-05:00</time>
  <hostName>foo.gridforum.org</hostname>
  <load1>1.2</load1>
  <load5>1.4</load5>
  <load15>0.9</load15>
</CPULoadEvent>
```

This event starts and begins with tags identifying the event type. Inside of these tags are XML elements that describe the source of the event, when the event was generated, and the data.

4.1.2 Using the schema from Section 4.2.2

To illustrate where the units or error bars would go in this XML mapping, an error estimate has been added to the event instance.

```
<VmstatCPULoadEvent>
  <description>1, 5, and 15 minute CPU load averages</description>
  <source>myhost.gridforum.org</source>
  <target>foo.gridforum.org</target>
  <timestamp>20000630130012.567123Z</timestamp>
  <values>
    <errorPlus> 2.0 </errorPlus>
    <errorMinus> 2.0 </errorMinus>
    <load1> 1.2 </load1>
    <load5> 1.4 </load5>
    <load15> 0.9 </load15>
  </values>
</CPULoadEvent>
```

4.2 Event Types

Our strategy is to define a base `GridEvent` element and then have all other events be defined using XML Schema and extend the `GridEvent` element directly or through a chain of event types that extend `GridEvent`. In short, we use single inheritance exclusively (no composition). In XML Schema, inheritance is performed by defining a *complexType* with the superclass in the *base* attribute. Optional elements are indicated with the attribute *minOccurs=0*.

Corresponding to the two approaches presented in the Information Models section (Section 3.2), there are two alternative XML representations of event types.

4.2.1 First Approach

The base type is represented as follows:

Base Grid Event Type Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:gp="http://www.gridforum.org/PerfWG/schemas">
  <xsd:complexType name="GridEvent"/>
    <xsd:element name="source" type="xsd:string"/>
    <xsd:element name="time" type="xsd:timeInstant"/>
    <xsd:element name="sequenceNumber" type="xsd:integer" minOccurs="0"/>
  </xsd:complexType>
</xsd:schema>
```

An example of a simple schema for CPU load information that derives from GridEvent is shown below.

CPU Load Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:gperf="http://www.gridforum.org/PerfWG/schemas">
  <xsd:annotation><xsd:documentation>
    This event describes the CPU load on a host at the specified time.
  </xsd:documentation></xsd:annotation>
  <xsd:element name="CPULoadEvent">
    <xsd:complexType base="grid:GridEvent" derivedBy="extension">
      <xsd:element name="hostname" type="xsd:string"/>
      <xsd:element name="load1" type="xsd:float"/>
      <xsd:element name="load5" type="xsd:float"/>
      <xsd:element name="load15" type="xsd:float"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

4.2.2 Second Approach

The concept of “measurement container” is most easily and naturally represented by an XML Schema element which encloses all the measurements in the event schema. Because this element uses the measurement type, it includes optional attributes for error bars and units, which then apply to all the measurements contained within the element. Of course, if individual measurement values need to have separate error bars and/or units, these can be provided by using “measurement type” elements for them, instead of scalar values.

Base Grid Event Type Schema

```
<xs:schema xmlns:xs="http://www.w3.org/1999/XMLSchema"
  xmlns:gp="http://www.gridforum.org/PerfWG/schemas">
<xs:complexType name="GridEventType">
  <xs:sequence>
    <xs:element name="timestamp" type="timestamp"/>
    <xs:element name="values" type="measurementType"/>
    <xs:element name="source" minOccurs="0"/>
    <xs:element name="target" minOccurs="0"/>
    <xs:element name="expiration" type="timestamp" minOccurs="0"/>
    <xs:element name="description" type="xs:string" minOccurs="0"/>
    <xs:element name="sequence" type="xs:integer" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Measurement Type Schema (part of GridEventType, above)

```
<xs:complexType name="measurementType" mixed="true">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="1">
      <xs:element name="SimpleUnit"/>
      <xs:element name="GeneralUnit"/>
    </xs:choice>
    <xs:element name="errorPlus" minOccurs="0" type="xs:float"/>
    <xs:element name="errorMinus" minOccurs="0" type="xs:float"/>
  </xs:sequence>
</xs:complexType>
```

An example of a simple schema for CPU load information which derives from GridEventType is shown below.

CPU Load Schema

Dan Gunter, Warren Smith

```

<xs:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
            xmlns:gperf="http://www.gridforum.org/PerfWG/schemas">
<xs:element name="VmstatCPULoadEventType">
  <xs:complexType base="GridEventType">
    <xs:complexContent/>
    <xs:extension/>
    <xs:element name="values" type="measurementType">
      <xs:extension/>
      <xs:sequence>
        <xs:element name="load1" type="xs:float"/>
        <xs:element name="load5" type="xs:float"/>
        <xs:element name="load15" type="xs:float"/>
      </xs:sequence>
    </xs:element>
  </xs:complexType>
</xs:element>

```

4.3 Event Descriptions

As described previously, there are two parts to an event description. First, the parameters used to describe when to generate events. Second, the filter used to determine which of these events should be sent to the consumer. We accomplish this by defining a base GridEventDescription and then extending this document type. Our basic GridEventDescription is:

Grid Event Description

```

<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
            xmlns:gp="http://www.gridforum.org/PerfWG/schemas">
  <xsd:complexType name="GridEventDescription"/>
  <xsd:element name="filter" type="xsd:string" minOccurs="0"/>
  <xsd:element name="sequenceNumber" type="xsd:integer" minOccurs="0"/>
</xsd:complexType>
</xsd:schema>

```

As you can see, the only element that is in the base grid event description is a filter that is a string. This string contains a logical expression of event element names, event parameter names, and values that the producer will use to filter events. We have not decided on what type of logical language to use for this filter. An example of a simple schema for a process status description is:

Instance of Grid Event Description

```

<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
            xmlns:gperf="http://www.gridforum.org/PerfWG/schemas">
  <xsd:annotation><xsd:documentation>
    This event description provides the information that will be passed to
    a producer when a consumer requests process status events.
  </xsd:documentation></xsd:annotation>
  <xsd:element name="ProcessStatusEventDescription">
    <xsd:complexType base="grid:GridEventDescription"
      derivedBy="extension">
      <xsd:element name="processID" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Note that the processID is required because it doesn't make sense to ask for information about a process without specifying which process.

4.4 Unit Types

If we decide to use a “unit dictionary” to keep string representations of commonly used units, there will be two ways of representing a unit in an event: as a simple string, or as a structure. These two types of units are called SimpleUnit and GeneralUnit, respectively. XML representations of the corresponding schemas are shown below. Note that even with the general unit schema, the unit names and prefixes are not enumerated but rather would be hardcoded into both the sender and receiver.

SimpleUnit schema

```
<xsd:element name="SimpleUnit" type="xsd:string">
</xsd:element>
```

GeneralUnit schema

```
<xsd:element name="GeneralUnit">
  <xsd:sequence>
    <xsd:element name="unitName" type="xsd:string"/>
    <xsd:element name="operand">
      <xsd:choice>
        <xsd:element name="mul"/>
        <element name="div"/>
      </xsd:choice>
    </xsd:element>
  </xsd:sequence>
  <xsd:element name="unitName" type="xsd:string"/>
</xsd:element>
```

In the SimpleUnit schema, the string representation takes the following form (shown here in EBNF notation). The <prefix> <unit> part of this grammar is also used in the *unitName* element of the GeneralUnit schema.

```
simple-unit ::= <prefix> <unit> <operator> <simple-unit> | <prefix> <unit>
prefix     ::= Y | Z | E | P | T | G | M | ... | p | f | a | z | y | Ki | Mi .. | Ei
unit       ::= b | B | Hz | s | ct | ...
operator   ::= / | *
```

As an example, the representation in the GeneralUnit schema of “megabits per second per disk” is:

```
<GeneralUnit>
  <unitName> Mib </unitName>
  <div/>
  <unitName> s </unitName>
  <div/>
  <unitName> ct </unitName>
</GeneralUnit>
```

An equivalent representation in the SimpleUnit schema would be:

```
<SimpleUnit> Mib/s/ct </SimpleUnit>
```

4.5 Unit Dictionary

There would be two required parts to each entry in the unit dictionary: the unique string for the unit, and a list of one or more descriptive text sections which are each identified with their target language. Because a unit dictionary requires structure, an international character set, and easy representation for printing and browsing, we recommend XML as the method of representation. Here is one possible schema and an example instance.

XML Schema for Unit Dictionary

```

<xsd:schema xmlns:xsd="http://www.w3.org/XMLSchema/"
  xmlns:gridUD="http://grid-namespace/UnitDictionary">
<xsd:element name="grid:UnitDictionary">
  <sequence>
    <xsd:element name="gridUD:unit">
      <xsd:element name="gridUD:name" type="xsd:string"/>
      <xsd:element name="gridUD:description">
        <sequence>
          <xsd:element name="gridUD:language" type="xsd:string"/>
          <xsd:element name="gridUD:text" type="xsd:string"/>
        </sequence>
      </xsd:element>
    </xsd:element>
  </sequence>
</xsd:element>

```

XML Instance of Unit Dictionary entry

```

<unit>
  <name> Mb/s </name>
  <description>
    <language> English </language>
    <text> Megabits per second. Commonly used for network bandwidth </text>
  </description>
</unit>

```

It is an open question whether the schema should be augmented with an Object Identifier (OID) which could be used as an alternative form of unit representation inside events.

References

- [1] ANSI/IEEE Std 268-1992. American National Standard for Metric Practice. Published by the IEEE, October 28, 1992.
- [2] BIPM. Le système international d'unités (SI) 6e édition, textes français et anglais. Sèvres, France. Bureau international des poids et mesures.
- [3] T. Howes, M. Smith, G. Good. *Understanding and Deploying LDAP Directory Services*. Macmillan Technical Publishing, 1999.
- [4] T. Howes and M. Smith. *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.
- [5] XML. <http://www.w3.org/XML/>
- [6] XML Schema. <http://www.w3.org/XML/Schema.html>
- [7] Theodore Wildi. *Metric Units and Conversion Charts - A metrication Handbook for Engineers, Technologists and Scientists*. IEEE Press, 1995.