Document: GWD-GIS-020-1
Category: Informational

Dan Gunter
Lawrence Berkeley National Laboratory
Keith Jackson
Lawrence Berkeley National Laboratory
June, 2001

The Applicability of RDF-Schema as a Syntax for Describing Grid Resource Metadata

1.0 Introduction

Grid applications, such as schedulers, authorization and authentication frameworks, and replication managers, need access to metadata about the objects of their operations. In this document these objects will collectively be called "resources". Resource metadata would include, for example, the names and locations of compute and storage hardware, available and maximum bandwidth on network links, lists of user accounts and real-world data on the corresponding person or group, and public keys and certificates. Some applications may need only a specialized set of metadata, but in general there will be a great deal of overlap, for example both schedulers and replication managers will need information about network links to make intelligent decisions and for failure recovery.

Because many different Grid applications will need to access and share the same resource metadata, a common syntax for the exchange of this metadata would be a useful service to Grid developers. A single syntax is needed by documents, discussions and applications, so that both people and programs can avoid an $O(N^2)$ translation problem between N different syntaxes, and the related O(N) tool-building problem. The main requirement for such a syntax is that it provide sufficient generality and extensibility to adequately describe all resource metadata, while remaining simple enough to discuss and manipulate both "by hand" and programmatically. While it is certainly possible to have one syntax for people and another for machines, the necessary translation layer would then require double the expertise and would be a likely area of subtle failures. Therefore, it is asserted that a single text-based syntax which is both unambiguous (and therefore machine-readable) and human-readable is preferable. The Grid Object Specification (GOS) syntax is an example of one such syntax.

However, a particular syntax necessarily is linked to a particular type of knowledge representation. It is important to evaluate how well that knowledge representation maps to the requirements of representing Grid resources. A crucial requirement is, of course, that Grid resources and their relationships can be expressed naturally and simply. Another important requirement is that the resulting expressions can be transformed into suitable input for existing information storage/retrieval technologies, such as hierarchical (e.g.: LDAP) or relational (e.g.: SQL) databases. Because it is impossible for one syntax to work equally well with all these technologies, the key question is whether the syntax provides a superset of the data model inherent in the data indexing technologies, and if so whether the syntax can then be restricted to be usable in conjuction with the technologies.

In this document, we will evaluate the Resource Description Framework Schema (RDF-Schema) syntax for its applicability to describing resource metadata and the implications for storage and retrieval from both hierarchical and relational data stores. The next two sections are background material: in Section 2, we will describe the core Resource Description Framework (RDF) concepts; we will then describe RDF-Schema in Section 3. In Section 4, a concrete Grid resource will be represented, and the applicability of RDF to both resource metadata and hierarchical and relational storage will be analyzed. Finally, Section 5 will summarize and suggest future directions of inquiry.

2.0 RDF

The Resource Description Framework (RDF)[7] is a World Wide Web Consortium (W3C)[11] Recommendation, which means that it is stable and normative. As it is a W3C product, it should not be surprising that RDF is focused on describing metadata on the World Wide Web (WWW). Fortunately, the heterogeneity of the web has forced the creators of RDF to produce a very general model and syntax, which can be examined without referring to Web pages at all. From [7]:

> The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain. The definition of the mechanism should be domain neutral, yet the mechanism should be suitable for describing information about any domain.

2.1 Data Model

RDF intends to represent named and valued properties of resources. This is essentially no different from attribute-value pairs or, in object-oriented (OO) terminology, instance variables of objects; it is also similar to entity-relationship (E-R) diagrams used for databases. The basic data model consists of four components:

- Resources: A *resource* is anything which can be named with a Universal Resource Identifier (URI). Because URI's are themselves extensible, this includes all Grid resources. For instance, all Grid object identifiers (OIDs) could be subsumed into a URI of the form *grid://oid/#.#.#.#.#*
- Literals: Atomic values, such as integers or strings. For the RDF syntax in XML, this is defined as any well-formed XML fragment (which is not further interpreted by the RDF processor).
- Properties: A *property* is a specific aspect, characteristic, attribute, or relation used to describe a resource. The set of properties is a subset of the set of resources, that is, properties are themselves resources. RDF-Schema defines properties in more detail.
- Statements: A *statement* is an ordered triple of (*predicate*, *subject*, *object*), where predicate is a property, subject is a resource, and object is a literal or a resource. These three parts of a statement are called, respectively, the *predicate*, *subject,* and *object*.

An RDF statement can be diagrammed as a directed graph by representing the subject and object by nodes, and representing the predicate as a directed arc pointing from the subject to the object. For example, a statement tuple (CPU-Utilization,grid://foobar.lbl.gov,0.82), which can be stated in English as "*grid://foobar.lbl.gov* has a *CPU-Utilization* of *0.82*", would be diagrammed like this:

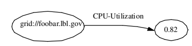

FIGURE 1. Diagram of Sample RDF Statement

The foundations for a system of resource types is laid by the predefined property RDF:Type, which can be used in statements like any other property, except that neither the subject nor the object can be a literal. The type system for literals is left up to the encoding, in the case of XML this means the XML-Datatypes[1] section of the XML Schema [9]definition. Relations between types is further elaborated in RDF-Schemas.

RDF also allows statements to be made about statements, for instance to represent thatthe statement above about CPU utilization came from the program *top*. To do this, RDF defines a mechanism for transforming a statement into a resource; this is called *reifying* the statement. Once a statement is reified, it has a URI and may be used like any other resource, for example: "*top* Measured [CPU-Utilization of *grid://foobar.lbl.gov* is 0.82]", where the reified statement is in square brackets. This is shown graphically in Figure2; note that the URI used for the reified statement is for illustrative purposes only, and is not constrained by the RDF specification.

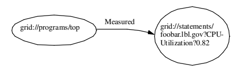

FIGURE 2. Diagram of an RDF Statement about a Reified Statement

2.2 Syntax

In addition to a data model, the W3C has defined a concrete syntax for representing RDF properties, resources, and statements using the Extensible Markup Language (XML)[3]. To be precise, there are two equivalent syntaxes: the *serialization syntax* and the *abbreviated syntax*. The latter syntax is simply a more compact form than the former, but they are otherwise equivalent and can be mixed freely. All the examples in this document will be in the simpler and clearer serialization syntax.

Both syntaxes use XML Namespaces [8], which like programming language namespaces (e.g. Java package names) provide a means of associating a set of things with a unique identifier. In the examples that follow, all the elements defined by the RDF syntax will be indicated with "*rdf:*" preceding the element name, and all other element prefixes of the form "*prefix:*" are invented for this document.

The example statement about host CPU utilization, "*CPU-Utilization* of *grid://foobar.lbl.gov* is *0.82*", diagrammed in Figure 1, would be represented thus in the serialization syntax:

```
<rdf:RDF>
    <rdf:Description ID="foobar"
     about="grid://foobar.lbl.gov">
        <g:CPU-Utilization>0.82</g:CPU-Utilization>
    </rdf:Description>
</rdf:RDF>
```

The 'Description' element relates to a resource, whose URI is provided in the 'about' attrbute. Each child element of the description is a property. In this simple case, there is only one property 'CPU-Utilization' with the literal value '0.82'. However, in general the property may itself be a resource (and its value may also be a resource). For instance, the CPU utilization could be represented by a "CPU-Measurement" resource, which in turn has properties UserTime and SystemTime. An XML representation of this would be:

```
<rdf:RDF>
    <rdf:Description about="grid://foobar.lbl.gov">
        <g:CPU-Utilization>
            <g:Description about="grid://schemas/hw/CPU-measurement">
                <g:UserTime>0.21</u:UserTime>
                <g:SystemTime>0.61</u:SystemTime>
            </g:Description>
        </g:CPU-Utilization>
    </rdf:Description>
</rdf:RDF>
```

Finally, it should be noted that RDF provides the property `rdf:type` to indicate that a resource belongs to a set of resources, or *class*. Both the subject and object of a statement using `rdf:type` must be a resource (not a literal). For example, to represent that "foobar.lbl.gov" is a type of "computer":

```
<rdf:RDF>
    <rdf:Description about="grid://foobar.lbl.gov">
        <rdf:type>
            <g:Description about="grid:/classes/hw#Computer"/>
```

```
        </rdf:type>
    </rdf:Description>
</rdf:RDF>
```

2.3 Further information

For the purposes of this discussion, all the advanced features of RDF syntax have been elided. Further details can be found at [7] and a good tutorial by Tim Bray is at [1].

3.0 RDF-Schema

The RDF data model provides a simple and elegant framework for describing the properties of resources. However, RDF does not provide a mechanism for declaring the properties, or for defining the relationships between these properties and other resources. Thus, while RDF can represent that CPU utilization is the property of the computer *foobar*, there is no standard vocabulary for stating that CPU utilization is a type of hardware measurement which applies to all computers. This "basic *type system* for use with RDF models"[5] is what RDF-Schema provides.

3.1 Data model

The RDF-Schema type system uses the RDF element *rdf*:type, discussed in Section 2.2. If resource A has a type of resource B, then A is said to be an instance of the class B. The core classes defined by RDF-Schema are:

- rdfs:Resource - RDF resources
- rdf:Property - RDF properties
- rdfs:Class - Generic concept of type or category.

The core properties defined by RDF-Schema are:

- rdfs:subClassOf - A transitive subset/superset relationship between classes (resources which have an rdf:type of rdfs:Class). A class should not, directly or indirectly, be a subClassOf itself.
- rdfs:subPropertyOf - A transitive subset/superset relationship between properties (subset of resources with an rdf:type of rdf:Property). A property should not, directly or indirectly, be a subPropertyOf itself.
- rdfs:seeAlso - Indicates further information related to a resource.
- rdfs:isDefinedBy - A subproperty of rdfs:seeAlso, which indicates a URI which defines the resource (in addition to the URI of the resource itself).

In order to specify which properties belong with which resources, RDF-Schema uses *constraints* on the classes of subjects and objects that a predicate (property) may be applied to. In a constraint, the subject is called the *domain* of the property and the object is called the *range*. This is different from most programming languages. For example, most OO languages could have a class called Computer will have an instance variable Utilization of type Number; in RDF-Schema this would be instead a pair of resources (Computer,Number) and a property, Utilization, which had a domain of Computer and a range of Number.

One consequence of this approach is that the properties do not “belong” to the class. Rather, anyone can add a property to a class by creating a constraint that includes the class in its domain.

The core constraint resources defined by RDF-Schema are:

- rdfs:ConstraintProperty - Superclass of all properties representing constraints.
- rdfs:range - Instance of rdfs:ConstraintProperty that specifies the range of a property. A property may have at most one *range* property. The domain of rdfs:range is rdf:Property, and the range of rdfs:range is rdfs:Class.
- rdfs:domain - Instance of rdfs:ConstraintProperty that specifies the domain of a property. A property may have zero or more *domain* properties. The domain of rdfs:domain is rdf:Property and the range of rdfs:domain is rdfs:Class
- rdfs:ConstraintResource - Used to represent a resource involved in a constraint, this element allows new constraint resources to be identified and used, thereby providing an extensibility mechanism.

3.2 Syntax

Because RDF-Schema is itself an instance of RDF, the syntax is also defined by the RDF XML syntax. All the elements specified above -- classes, properties, and constraints -- can be represented directly in RDF. To show this, consider the statement that “CPU utilization is a measurement that applies to all computers”, posed as a capability of RDF-Schema but not (the standard elements of) RDF in Section 3.0. The Computer class could be represented as a subclass of Resource:

```
<rdf:RDF>
    <rdf:Description about=”grid://classes/hw” ID=”Computer”>
        <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:subClassOf
         rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    </rdf:Description>
</rdf:RDF>
```

and the CPU utilization property could be represented as a property which has Computer as its domain and Number as its range:

```
<rdf:RDF>
    <rdf:Description ID=”cpu-utilization”>
        <rdf:type
         resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
        <rdfs:domain resource=”grid://classes/hw#Computer”>
        <rdfs:range
         resource=”http://www.w3.org/2000/03/example/classes#Number"/>
    </rdf:Description>
</rdf:RDF>
```

RDF-Schema does not specify how a schema processor should interpret schema syntax, but anticipated uses include “a validator will look[ing] for errors, an interactive editor [that] might suggest legal values, and a reasoning application [that] might infer the class and then announce any inconsistencies” (from [5], Section 3).

Some of the details of RDF-Schema (such as elements for comments and documentation) have been left out for brevity; for more information, refer to the specification[5].

4.0 Analysis

The analysis of the applicability of RDF-Schema will focus on two requirements: (1) the facility with which it can represent Grid resources, and (2) the facility with which these representations could be stored in a hierarchical or relational database. As we shall see, the second question is difficult and open-ended, so in this section will only introduce the issues that future documents and discussions will need to grapple with at length.

4.1 Representing Resources

The inheritance model of RDF-Schema makes it well-suited to the representation of Grid resource metadata. Any named and distinct Grid entity can be an instance of a class of resources, which can in turn be arranged into superclasses and subclasses. Many compute resources, by the very nature of the engineering process, have well-defined characteristics and share concepts and operational modes. For instance, the very general class “storage” could have “disk”, “tape”, and “network” as subclasses. The “disk” class of storage could have various protocols such as “SCSI” and “IDE”. One advantage of this hierarchy is that a schema processor which only knows about the general class of “storage” could still infer some useful information from an instance of the class “Ultra-wide SCSI II”, namely that it stores data. Similarly, a base property of “free space” might be subclassed to describe various types of disks, memory, or archives while still providing useful information at the highest level.

Not all Grid resources will fit neatly into a hierarchy, and in these cases a combination of multiple inheritance, to incorporate elements of two different hierarchies, or composition, to rearrange finer-grained components of resources or properties in diverse ways, can be used. For the purposes of this document, the important point is that all these modes are supported by the RDF-Schema model and syntax.

4.2 Example

Some trivial examples were presented to explain RDF-Schema Section 3.0, but in order to evaluate applicability to real Grid resources some non-trivial resources also need to be represented. In this section, we will consider a fuller example: resource allocation for a scheduler.

4.2.1 Allocation Example Overview

The properties and resources in this example are based on a working draft submitted to the scheduling working group [6], which provides the following list of “properties” that describes a resource allocation by a Grid scheduler:

- Consumer: identity

- Authorization: class, priority
- Resource manager: identity
- Resource: type, device identifier
- Period: start time, end time
- Capacity: transactional, storage
- Parallelism: hard, soft
- Notification: event mask, signalling mode
- Resource-specific:
  - processors: program invocation
  - network: network path
  - disk: access pattern

In order to model this in RDF-Schema, we will represent the "properties" above as both a class of rdf:Resource and a rdf:Property. A diagram illustrating this for the "Period" property of scheduler allocation is shown in Figure 3.

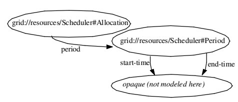

FIGURE 3. Diagram of Period Property of Scheduler Allocation Class

The exceptions to this procedure are the Capacity and Parallelism properties. In this case, the sub-properties have an "is-a" relationship with their respective parents (e.g.: storage capacity "is-a" type of capacity), so instead Capacity and Parallelism are each modeled as an rdf:Property and extended using the *subPropertyOf* element.

For simplicity's sake, the secondary properties will be left as opaque resources. Note that many of these would be either literals or generic, and therefore would have existing syntaxes which could just be referred to by a URI.

The last category, "resource-specific", suggests subclassing, so the class of Allocation will be divided into three sub-classes: ProcessorAllocation, NetworkAllocation, DiskAllocation.

4.2.2 Allocation Example RDF-Schema

In the XML below, the namespace for RDF is “rdf” and the namespace for RDF-Schema is “rdfs”. The default namespaces for the document is *grid://Scheduler*.

For readability, the Description element for an RDF-Schema Class uses an “rdfs:Class” element, which is a legal syntax abbreviation; similarly, the Description of an RDF Property uses an “rdf:Property” element.

```
<rdf:RDF xml:lang="en"
    xmlns=”grid://Scheduler”
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

    <!-- Allocation class and subclasses -->
    <rdfs:Class rdf:ID="Allocation">
        <rdfs:comment>Allocation resource superclass</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    </rdfs:Class>
    <rdfs:Class ID="ProcessorAllocation">
        <rdfs:comment>Allocation for use of processor(s)</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#Allocation"/>
    </rdfs:Class>
    <rdfs:Class ID="DiskAllocation">
        <rdfs:comment>Allocation for use of disk(s)</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#Allocation"/>
    </rdfs:Class>
    <rdfs:Class ID="NetworkAllocation">
        <rdfs:comment>Allocation for use of network link(s)</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#Allocation"/>
    </rdfs:Class>

    <!-- Top-level attributes -->
    <rdfs:Class rdf:ID=”Consumer”/>
    <rdf:Property ID=”consumer”>
        <rdfs:domain rdf:resource=”#Allocation”/>
        <rdfs:range rdf:resource=”#Consumer”/>
    </rdf:Property>
    <rdfs:Class ID=”Authorization”/>
    <rdf:Property ID="authorization">
        <rdfs:domain rdf:resource=”#Allocation”/>
        <rdfs:range rdf:resource=”#Authorization”/>
    </rdf:Property>
    <rdfs:Class rdf:ID=”ResourceManager”/>
    <rdf:Property ID="resourceManager">
        <rdfs:domain rdf:resource=”#Allocation”/>
        <rdfs:range rdf:resource=”#ResourceManager”/>
    </rdf:Property>
    <rdfs:Class rdf:ID=”Resource”/>
    <rdf:Property ID="resource">
        <rdfs:domain rdf:resource=”#Allocation”/>
        <rdfs:range rdf:resource=”#Resource”/>
    </rdf:Property>
    <rdfs:Class rdf:ID=”Period”/>
    <rdf:Property ID="period">
        <rdfs:domain rdf:resource=”#Allocation”/>
```

```
    <rdfs:range rdf:resource="#Period"/>
</rdf:Property>
<!-- Capacity and Parallelism are
    really just superclasses, so they are
    included in the next section
-->
<rdfs:Class rdf:ID="Notification"/>
<rdf:Property ID="notification">
    <rdfs:domain rdf:resource="#Allocation"/>
    <rdfs:range rdf:resource="#Notification"/>
</rdf:Property>
<rdfs:Class rdf:ID="Program"/>
<rdf:Property ID="program">
    <rdfs:domain rdf:resource="#ProcessorAllocation"/>
    <rdfs:range rdf:resource="#Program"/>
</rdf:Property>
<rdfs:Class rdf:ID="NetworkPath"/>
<rdf:Property ID="networkPath">
    <rdfs:domain rdf:resource="#NetworkAllocation"/>
    <rdfs:range rdf:resource="#NetworkPath"/>
</rdf:Property>
<rdfs:Class rdf:ID="DataAccessPattern"/>
<rdf:Property ID="dataAccessPattern">
    <rdfs:domain rdf:resource="#DiskAllocation"/>
    <rdfs:range rdf:resource="#DataAccessPattern"/>
</rdf:Property>

<!-- Sub-properties, modeled as properties of the
      resource classes.
-->
<rdf:Property ID="identity">
    <rdfs:domain rdf:resource="#Consumer"/>
    <rdfs:domain rdf:resource="#ResourceManager"/>
</rdf:Property>
<rdf:Property ID="class">
    <rdfs:domain rdf:resource="#Authorization"/>
</rdf:Property>
<rdf:Property ID="priority">
    <rdfs:domain rdf:resource="#Authorization"/>
</rdf:Property>
<rdf:Property ID="resourceType">
    <rdfs:domain rdf:resource="#Resource"/>
</rdf:Property>
<rdf:Property ID="startTime">
    <rdfs:domain rdf:resource="#Period"/>
</rdf:Property>
<rdf:Property ID="endTime">
    <rdfs:domain rdf:resource="#Period"/>
</rdf:Property>
<rdf:Property ID="eventMask">
    <rdfs:domain rdf:resource="#Notification"/>
</rdf:Property>
<rdf:Property ID="signallingMode">
    <rdfs:domain rdf:resource="#Notification"/>
</rdf:Property>

<!-- Capacity -->
<rdf:Property ID="capacity">
    <rdfs:domain rdf:resource="#Allocation"/>
```

```
    </rdf:Property>
    <rdf:Property ID="transactional">
        <rdfs:subPropertyOf rdf:resource="#capacity"/>
    </rdf:Property>
    <rdf:Property ID="storage">
        <rdfs:subPropertyOf rdf:resource="#capacity"/>
    </rdf:Property>

    <!-- Parallelism -->
    <rdf:Property ID="parallelism">
        <rdfs:domain rdf:resource="#Allocation"/>
    </rdf:Property>
    <rdf:Property ID="hard">
        <rdfs:subPropertyOf rdf:resource="#parallelism"/>
    </rdf:Property>
    <rdf:Property ID="soft">
        <rdfs:subPropertyOf rdf:resource="#parallelism"/>
    </rdf:Property>

</rdf:RDF>
```

Although RDF-Schema syntax is, in its raw form, quite verbose, because it is valid XML there exist many tools to render it easier to generate and read. For example, loading this schema's XML into a recent version of Microsoft's Internet Explorer™ will show, in addition to some syntax coloring, each nested XML tag with a [-] or [+] symbol in front of it, which can be used to collapse/expand the element.

4.3 Efficient Storage and Retrieval

Schemas can, in theory, be stored as XML documents and with the use of XML parsers, be used programmatically. However, for many purposes this is far too inefficient and unstructured. Most application needs would, however, be met by either a hierarchical directory service or a relational database that provided access to the schema's metadata. This section will make a preliminary assessment of the steps needed to store an RDF-Schema in these two services.

4.3.1 Hierarchical Directory Service

The most common method of indexing Grid resources, at present, is a directory service such as those which comply with the Lightweight Directory Access Protocol (LDAP) [10]. In a directory service, the data is modeled as a tree. At each "level" in the tree, the value of some set of attributes associated with each node distinguishes it uniquely from its siblings. Thus, each node in the tree can be found by a unique combination of attribute names starting from the root of the tree.

In order to illustrate the transformations necessary to move from an RDF-Schema model to a subtree in a directory service, we will turn again to our scheduler allocation example. For brevity's sake, we will model a subset of the entire schema: the ProcessorAllocation subclass and its associated period, transactional capacity, and program invocation properties. A diagram of the subset is shown in Figure4. This subset is still non-trivial: it includes inheritance of both the "primary" resource, Allocation, and of a property of the resource, Capacity, and another resource, Period, which has more than one property.

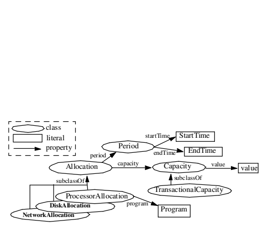

FIGURE 4. Subset of Scheduler Allocation Schema

Because LDAP does not have true inheritance, it is necessary to explicitly include all the properties of a node's superclasses in the node itself. This can be done by "walking" the inheritance tree (i.e. the "subclassOf" relationships) from a given resource to the root, accumulating properties as we go. For example, the ProcessorAllocation subclass will have, in addition to the "program" property, all the properties (and property sub-trees) which have as their domain its superclass, Allocation.

The tree of properties and sub-properties will expand into nodes and sub-nodes in the directory service. When the range of a property is a literal value, the node represented by the domain of the property will have a literal-valued attribute, e.g., the "Period" node will have a literal "StartTime" attribute because the range of the "startTime" property is a literal. And when the range is another resource, the domain node will have a sub-node which represents that resource, and so on. Multiple instances of properties will result in either multiple sub-nodes or multiple literal values.

Application of these principles to the schema in Figure 4 to a simple LDAP schema which places scheduler allocation instances under users (represented here as Alice and Bob), is shown in Figure 5. It is asserted here that the necessary transformation from the XML syntax of the RDF-Schema, to the schema syntax used by LDAP could be automated, using one of a number of XML tools.

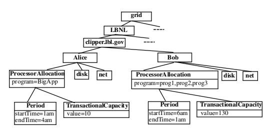

FIGURE 5. LDAP Schema with Scheduler Allocation

4.3.2 Relational Database

An automated mapping from RDF to a relational database is also certainly possible, the only question is how to do it efficiently. The answer to this, in turn, depends partially on what type of scalability and efficiency you need from the resulting database.

Rather than try to answer this question here, we will simply refer the reader to several proposed relational schemas (expressed as SQL statements) summarized at http://www-db.stanford.edu/~melnik/rdf/db.html ; one of the more straightforward yet seemingly complete schemas can be found in full at http://uxn.nu/wraf/RDF-Service/doc/rdf.sql.

5.0 Conclusions and Future Directions

In this document the basics of both RDF and RDF-Schema were presented, then an analysis of its applicability was performed, including a sample syntax for a scheduler's allocation entity and a brief discussion about the storage of RDF and RDF-Schema in both hierarchical and relational types of databases.

RDF seems to be perfectly able to represent Grid resources, and offers a number of compelling reasons to consider its use for this purpose. First, as an existing standard, it offers the promise of interoperability with the commercial world. RDF is rapidly becoming the commercial standard for maintaing meta-data for the growing web services market, and is supported by Microsoft, Netscape/AOL, IBM, etc. Second, the generality of the framework should allow the representation of both current and future Grid meta-data, independent of the underlying mechanisms -- such as LDAP or relational databases -- used to store and retrieve it. Separating the the data modeling problem from the storage and retrieval problem will allow our models to be used with a variety of underlying technologies; all that will be required is a mapping between RDF and the underlying technology, e.g., RDF to LDIF for the current generation Grid Information Service. Third, because of the growing use of RDF in the commerical world, there are a large number of RDF specific tools to help generate, manipulate, query, and reason about RDF formatted meta-data. While this argues for the consideration of RDF, more examples and detailed discussions about storage in various types of databases are needed in order to make an informed decision about RDF's future utility for representing Grid information.

6.0 References

[1] P. Biron, A. Malhotra, eds.,"XML Schema Part 2: Datatypes", W3C Working Draft 25 February 2000. http://www.w3.org/TR/xmlschema-2

[2] T. Bray, "What is RDF?", 24 January 2001. http://www.xml.com/pub/a/2001/01/24/rdf.html

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, eds., "Extensible Markup Language (XML) 1.0", W3C Recommendation, 10 February 1988. http://www.w3.org/TR/REC-xml

[4] T. Bray, D. Hollander, A. Layman, eds.,"Namespaces in XML", W3C Recommendation, 14 January 1999. http://www.w3.org/TR/REC-xml-names

[5] D. Brickley, R. V. Guha, "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation 27 March 2000. http://www.w3.org/TR/rdf-schema

[6] K. Czajowski, "Describing Grid Allocations", GWD-Sched-3-1, Global Grid Forum Scheduling Working Group (http://www.cs.nwu.edu/~jms/sched-wg/)

[7] O. Lassila, R. Swick, eds., "Resource Description Framework (RDF) Model and Syntax", W3C Recommendation, 22 February 1999. http://www.w3.org/TR/REC-rdf-syntax

[8] A. Malhotra, N. Sundaresan, "RDF Query Specification", W3C Query Languages Workshop, 3 December 1998. http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html

[9] H. Thompson, et. al., eds.,"XML Schema Part 1: Structures", W3C Working Draft 25 February 2000. http://www.w3.org/TR/xmlschema-1

[10] M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", IETF RFC 2251, December 1997.

[11] World Wide Web Consortium (W3C). http://www.w3.org/