GOSv1: Defining Schemas for the Grid Information Services

A proposal to be discussed as part of the Gridforum

Version: 1.0
Date: 05/12/2000
Internal Number: GWD-GIS-001

Gregor von Laszewski and Peter Lane

Mathematics and Computer Science Division, Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439 U.S.A.

Steve Fitzgerald

University of Southern California Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292-6695
U.S.A.

Brett Didier and Karen Schuchardt

Battelle, Pacific Northwest National Laboratory
Richland, WA 99352
U.S.A.

Remark: please note that the HTML versions does not display the images, we recomend to use the PS or PDF version of this document.

Comments: Please send comments to gregor@mcs.anl.gov
Editor: Gregor von Laszewski

Contents

1 Scope

The Grid Object Specification (GOS) format defines formal syntax for the definition of objects that form the core of the Grid Information Services. A formal syntax enables the definition of objects shared by a large number of participants. Furthermore, it allows the creation of automated tools for manipulating the objects. The GOS format also documents policy issues and provides a detailed semantic description of the information to the reader.

The format builds on the syntax developed as part of the Globus Metacomputing Directory Service (MDS) project [1, 4] and has its roots in LDAP terminology [9]. A prototype implementation has been developed as part of the CoG project [5].

The rest of this specification is structured as follows. First, we use an example to demonstrate the basic syntax. Then we introduce the detailed grammar. Finally we present a set of tools for manipulating the specifications.

2 Example

We begin with a simple example intended to provide an intuitive overview of the GOS format while including all features of the format. In this example we define an object that represents a compute resource. Note that the example in this document does not represent a complete and proper definition of the compute resource object. A complete proposal for such an object can be found elsewhere [7].

2.1 Objects

```
Grid::ComputeResource OBJECT-CLASS ::= {
  SUBCLASS OF Grid::PhysicalResource
  RDN = hn (hostName)
  CHILD OF {
    Grid::organizationalUnit,
    Grid::organization,
  }
  MUST CONTAIN {
    canonicalSystemName :: cis,
    manufacturer :: cis, single,
    model :: cis, single,
    serialNumber :: ces , single,
  }
  MAY CONTAIN {
    diskDrive :: dn, multiple,
  }
}
```

Each object should have an appropriate description that explains its usage, the attributes, their types and the intended use. For example, we augment the attribute serialNumber with the following explanation:

Attributes

serialNumber: (ces) The serial number is used to distinguish the compute resource from other products of the manufacturer or vendor.

...

Our object is associated with the namespace *Grid* and has the name ComputeResource. The object contains attributes that are grouped in a MUST CONTAIN and

MAY CONTAIN sets. As the names indicate, these groups define the validity of an object based on its attributes. Thus, if the attribute *serialNumber* is not defined in an object instantiation, the object would not be valid. Each attribute is identified by a name, a type, and a rule if the attribute can have multiple values (appear multiple times). In our case we see attributes for distinguished names, and for case-sensitive, and case-insensitive strings [8].

2.2 Aggregates

Next, we extend our example to include information related to geographical coordinates. Since such information can also be useful for other objects, we would like to be able to define an aggregation of attributes that can be included in other objects. We name such a specialized object class an *aggregate class*. Aggregate classes help to keep the definition of the objects quite small while referring to previously defined aggregates that can be reused in multiple objects. Additionally, they are used to structure related attributes.

> Note: Some Information Services do not allow use of multiple inheritance. In order to have an easy way of expressing this, aggregate objects are useful.

```
Grid::Location AGGREGATE-CLASS ::= {
  MUST CONTAIN {
    static :: cisboolean, single
  }
  MAY CONTAIN {
    latitude :: cisfloat, single
    longitude :: cisfloat, single
    locationName :: cis
  }
}
```

The description of the attributes could look like the following:

Attributes

static:	(boolean) true if the location of the object is fixed and can be associated with latitude and longitude values, and false if the object is moving.
locationName:	(cis) e.g., state/city/.../floor/room/desk/...
longitude:	(cisfloat) relative east-west geographical position
latitude:	(cisfloat) relative north-south geographical position

Aggregate objects are not used by themselves, but they are used to include all MUST CONTAIN and MAY CONTAIN attributes in an object in which they are referred. As shown next, we extend our example, the ComputeResource object, to include the aggregate object Grid::location.

```
Grid::ComputeResource OBJECT-CLASS ::= {
  SUBCLASS OF Grid::PhysicalResource
  RDN = hn (hostName)
  CHILD OF {
    Grid::organizationalUnit,
    Grid::organization,
  }
  AGGREGATES {
```

```
    Grid::Location
  }
  MUST CONTAIN {
    canonicalSystemName :: cis,
    manufacturer :: cis, single,
    model :: cis, single,
    serialNumber :: ces , single,
  }
  MAY CONTAIN {
    diskDrive :: cis, multiple,
  }
}
```

Besides the MUST CONTAIN and MAY CONTAIN groups we have an AGGREGATES and a CHILD OF group. The AGGREGATES group includes all objects which are "inlined" into the primary objects; that is, all attributes of the aggregate objects are placed appropriately in the MAY or MUST CONTAIN groups. A few restrictions have to be kept in mind. First, the type, the multiplicity, and the presence (MAY CONTAIN or MUST CONTAIN) of attributes must be consistent in case an attribute is defined with the same name in multiple aggregate objects. In our case, the primary object Grid::ComputeResource would end up in an object with the following attributes:

```
Grid::ComputeResource OBJECT-CLASS ::= {
  SUBCLASS OF Grid::PhysicalResource
  RDN = hn (hostName)
  CHILD OF {
    Grid::organizationalUnit,
    Grid::organization,
  }
  MUST CONTAIN {
    canonicalSystemName :: cis,
    manufacturer :: cis, single,
    model :: cis, single,
    serialNumber :: ces , single,
    static :: cisboolean , single
  }
  MAY CONTAIN {
    diskDrive :: cis, multiple,
    latitude :: cisfloat , single,
    longitude :: cisfloat , single,
    locationName :: cis
  }
}
```

Here, the CHILD OF group indicates that the Grid::ComputeResource object can be stored logically in an Grid::organizationalUnit or and Grid::organization. The RDN, or resources description name, is a convenient abbreviation that is associated with the object. A primary object can be a SUBCLASS of another primary object. In our case the Grid::ComputeResource object inherits all attributes that are defined by the CHILD OF, AGGREGATE, MUST CONTAIN, and MAY CONTAIN groups of the Grid::PhysicalResource.

Next, we define the grammar more formally.

3 Grammar

The grammar is specified in Extended Backus Naur Form (EBNF). The following symbols are used to describe the grammar:

Usage	Semantics
$(< a >)$def	0 or 1 occurrences of a
$(< a >)$+	1 or more occurrences of a
$(< a >)$*	0 or more occurrences of a
"T"	terminal
"#"	comments character

To simplify the grammar, multiple blanks between terminal symbols are not addressed. It is assumed that multiple blanks will be reduced to one.

3.1 Grid Object Specification

Each file following the GOS format contains one or more objects.

```
<gos> ::= (<class>)+
```

3.2 Class

A class can be either an object or an aggregate class.

```
<class> ::= (<objectclass> | <aggregateclass>)
```

3.3 Object Class

The rule for *object* is defined below. The objectdefinition comprises several constructs. Note that an object in the schema may or may not be instantiated as a primary object. Primary objects will contain constructs such as RDN, CHILD OF, and AGGREGATES, whereas aggregate classes will not. Each construct is defined following the grammar specification.

```
<objectclass> ::= <objectname> "OBJECT-CLASS" "::=" "{"
             (SUBCLASS OF <objectname>)def
             (RDN = <attributename>  "("
             <attributename>  ")")def
             (CHILD OF {
                (<objectname>;)+
             })def
             ( "AGGREGATES "  "{"
             (<aggregatename>  ",")(*)
              "}")
             ( "MAY CONTAIN "  "{"
             (<attribute>  ",")(*)
              "}")
             ( "MUST CONTAIN "  "{"
             (<attribute>  ",")(*)
              "}"
              "}")
```

3.4 Object and Aggregate Names

The object and aggregate names are a simple concatenation of the namespace and the object name with the "::" character as deliminator.

```
<objectname> ::= <namespacename> ``::'' ( <name> | <objectname> )
<aggregatename> ::= <name>
```

3.5 Aggregate Class

The rule for *aggregates* is defined below. An aggregate contains only the attributes defined under MAY CONTAIN and MUST CONTAIN. Aggregates do not use subclassing or inheritance (see Section 3.10).

```
<aggregateclass> ::= <objectname> "AGGREGATE-CLASS" "::=" "{"
             ( "MAY CONTAIN "  "{"
             (<attribute>  ","){*}
              "}")
             ( "MUST CONTAIN "  "{"
             (<attribute>  ","){*}
              "}"
              "}")
```

3.6 Namespace

Namespaces are used to group a set of object definitions together in order to avoid a name collision. Two special namespaces exist at the moment: *X.500*, which indicates objects that are derived from the X.500 object specification, and *Grid*, which indicates objects that are building the core grid objects. Each application can define its own namespace. Nevertheless, since projects as part of the Grid activities may use the same abbreviation, we recommend registering the name with the Grid Information Services working group. A document on this issue will be prepared in the future.

Namespaces are defined as part of the objectname. The rule for a namespace token is

```
<namespacename> ::= <name>
```

Each namespace has a special object called <namespace>::top. This object is inherited by all objects of this namespace.

3.7 Attributes

Attributes are defined in the following way:

```
<attribute> ::= <attributename>  ``::'' <type>
<attributename> ::= <name>
```

To be discussed: If an attribute is defined by an extended schema, the name is proceeded by the "ext" terminal. The editor is of the oppinion this is part of the implementation detail. **Note:**

```
<attributename> ::= <name> | ``ext'' <name>
```

A complete list of LDAP types is defined in [2]. For our purposes we are mostly interested in the following types:

```
<type>  ::= ``cis'' | ``ces'' | ``dn'' | ``bin'' |
             ``int'' | <cis type>
```

We decided to include some additional types to make it easier for users to decide on the semanticals, though these types are in reality of type *cis*. In future versions of LDAP these types may be represented directly.

```
<cis type>  ::= ``cisfloat'' | ``cisdate'' | ``cisboolean''
```

- cisfloat – This type represents floating-point numbers formatted as strings (e.g. 0.02, 12.5). Currently we do recommend not to use the normed exponetial form.
- cisdate – This type represents the date in a way similar to the output of date -u except that the day of the week is not kept and the month is provided as an integer value (e.g. 4 20 20:00:34 GMT 1999)
- cisboolean – this type represents Boolean values (e.g., “true” or “false”).

3.8 Basic Tokens

As basic token, we define names, characters, and numbers. The “...” in the character and number definition carry the usual “range-of” connotation.

```
<name> ::= <character> (<character> | <number>){*}
<character> ::= "a" | ... | "Z"
<number> := "0" | ... | "9"
```

Each of the major constructs of the grammar are now defined. Some of these are defined in literature. Others have been added to communicate policy information regarding the structure of the Directory Information Tree (DIT) or how objects are to be instantiated.

3.9 Comments

All characters starting from the character “#” until the end of the line are considered to be part of a comment. A comment can belong to an object or an attribute.

We distinguish

```
<comment> := "#" <asciicharacter>*
<attributecomment> := <comment>
<objectcomment> := <comment>
```

3.10 Definitions and Their Relevance to LDAP

It is useful to understand the relationship between terms used in the grammar and terms used while describing the concepts of LDAP. Although the grammar introduced above is a general information model, it provides some nice features that allow it to be maped easily into an information model based on LDAP terminology. As we discuss in [6], we can map the same model into an XML description.

SUBCLASS OF Objects can be derived from each other to provide data inheritance. This concept is referred to in LDAP as subclassing or object class inheritance. An object that subclasses another object inherits all of the required and optional attributes of the superclass. The special abstract class “top” is inherited by all objects. This class contains at least the required attribute “objectclass” to guarantee that each object contains at least one object class. Similarly we define the class "Grid::Top" which is inherited by most objects. Note that LDAP does not support multiple inheritance, as defined in object-oriented languages such as C++. Nevertheless, because of the support of aggregation, an aggregate object can be a member of more than one instantiated object class.

CHILD OF The "CHILD OF" construct is used to define a policy regarding the possible location(s) of the object in the DIT. An object that is specified as a "CHILD OF" a class or classes can be placed in the DIT only under objects of the specified type. This construct represents policy information only and is not in any way enforced. The intent of this policy is to keep a well-structured tree where objects can be readily found. This attribute is defined only for primary objects.

RDN As specified in [9], each component of a distinguished name is called a relative distinguished name (RDN). Since each RDN is unique, it guarantees that each entry in the LDAP directory has a unique name. The RDN field in the notation introduced in this paper specifies only the last component of the distinguished name which is assembled by the position in the DIT. This attribute is defined only for primary objects.

AGGREGATES The "AGGREGATES" construct is used to define the complete set of objects that are created together to form the primary object. As mentioned previously, LDAP does not support multiple inheritance. However, through the use of aggregation, the primary object will include the attributes of all the aggregate objects. Aggregate objects are not allowed to be structural objects; that is, they cannot be instantiated directly as objects in the DIT. An example of an aggregate object is Grid::Location. It acts as an aggregate of several objects such as Grid::PhysicalResource when the latter is instantiated and thus will supply its attributes to the primary object.

MAY CONTAIN Attributes which are listed under MAY CONTAIN are optional attributes. When an object is uploaded to the LDAP server and schema checking is enabled, no error is returned if the attribute is missing in an object instantiation.

MUST CONTAIN Attributes which are listed under MUST CONTAIN are required attributes. When an object is uploaded to the LDAP server and schema checking is enabled, an error is returned if the attribute is missing in an object instantiation.

4 Citation of Attributes

A convenient way to cite attributes in other documents is to use a string-based token, which is based on the namespace, the class, and the attribute. We use the following short form:

```
<namespace>::<objectclass name>:<attribute name>
```

Thus, the string

```
Grid::sotware:description
```

would refer to the attribute description that is part of the object software in the namespace Grid.

5 Conversion Tools

The purpose of Grid Object Specification is to provide a common basis for developing tools that rely on the existence of "Grid" objects. The relationship between the format specification, the output generated by a translation tool, and the application using this output is illustrated in Figure 1. An ASCII representation of this figure can be found in Appendix B.

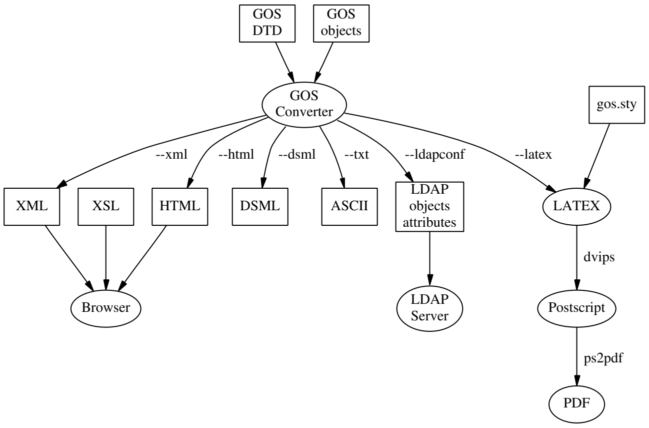

Figure 1: *The relationship between the tools and the specification files*

The advantage of separating the format specification from the actual implementation is to be able to use a variety of tools that allow the automatic generation of data structures and other useful but tedious tasks. A conversion tool should provide the following output formats:

- Generate "at" and "oc" files, which are used at startup time of the LDAP server to initialize the schema. Often these two files are referred to in the slapd.conf file.
- Generate a set of HTML pages describing the GOS for convenient browsing of the specification. The URL to this page is defined in http://www.globus.org/mds.
- Transform the LDAP based grammar into an XML based document and vice versa.

To date, we have developed one prototype tool called *gosConverter*. This tool provides the following conversion options:

–input

stores the LDAP based specification of GOS. The default file is called "grid-info-objectspecification.defs".

–html

translates the input GOS schema definition to a set of HTML pages that can be viewed with a web browser.

–mdsml

translates the input GOS schema definition to mdsml, a XML language that is an extension to DSML.

–dsml

translates the input GOS schema definition to DSML.

–ldapconfig

translates the GOS schema two files, which store the attributes and the object classes thar are used to initialize the slapd daemon.

Attributes File

The attributes are written into the file "grid-info-attributes.def." This file is a slapd.ac.conf file. To be consistent, we suggest renaming this file in future releases to "grid.slapd.at.conf". An example for such a file is shown below.

```
attribute o                        cis
attribute ou                       cis
attribute aci                      ces
attribute objectName               cis
attribute ttl                      cis
attribute userPassword             bin
attribute o                        cis
attribute ou                       cis
attribute email                    cis
attribute phone                    cis
attribute facsimiletelephonenumber cis
attribute gridadministrator        cis
attribute seeAlso                  dn
```

Object Class File

The attributes are written into the file "grid-info-objects.def." This file is a slapd.oc.conf file. To be consistent, we suggest renaming this file in future releases to "grid.slapd.oc.conf". An example if this file is shown below.

```
objectclass GridTop
    requires
        objectclass,
        lastupdate
    allows
        aci,
        objectName,
        ttl

objectclass GridMDSAdministrator
    requires
       objectclass,
       lastupdate,
       userPassword,
       cn
    allows
       commonName,
       seeAlso,
       o,
       ou,
       aci,
       objectName,
       ttl
```

A LaTeX Binding

For convenience, it is possible to define the Grid Object Classes with the help of a LaTeX file. A conversion tool is available that can translate the LaTeX file into an XML object definition. Other manipulation tools are available that can be used to create from the XML objects a standard LaTeX description, a Web page, ASCII text, or a simple browser.

To use this convenient way of specifying the objects the simple example given below shows how.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\GRIDOBJECT{GridObjectExample}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

The object {\em GridObjectExample} is used to ... .

%------------------------------------------
\ begin{verbatim}
GridObjectExample  OBJECT-CLASS ::= {
  SUBCLASS OF GridTop
  RDN = cn (commonName)
  CHILD OF {
     organization;
     organizationalUnit;
  }
  MUST CONTAIN {
     mustAttribute              :: cis ::single,
  }
  MAY CONTAIN {
     mayAttrubuteA              :: cis,
     mayAttrubuteB              :: cis,
  }
}
\ end{verbatim}

%------------------------------------------
\begin{ATTRIBUTES}{AAAAAAAAAA}

\item[mustAttribute] (cis,single)
        This attribute stores ...

\item[mayAttributeA] (cis)
        This attribute stores ...

\item[mayAttributeB] (cis)
        This attribute stores ...

\end{ATTRIBUTES}
```

B ASCII Graph Notation

The tools graph in this paper is produced with dot [3].

```
digraph tools {

XML [shape=box];
XSL [shape=box];
HTML [shape=box];
DSML [shape=box];
ASCII [shape=box];
LDAP [shape=box,label="LDAP\nobjects\nattributes"];
GOSDTD [shape=box,label="GOS\nDTD"];
CONVERTER [label="GOS\nConverter"];
GOS [shape=box,label="GOS\nobjects"];
GOSSTY [shape=box,label="gos.sty"];
LDAPSERVER [label="LDAP\nServer"];

GOS         -> CONVERTER;
GOSDTD      -> CONVERTER;
CONVERTER   -> XML         [label="--xml"];
CONVERTER   -> DSML        [label="--dsml"];
CONVERTER   -> HTML        [label="--html"];
CONVERTER   -> ASCII       [label="--txt"];
CONVERTER   -> LDAP        [label="--ldapconf"];
XML         -> Browser
XSL         -> Browser
HTML      -> Browser
LDAP        -> LDAPSERVER
CONVERTER   -> LATEX       [label="--latex"];
GOSSTY      -> LATEX       ;
LATEX       -> Postscript  [label="dvips"];
Postscript -> PDF          [label="ps2pdf"];
}
```

References

[1] The Globus Object Class Definitions. http://www-isi.globus.org/mds/OCBrowser/globus_object_defs.html.

[2] HOWES, T., KILLE, S., YEONG, W., AND ROBBINS, C. *The String Representation of Standard Attribute Syntaxes*, Mar. 1995. RFC 1778.

[3] KOUTSOFIOS, E., AND NORTH, S. C. *Drawing Graphs with dot*, 1.5 ed. AT&T Labs - Research, Murray Hill, NJ, March 1999.

[4] STELLING, P., FOSTER, I., KESSELMAN, C., C.LEE, AND VON LASZEWSKI, G. A Fault Detection Service for Wide Area Distributed Computations. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing* (July 1998), pp. pp. 268–278.

[5] VON LASZEWSKI, G. The Globus CoG Web pages. http://www.globus.org/cog.

[6] VON LASZEWSKI, G., , AND LANE, P. An XML binding for the Grid Object Specification. draft, Gridforum Information Services Working group, Feb. 2000.

[7] VON LASZEWSKI, G., AND FITZGERALD, S. Representing Compute Resources. draft, Gridforum Information Services Working group, Feb. 2000.

[8] WAHL, M., COULBECK, A., HOWES, T., AND KILLE, S. *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*, Dec. 1997. RFC 2252.

[9] YEONG, W., HOWES, T., AND KILLE, S. *X.500 Lightweight Directory Access Protocol*, July 1993. RFC 1487.

Index