GOSv3: A Data Definition Language for Grid Information Services

Gregor von Laszewski, Mike Helm, Steven M. Fitzgerald, Pete Vanderbilt, Brett Didier, Peter Lane, Martin Swany
Information Services
Request for Comments: GWD-GIS-011-11
Obsoletes: GWD-GIS-001
Category: informational

Filename: http://www.mcs.anl.gov/gridforum/gis/reports/gos-v3/gos-v3.2.html

Scope

The Grid Object Specification (GOS) language is a data definition language (DDL) [1] for Grid Information Services (GIS). The GOS is intended to provide a very simple mechanism for describing entities used within Grids and grid-based applications. It allows for the elementary definition of such entities so that information sharing is encouraged amongst users, applications, and services. Additionally, GOS allows the GGF to maintain in a simple manner a large number of specifications suggested by various independent groups and virtual organizations as part of the GGF standardization process.

The GOS format builds on the syntax developed as part of the Globus Metacomputing Directory Service (MDS) project [2,7,6] and has its roots in LDAP terminology [11,9,8]. The format, however, has evolved into a very simple but generic form. Thus, it is easy to generate parsers that can translate GOS in various formats including ASN.1, LDAP rfc2252 syntax, XML, HTML, and SQL tables. Therefore, GOS provides an ideal starting point for defining entities as used in Grids. The scope of GOS does currently not define either a data manipulation language (DML) [1] or a data model [1].

Contents

1 General Issues

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [15]. In places where "LDAP" is used we mean, "LDAPv3". All LDAP specific constructs are based on LDAPv3 as defined by RFC2251 and supporting RFCs.

2 Introduction

The Grid Object Specification (GOS) language allows one to specify entities used within Grids via namespaces, object classes and attributes. In this document we refer to a specification following the GOS as a *GOS schema*. A namespace encapsulates a set of object classes, while an object class encapsulates a set of attributes that are (typically) related. An attribute is a datum that is comprised of a name and a value of a given type. Attributes can be multivalued. The GOS syntax provides additional constructs (a) to

augment namespaces, object classes, and attributes with descriptive comments that are intended to describe their purpose and usage (b) to simplify the maintenance of a large number of definitions in a heterogeneous fashion amongst many object class designers. This additional information contains, for example, an Object Identifier (OID) field to uniquely identify object classes and attributes [13,10]. GOS does not specify the physical representation of entities or attribute values in this revision. This representation is implementation specific and is outside the scope of this specification.

2.1 Example

In this example we provide an intuitive overview of the GOS format while including many features of the format (Figure 1). We define a fictitious *object class* that can be used to define an entity representing a compute resource within a Grid.[1]. Each GOS schema contains a number of clauses. Each of these clauses is introduced by a keyword. Though keywords are case insensitive, capitalization may be used to increase readability. The following clauses are used in the definition of the ComputeResource: **namespace**, **objectclass, oid, description, must contain, may contain**. Two additional keywords are used as part of the type definitions of attributes: **single-valued** and **multiple-valued**. GOS does not define the instantiation of objects instead it only provides the structure of the objects similar to the use of IDL within CORBA.

NAMESPACE ggf **{**

DESCRIPTION {
This GOS schema describes objects as used by the GGF
}

NAMESPACE gis-wg **{**

DESCRIPTION {
Objects within this namespace are defined by the GIS working group
}

OBJECTCLASS ComputeResource **{**
OID { 1.3.6.1.4.1.6757.2.2.3.22 **}**
DESCRIPTION {
A computational resource such as a computer
}
MUST CONTAIN {

hostname :: **single-valued**, cis,
{ *The hostname of the machine as defined in DNS* }

[1] Note that the example in this document does not represent a complete nor a proper definition of a compute resource entity.

```
        canonicalSystemName :: single-valued, cis,
                { The model of the computer as obtained by for example sysinfo
                  or info-guess. };

        manufacturer:: single-valued, cis,
                { The manufacturer of the compute resource }


        model :: single-valued, cis,
                { The model of the compute resource }

        serialNumber :: single-valued, numeric,
                { The serial number of the compute resource  }
      }
      MAY CONTAIN {
        diskDrive :: multiple-valued, ces,
                { Description of the disk drive available};
      }

  }
}
```

Figure 1: Example of a fictitious GOS schema defining a compute resource in GOS.

2.2 Elements of a GOS schema

A GOS schema contains a number of clauses each of which we explain in more detail in this section. A GOS schema defines namespaces containing a number of objects that are used to group a number of related attributes.

2.2.1 NAMESPACE

Each object specification must be encapsulated in a namespace. In our example we have chosen a namespace for the Grid Information Services Working Group which is part of the Global Grid Forum namespace. Using namespaces allows the easy maintenance of a large number of definitions provided by the numerous groups contributing the Grid Forum community. Typically each group is responsible to maintain its own namespace.

2.2.2 OBJECTCLASS

Within each namespace are object specifications, called object classes, that define the structure of a number of entities. Each object class definition contains a list of attributes. In our example we define an object class representing a set of compute resources.

2.2.3 OID

Each object class may be assigned an optional object identifier, which is called an OID [10]. In case the GOS schema is intended to be part of an LDAP based implementation strategy OIDs shall be defined. We recommend that an object class definition should not be changed without having a new OID assigned. This OID number uniquely identifies a particular object class and attribute definition. We believe that these OID numbers should be assigned by the Grid Forum to object class and attribute definitions, which have been sanctioned by the Grid Forum. (How to obtain a Grid Forum specific OID is subject to debate within the GIS-WG).

2.2.4 DESCRIPTION

Each object class must contain a description clause. This clause provides for additional documentation, which typically includes a concise explanation or a statement of purpose of the object class and its usage. Each attribute must have also a description associated with it. For better readability the keyword DESCRIPTION is omitted from the definition of attributes.

2.2.5 MUST CONTAIN

The MUST CONTAIN clause introduces the set of attributes that are **required** in each and every entity. In the example, we have included four such attributes. The ``serialNumber'' is of type ``numeric,'' the others are of type case-insensitive string. Each of these attributes may appear only once in an entity. This is denoted by keyword ``single-valued.'' A list of single-valued attributes may be chosen as KEY to attempt to uniquely identify an entity by a subset of its attribute values.

2.2.6 MAY CONTAIN

It is also possible to include optional attributes with each entity. These attributes are defined via the MAY-CONTAIN clause. In this example we see that attribute ``diskDrive'' may be associated with a particular ComputeResource. Since a computer may contain several disk drives, we would like to define an entity that contains any number of diskDrive attributes. The ``multiple-valued'' keyword is used to denote that the corresponding attribute can be used zero or more times.

2.2.7 Attribute Syntax

Currently, a set of basic attribute syntaxes or types is supported. These attribute syntaxes are described in more detail in rfc2252. Elementary types include: numeric, cis, ces, numeric, and boolean. Additional types may be supported by implementations of GOS, but must be clearly documented.

2.2.8 Inheritance

As in other object oriented languages inheritance provides a simple way of extending objects with other prior defined object classes . If we want to associate geographical

information with each compute resource, we can use an object class representing the Location and inherit them in our ComputeResouce object class.

```
NAMESPACE ggf {

   …

   NAMESPACE gis-wg {

      OBJECTCLASS  Location {
        DESCRIPTION {
            This object class defines a set of
             attributes to define the geographical
             location of an entity
         }
          MUST CONTAIN  {
              isStationary  ::single-valued, boolean;
                      { if true the computer is stationary such as a desktop };
          }
          MAY CONTAIN  {
              locationName ::  multiple-valued, cis,
                       { the name of the location such as the room number}
              latitude :: single-valued, numeric,
                       { the latitude of the location }
              longitude :: single-valued, numeric,
                       { the longitude of the location }
              altitude :: single-valued, numeric,
                       { the altitude of the location }
          }

      }
      OBJECTCLASS ComputeResource {
          INHERITS FROM { ggf.gis-wg.Location; ggf.gis-wg.ComputeResource }
          … < see Figure 1 >
       }

                  …
   }
}
```

Figure 3: Example of inheritance as used in a fictitious GOS schema defining a compute resource

3 Grammar

The grammar for the GOS is presented in this section via a set of productions represented in Extended Backus-Naur Form (EBNF) as defined in RFC822. We assume that lexical analysis is first performed on the input stream, which yields a stream of terminals. The terminals are grouped into three classes: keywords words, delimiters, and tokens. Additionally, white spaces and comments are consumed by the lexical analysis phase. Comments are delimited by the sharp (#) character. Each character after a sharp till the next occurrence of a newline is considered to be a comment. .

3.1 Keywords

All keywords are case insensitive. By convention, however, we typically use uppercase letters for keywords that introduce clauses and lowercase letters otherwise. Within this document, we have placed keywords in bold font to aid readability. The set of keywords include:

> ALIAS, AUXILIARY, DESCRIPTION, INCLUDE, INHERITS FROM, KEY, KIND, multiple-valued, OBJECT CLASS, OID, MAY CONTAIN, MUST CONTAIN, NAMESPACE, single-valued, STRUCTURAL

3.2 Delimiters

A number of delimiters or punctuation characters are defined to aid in the readability and to simplify parsing. These delimiters include:

> + , ; ‘ “ { } :: #

3.3 Tokens

All other terminals are defined as one of three tokens: *string*, *oid-number*, and *name.* These tokens are defined by the following regular expressions:

Token	Regular Expression	*Example*
string	Text as defined in rfc822 with the exception that the use of “{“ and “}” is masked with “\{“ and “\}”	This is a string
oid-number	`([0-9]+\.)*[0-9]+`	125.3.4.5
name	`[a-zA-Z][-a-zA-Z0-9]*`	ComputeResource

3.4 Includes

At times it is desirable to create schemas dependent on the inclusion of other schemas. In order to allow this kind of schema maintenance the include statement can be used that provides the mechanism for inclusion of schemas. The include statement is optional feature of GOS. Includes may be nested but not circular. The implementer is required to give a clear specification of error conditions upon failure of includes.

```
include = INCLUDE absoluteURI
```

where the absoluteURI is defined in RFC2396 [17].

INCLUDE file://home/laszewski/schema-a
INCLUDE http://home/laszewski/schema-b

Figure 4: The include statement allows to include schemas located at various URIs.

3.5 Productions

A single GOS file may contain zero or more namespace definitions followed by one or more object class definitions

```
gos = *( include ) 0#1( namespace )
namespace = NAMESPACE name “{“
                    0#1( OID “{“ oid-number “}”)
                    objects-and-attributes
                “}”
 objects-and-attributes = *( include )  / *( namespace ) / ( 1#(
object-def ) *(attribute-def) )

If an OID is specified within a namespace all OIDs defined within this
namespace will have a prefix of the OID for the namespace, e.g. the
OIDs will be concatenated.
```

3.5.1 Namespaces

Namespaces provide a simple way to organize schemas. Objects and attributes defined within a namespace are distinct from objects and attributes with the same name defined within another namespace. Namespaces can be nested.

NAMESPACE ggf-gis-wg {

DESCRIPTION {
This namespace provides the entities defined by the Grid Information Services working group.
};

```
    …

    NAMESPACE people {

    }

    NAMESPACE  computers {

    }
  }

  NAMESPACE ggf-perf-wg {

    DESCRIPTION {
        This namespace provides the entities defined by the Grid Information Services
        working group.
    }

    …


  }
```

Figure 4: .An example of the use of multiple namespaces.

3.5.2 Object Class Definitions

An object class provides a set of constraints on entities. In particular, it specifies which attributes must appear or may appear in an entity. The clause defining a class definition is defined as follows:

```
     class-name = name
     class-def = OBJECTCLASS class-name “{“
                                 OID “{“ oid-number “}”
                                 DESCRIPTION “{“ string+ “}”
                                 0#1{ KIND “{“ AUXILIARY  | STRUCTURAL
“}”}
                                 0#1( KEY “{“ key-list “}” )
                                 0#1( inheritance-spec )
```

```
                                0#1( MUST CONTAIN "{" attribute-list
"}" )

                                0#1( MAY CONTAIN "{" attribute-list "}"
)

                        "}"
```

We explain in more detail each of the productions.

3.5.2.1 KIND

To simplify a translation from GOS to the LDAP data model we have included a KIND clause, that allows the specification of the KIND of an object class as specified in RFC 2252. We restrict the use of KIND to the keywords AUXILIARY and STRUCTURAL. We do not support ABSTRACT or any restriction on the use of AUXILIARY classes within structural object classes. In general we do not recommend the use of this keyword as the inheritance clause more easily expresses an object oriented data definition.

3.5.2.2 KEY

To simplify the translation of GOS to the LDAP, we have included a KEY clause. In LDAP an entity may be uniquely identified via a subset of attributes that are part of its definition. The collection of attributes that are used to identify an entity uniquely is known as its primary key. The KEY clause is used to specify the primary key. If this clause is not provided, we assume that there is no uniform use of a single primary key. The productions for key-list and attribute-name and are as follows:

```
key-list  = attribute-name *( ',' attribute-name )
attribute-name = name
```

3.5.3 Inheritance

Object class definitions may contain an inheritance specification of the form

```
inheritance-spec = INHERITS FROM "{" class-name-list "}"
class-name-list = class-name *( "," class-name )
```

When object class B inherits from class A, then any instance of B has all the properties attributed to it by both A and B. In particular, any instance of B must have all the required attributes of both A and B and may have optional attributes from both A and B. The terms ``superclass" and ``subclass" may be used to describe the inheritance relation: if object class B inherits from class A then A is a superclass of B and B is a subclass of A. If an object class definition for B has more than one name in the INHERITS FROM clause, then B has all the named classes as superclasses and instances of B must have all the properties attributed to it by B and all its superclasses.

In general, if a superclass has a particular attribute specification, the subclass should not have a specification for the same attribute. If it does, there are some rules:

- If the superclass has the attribute in its MUST CONTAIN clause, so should the subclass. (If the superclass specifies MAY CONTAIN clause, the subclass can ``promote" it to MUST CONTAIN (or KEY).)
- If the superclass specifies the attribute as **single-valued**, the subclass must do the same. (If the superclass species **multiple-valued**, the subclass may specify **single-valued**.)
- The type of an attribute in a subclass must be the same as in the superclass.

3.5.4 User Defined Types or Attribute Syntax Definitions

An attribute syntax describes the format and interpretation of a class of attribute values. The name “cis” in Example 1 names a syntax whose values are case insensitive strings. The predefined set of attribute types are the LDAPv3 set of syntaxes. For a complete list, RFC2252 and RFC 1778 must be consulted. Examples of predefined attribute syntax names include: **binary, bitstring, Boolean, case exact string (ces), case insensitive string (cis), objectclass, numeric.**

A user, however, might want to define a new attribute syntax independent of any object-class definition. This can be accomplished by defining a new attribute based upon an existing attribute. This new attribute can be either an alias of or derived from the original attribute.

Within the GOS, there are two possible productions that can be used to define a new attribute:

```
attribute-def =

        ATTRIBUTE name “{“
                ALIAS OF “{“ name “}”
                 DESCRIPTION “{“ string + “}”
                ”}”

        /

        ATTRIBUTE name “{“
                OID “{“ oid-number “}” “;”
                0#1( DERIVED FROM “{“ name “}”  )
                DESCRIPTION “{“ string + “}”
        ”}”
```

The first production defines a new attribute syntax as an alias. Within this production, there is no OID clause. An attribute syntax that is an alias for another attribute syntax is not assigned a new OID; it simply uses the OID assigned to the original attribute syntax.

The second production defines a new attribute syntax that has either a new semantic meaning or a unique representation. In either case, a unique OID must be assigned to this attribute syntax. The OID is assigned to the attribute syntax via the OID clause. An attribute can be derived from an existing attribute similar to an ALIAS with this difference: ALIAS will provide the same OID, the DERIVED FROM attributes will require a new OID.

This revision of GOS does not offer a means to identify new syntaxes or data types or manipulation rules for LDAP or other information stores. A future revision may address this need.

3.5.5 Attribute Lists

Each entity contains a set of attributes. Within the GOS the set of attributes associated with an objectclass is defined via the production:

```
attribute-list = 1#( attribute-spec ; )
attribute-spec = name " "::" opt-modifier name "{" string "}"
opt-modifier = ( ( single-valued / multiple-valued ) "," )
```

Each attributed contains its type and an optional modifier. The attribute type must be one of the predefined attributes. The optional modifier indicates whether or not the attribute can be used more than once within an entity. The ``single-valued" modify indicates that the attribute can only be used within an entity, whereas the ``multiple-valued" modify indicates that the attribute can be used any number of times within an entity. If the modifier is not present, the attribute can be used only once within an entity.

4 Authors' Addresses

Steve Fitzgerald
Department of Computer Science
California State University, Northridge
18111 Nordhoff Street
Northridge, CA 91330-8281 phone: (818) 677-3314
fax: (818) 677-2140
e-mail: Steven.Fitzgerald@ecs.csun.edu

Gregor von Laszewski
Mathematics and Computer Science Division
9700 South Cass Avenue
Argonne National Laboratory
Argonne, IL 60439, U.S.A.
phone: (630) 252 0472
fax: (630) 252 5986
e-mail: gregor@mcs.anl.gov

Peter Lane
Mathematics and Computer Science Division
9700 South Cass Avenue
Argonne National Laboratory
Argonne, IL 60439, U.S.A.
e-mail: lane@mcs.anl.gov

Martin Swany
Department of Computer Science
University of Tennessee
1122 Volunteer Blvd
Knoxville, TN 37996-3450
phone: (865) 974-6758
fax: (865) 974-4044
e-mail: swany@cs.utk.edu

Mike Helm
50a-3111 Lawrence Berkeley Lab.
Berkeley, CA 94720 U.S.A.
phone : (510) 486-7248
fax: (510) 486-6712
e-mail: helm@es.net

Pete Vanderbilt
e-mail: pv@nas.nasa.gov

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

Acknowledgement

We like to thank the GIS working group for their many helpful comments and contributions. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid program.

Copyright

Bibliography

1

The Globus Object class Definitions.
http://www-isi.globus.org/mds/OCBrowser/globus_object_defs.html.

2

An Introduction to Database Systems, sixth ed.
Systems Programming. Addison-Wesley, 1995.

3

CROCKER, S.
The LDAP Data Interchange Format (LDIF), June 2000.
RFC 2849.

4

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

5

IANA.
OID enterprise numbers.
http://www.isi.edu/in-notes/iana/assignments/enterprise-numbers.

6

VON LASZEWSKI, G., FITZGERALD, S., DIDIER, B., AND SCHUSCHARDT, K.
Defining Schemas for the Grid with Gos, the Grid object specification.

Gridforum Working Group Document GIS-WG 1, Argonne National Laboratory and Pacific Northwest Laboratory, June 2000.
http://www.gridforum.org.

7

VON LASZEWSKI, G., FITZGERALD, S., FOSTER, I., KESSELMAN, C., SMITH, W., AND TUECKE, S.
A Directory Service for Configuring High-Performance Distributed Computations.
In *Proc. 6th IEEE Symp. on High-Performance Distributed Computing* (?? 1997), pp. pp. 365-375.

8

WAHL, M.
A Summary of the X.500(96) User Schema for use with LDAPv3, Dec. 1997.
RFC 2256.

9

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

10

WAHL, M., COULBECK, A., HOWES, T., AND KILLE, S.
Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions, December 1997.

11

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

12

GOOD, G.
The LDAP Data Interchange Format (LDIF), June 2000.
ftp://ftp.isi.edu/in-notes/rfc2849.txt.

13

Chadwick, D.
Understanding X.500 The Directory,
International Thompson Computer Press, 1996

14

ITU, THE DIRECTORY: MODELS, X.501 (08/97), 1997

[15] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[16] Standard for the format of ARPA internet text message, RFC822, August 13, 1982.

[17] Berners-Lee, Fielding, Masinter, Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, IETF, 1998.

5 Appendix

In this appendix, we present examples of the ComputeResource structural object defined using two other DDLs: RFC226 syntax and SQL.

5.1 Structured Query Language Format

Structured Query Language (SQL) is a fourth-generation language that contains many dialects. SQL defines both a DDL and a DML for the relational data model. The DDL is used to define the structure of tables and relationships between these tables.

The GOS format can be converted into this specific form. For example, the following SQL based definition could be generated directly from the ComputeResource. Note that a secondary table is created to ensure that data is kept within an appropriate normal form, e.g., 3NF. This additional table, ComputeResource-diskDrive also defines a relationship, via a foreign key, with the ComputeResource. This is necessary to ensure the consistency of the database.

```
 CREATE DOMAIN  hostname            CHAR(*)
 CREATE DOMAIN  canonicalSystemName CHAR(*)
 CREATE DOMAIN  manufacturer        CHAR(*)
 CREATE DOMAIN  model               CHAR(*)
 CREATE DOMAIN  serialNumber        NUMERIC(*)

 CREATE TABLE GridComputeResource
 (
   hostname            DOMAIN(hostname),
   canonicalSystemName DOMAIN(canonicalSystemName),
   manufacturer        DOMAIN(manufacturer),
   model               DOMAIN(model),
   serialNumber        DOMAIN(serialNumber),

   PRIMARY KEY (hostname),
);

CREATE TABLE GridComputeResource-diskDrive
(
   hostname            DOMAIN(hostname),
   diskDrive           DOMAIN(diskDrive),

   PRIMARY KEY (hostname),
   FOREIGN KEY (hostname) REFERENCES ComputeResource
);
```

5.2 RFC2256 Syntax

Many LDAP based directory servers utilize the RFC 2256 syntax to define both attributes and object classes. New data definitions must be loaded into these servers prior to the storage of entities defined by these new definitions. The GOS format can be convert into this specific form. In fact, the Globus project provides such a translator.

A possible translation of the ComputeResource structural object class is presented below without explanation.

```
(  1.3.6.1.4.1.6757.2.3.1 NAME 'hostname' UP NAME
   EQUALITY caseIgnoreMatch
   SUBSTR caseIgnoreSubstringsMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
   SINGLE-VALUE
)

(  1.3.6.1.4.1.6757.2.3.1 NAME 'canonicalSystemName'
   EQUALITY caseIgnoreMatch
   SUBSTR caseIgnoreSubstringsMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)

(  1.3.6.1.4.1.6757.2.3.1 NAME 'model'
   EQUALITY caseIgnoreMatch
   SUBSTR caseIgnoreSubstringsMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)

(  1.3.6.1.4.1.6757.2.3.3 NAME 'serialNumber'
   EQUALITY numericStringMatch
   ORDERING numericStringOrder
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.36
)

(  1.3.6.1.4.1.6757.2.3.2 NAME 'diskDrive'
   EQUALITY caseExactMatch
   SUBSTR caseExactSubstringsMatch
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
)

( 1.3.6.1.4.1.6757.2.2.3.22 NAME 'GridComputeResource'
  SUP top
  DESC "A computational resource such as a computer"
  MUST ( hostname $ canonicalSystemName $  manufacturer
         model    $ serialNumber        )
  MAY ( diskDrive )
)
```