GOSv2: A Data Definition Language for Grid Information Services

1 Scope

The Grid Object Specification (GOS) language is a data definition language (DDL) [2] for Grid Information Services (GIS). The intent of this language is to specify the definition of object classes. These object classes are used to specify the contents of entities that pertain to the Grid and grid-based applications. The GOS allows for a precise definition of such entities so that information sharing is encouraged amongst users, applications, and services.

The GOS format builds on the syntax developed as part of the Globus Metacomputing Directory Service (MDS) project [1, 7, 6] and has its roots in LDAP terminology [11, 9, 8]. The format, however, has evolved into a generic form. The generic form allows for the construction of automated translators that can generate other implementation specific forms, e.g., LDAP rfc2256 syntax and SQL syntax. We do not define either a data manipulation language (DML) [2] nor a data model [2].

2 Introduction

The GOS allows one to specify entities used within the Grid via object-class and attribute definitions. An object class is simply a set of attributes that are (typically) related, and an attribute is single datum that is comprised of a name and a value of a given type. The syntax also provides constructs to include additional relevant information.

The additional information is necessary (a) to better understand the intent and purpose of the object class and (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, a description and an Object IDentifier (OID) field. An OID is a unique identifier assigned to each unique definition of an object class and an attribute [10].

Within the GOS, we distinguish three types of object classes: abstract, structural, and auxiliary. These object classes enable the definition of an entity to be defined easily and allows for the definition to be extended at a later time. In general, every entity contains an abstract class (e.g., “GridTop”), at least one structural object class, and zero or more auxiliary object classes. Each of the object class definition defines a set of attributes or fields associated with an entity of the corresponding type.

Attributes are defined in one of two ways. First, they can be defined as an aliases of an existing attribute. These attributes are defined as part of an object class definition. Second, they can be defined within a separate syntactic construct. The definition of attributes do not include physical representation. Such representation is implementation specific, as such are defined elsewhere.

3 Introductory Example

We begin with a simple example intended to provide an intuitive overview of the GOS format while including most features of the format. In this example we define an object class that represents a compute resource, known as Grid-ComputeResource. Note that the example in this document does not necessarily represent a complete nor a proper definition of a compute resource entity.

Each object class definition contains a number of clauses. Each of these clauses is introduced by a keyword. The GridComputeResource’s definition includes six clauses: OID, DESCRIPTION, STRUCTURAL, KEY, MUST CONTAIN, and MAY CONTAIN. (By convention we use uppercase for keywords that introduce a clause within the GOS.)
GridComputeResource OBJECT CLASS

OID 1.3.6.1.4.1.6757.2.2.3.22
DESCRIPTION "A computational resource such as a computer"

STRUCTURAL
KEY hostname :: single, cis;
MUST CONTAIN {
    canonicalSystemName :: single, cis;
    manufacturer :: single, cis;
    model :: single, cis;
    serialNumber :: single, numeric;
}

MAY CONTAIN {
    diskDrive :: multiple, ces;
}

Figure 1: Example of a structural object class

OID
Each object-class definition is assigned an object identifier, which is called an OID [10]. An object class definition should not be changed without having a new OID assigned.1 This OID number uniquely identifies a particular definition. We believe that these OID numbers should be assigned by the GridForum to object-class and attributes definitions, which have been sanctioned.

DESCRIPTION
Each object class contains a description clause. This clause provides for additional documentation which typically include a concise explanation or a statement of purpose of the object class.

STRUCTURAL
Recall that there are three types of object classes: abstract, structural, and auxiliary. The GridComputeResource is a structural object class. Each entity that is defined must contain one structural object class. The entity’s definition can be extended by the inclusion of either abstract object classes, auxiliary object classes, or both.

KEY
Each entity is comprised of a number of attributes. One or more of these attributes can be used to uniquely identify an entity. In most cases a single attribute serves this role. In the GridComputeResource, the value of the “hostname” attribute uniquely identifies an entity. Additional information about the attribute “hostname” is provided after the “::” symbol. In this example, we see that hostname may be used as an attribute at most once and its a case-insensitive string, or “cis.”

Within LDAP, which follows the hierarchical data model, the key is also known as the Relative Distinguished Name (RDN). The RDN coupled with the an entity’s position in the DIT (Directory Information Tree) provides unique “name” of the entity. Within the relational model the key is known as the primary key or candidate key.2 The primary key, couple with the object class (or the table) name uniquely names the entity.

1Here we have assumed that the GridForum has assigned the OID 1.3.6.1.4.1.6757.2.2.3.22 to the GridComputeResource object class; The number 1.3.6.1.4.1.6757 has been assigned to the GridForum by IANA[5] for the purpose defining unique OID numbers.

2There may exist several candidate keys for an entity of which only one is deemed the primary key.
MUST CONTAIN

In addition to the attribute(s) associated with the key, an entity may contain a number of additional attributes. 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.”

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 that attribute “diskDrive” may be associated with a particular GridComputeResource. Since a computer may contain several disk drives, we would like to define an entity that contains any number of diskDrive attributes. The “multiple” keyword is used to denote that the corresponding attribute can be used zero or more times.

Example Instance of a GridComputeResource

An entity of type GridComputeResource can be represented in many ways. Here we provide an example in LDIF format[3], which is comprised of a number of attribute-value pairs.

    dn: hostname=burns.csun.edu, o=Grid
    objectclass: GridComputeResource
    hostname: burns.csun.edu
    canonicalSystemName: mips sgi irix 6.5
    manufacturer: SGI
    model: O2000
    serialNumber: 82164405
    diskDrives: /dev/root
              /dev/usr
              /dev/dsk/dks0d4s

To distinguish this entity from other entities, we have given it a unique textual identifier, which we call a distinguished name (or a DN). We have given this entity the name “hostname=burns.csun.edu, o=Grid.” This name is provided as the first line of the entity and is prepend with attribute name “dn”. The entity also has an additional attribute, objectclass, that is not defined via the GridComputeResource specification. This attribute defines the type of object class of the entity.

Auxiliary Object Classes

Auxiliary object classes are very important. Modifying the definition of an existing object class after-the-fact can be broadly disruptive. Auxiliary object classes provide a mechanism to extended the definition of entities that are used within the grid — without the need to modify all entities that share the same structural object class. This is similar in the ways in which subclass declarations within object-oriented languages can share the definition of a superclass.

Auxiliary object classes can also be helpful in the initial design phase of the types of entities. Attributes that are related can be group together. For example, we could define memory characteristics of a computer within a separate auxiliary class. Attributes can include: physical memory size, virtual memory size, available memory, access time, page size, and total swap space. Grouping these attributes together into a single object class is natural since they are related.

Alternatively, one could have associated these attributes with the original definition of the GridComputeResource object class. This makes the definition of this object class more complicated and less clear. The use of an auxiliary object class helps to keep the definition of structural object classes and other auxiliary object classes quite small and concise.

Consider the existing definition of the GridComputeResource object class. If we want to associate geographical information with each compute resource, we can use an auxiliary object class to extend the GridComputeResource
object class. Notice that such geographical information is not restricted to just compute resources, but can be used by many other types of entities. We can define an auxiliary object class that contains geographical information, and then associate this object class with many entities.

GridLocation OBJECT CLASS

OID 1.3.6.1.4.1.6757.2.2.3.23

DESCRIPTION "This object class defines a set of attributes to define the geographical location of an entity"

AUXILIARY

MUST CONTAIN {
   isStationary :: single, Boolean;
}

MAY CONTAIN {
   locationName :: multiple, cis;
   latitude :: single, float;
   longitude :: single, float;
   altitude :: single, float;
}

Figure 2: Example of an auxiliary object class

Notice that the format of this object class definition is similar to the definition of a GridComputeResources. The two major differences are the replacement of the STRUCTURAL keyword for the AUXILIARY keyword, and the removal of the KEY clause.

The AUXILIARY keyword may be followed by the name of a structural object class. This additional field restricts which structural object classes can associated with the current auxiliary object class. For example, this is useful for an auxiliary class that defines memory characteristics. We can indicate that the auxiliary class extends the definition of a compute resource, i.e., the object class GridComputeResource.

Without the addition field, no restriction is placed on the use of the auxiliary object class. In the case of the GridLocation, we have not placed any restriction. The GridLocation can be associated with any number of other structural classes.

An entity that is defined both by the GridComputeResource and GridLocation can be represented via LDIF as shown in Figure 3:

dn: hostname=burns.csun.edu, o=Grid
objectclass: GridComputeResource
objectclass: GridLocation
hostname: burns.csun.edu
canonicalSystemName: mips sgi irix 6.5
manufacturer: SGI
model: O2000
serialNumber: 82164405
diskDrives: /dev/root
diskDrives: /dev/usr
diskDrives: /dev/dsk/dks0d4s
isStationary: True
longitude: 135
latitude: 34

Figure 3: LDIF representation of an entity of type GridComputeResource and GridLocation
4 Grammar

The grammar for the GOS is presented in this section via a set of productions represented in Extended Backus Naur Form (EBNF). 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, whitespaces and comments are consumed by the lexical analysis phase. Comments are delimited by the sharp (#) character and a newline.

4.1 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 place keywords in bold font to aid readability. The set of keywords include:

<table>
<thead>
<tr>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>ALIAS</td>
</tr>
<tr>
<td>AUXILIARY</td>
</tr>
<tr>
<td>CLASS</td>
</tr>
<tr>
<td>CHILD</td>
</tr>
<tr>
<td>CONTAIN</td>
</tr>
<tr>
<td>DERIVED</td>
</tr>
<tr>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>FROM</td>
</tr>
<tr>
<td>KEY</td>
</tr>
<tr>
<td>multiple</td>
</tr>
<tr>
<td>OBJECT</td>
</tr>
<tr>
<td>OF</td>
</tr>
<tr>
<td>OID</td>
</tr>
<tr>
<td>MAY</td>
</tr>
<tr>
<td>MUST</td>
</tr>
<tr>
<td>single</td>
</tr>
<tr>
<td>STRUCTURAL</td>
</tr>
</tbody>
</table>

4.2 Delimiters

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

+ , :: ; ' " { }

4.3 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:

<table>
<thead>
<tr>
<th>Token</th>
<th>Regular Expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;string&gt;</td>
<td>('</td>
<td>&quot;'</td>
</tr>
<tr>
<td>&lt;oid-number&gt;</td>
<td>([0-9]+.)*[0-9]+</td>
<td>125.3.4.5</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>[a-zA-Z-*]*</td>
<td>GridComputeResource</td>
</tr>
</tbody>
</table>

4.4 Productions

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

<gos> ::= (<attribute-def>)* (<object-def>)+

4.4.1 Attribute Definitions

The object classes definitions provide a way to define new attributes. These attributes are essentially aliased to other pre-defined attributes. This is convenient because the definition is close to its use, and reduces the complexity of defining new attribute types. For example in Figure 1, we define an new attribute “diskDrive” that is of the same type as a “ces,” or a case exact string.

The “ces” attribute is an example of a predefined attribute. Currently, the predefined set of attributes are taken from LDAP attributes. For a full list of types defined for LDAP refer to RFC1778[4]. Examples of predefined attributes include:

- binary
- case exact string (ces)
- numeric
- bitstring
- case insensitive string (cis)
- objectclass
- Boolean
A user, however, might want to define a new attribute 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> ::= <name> ATTRIBUTE
DESCRIPTION <string> +
ALIAS OF <name>
```

```
| <name> ATTRIBUTE
OID <oid-number>
DESCRIPTION <string> +
( DERIVED FROM <name> )?
```

The first production defines a new attribute as an alias. Within this production, there is now OID clause. An attribute that is an alias for another attribute is not assigned a new OID; it simply uses the OID assigned to the original attribute.3

The second production defines a new attribute that has either a new semantic meaning or an unique physical representation. In either case, an unique OID must be assigned to this attribute. The OID is assigned to the attribute via the OID clause.

If the attribute has the same physical representation of an existing attribute, the DERIVED-FROM clause is used to indicate another attribute that shares the same physical representation – but has a different semantic meaning. If, however, the clause is omitted, the attribute has a unique physical representation. We do not provide a mechanism to define the physical representation, since it is implementation specific.

### 4.4.2 Attribute Lists

Each of the entity contains a set of of attributes. Within the GOS the set of attributes associated with an object class is defined via the `<attribute-list>` production. The production for an `<attribute-list>` is:

```
<attribute-list> ::= ( <attribute-spec> ; )+
<attribute-spec> ::= <name> "::" <opt-modifier> <name>
<opt-modifier> ::= ( ( single | multiple ) "," )?
```

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” modify indicates that the attribute can only be used within an entity, whereas the “multiple” 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.

### 4.4.3 Object Class Definitions

There are three types of object classes: abstract, structural, and auxiliary. Although one set of grammar rules can be used to describe all three object class types, we have broken them down into three separate sets of production rules, for simplicity.

```
<object-def> ::= <abstract-def>
| <structural-def>
| <auxiliary-def>
```

---

3 Attributes that are defined as part of an object class are considered aliases, and the appropriate `<attribute-def>` is implied.
Abstract Object Class

The abstract class is used to define a base set of attributes that can be associated with all entities. Each entity must include one of the defined abstract object classes.

```
<abstract-def> ::= <name> OBJECT CLASS
    OID <oid-number>
    DESCRIPTION <string>+
    ABSTRACT
        ( MUST CONTAIN "{{ <attribute-list> "}}" )?
        ( MAY CONTAIN "{{ <attribute-list> "}}" )?
```

Structural Object Class

The structural class is used to define the basic structure of an entity. The definition of an entity can be further defined via the auxiliary object classes.

```
<structural-def> ::= <name> OBJECT CLASS
    OID <oid-number>
    DESCRIPTION <string>+
    STRUCTURAL
        ( KEY <key-list> )?
        ( CHILD OF "{{ <name-list> "}}" )?
        ( MUST CONTAIN "{{ <attribute-list> "}}" )?
        ( MAY CONTAIN "{{ <attribute-list> "}}" )?
```

An entity must be uniquely identified via a subset of attributes that are part of its definition. The attributes that are used to uniquely identify an entity 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.

A structural object object may also define a CHILD-OF clause. This clause is helpful where an entity of this object class is named via the composition of a path, such as is done within a hierarchical database like LDAP. It defines the appropriate locations in which an entity can appear within the hierarchy. The positions are specified by the object-class name associated with an entity. This clause has been included primarily for documentation purposes.

The productions for `<key-list>` and `<name-list>` are as follows:

```
<key-list> ::= <attribute-spec> ('+' <attribute-spec> )* ':'

<name-list> ::= <name> ( ';' <name> )* ':'
```

Auxiliary Object Class

Recall that an auxiliary object class is used to extend the definition of an entity via the inclusion of more attributes. In structure, it is very similar to the abstract object class. The only difference is in the AUXILIARY clause.

```
<auxiliary-def> ::= <name> OBJECT CLASS
    OID <oid-number>
    DESCRIPTION <string>+
    AUXILIARY ( <name> )?
        ( MUST CONTAIN "{{ <attribute-list> "}}" )?
        ( MAY CONTAIN "{{ <attribute-list> "}}" )?
```

This clause has an option argument. This argument places a restriction on which structural object classes are associated with this auxiliary object. Without the addition field, no restriction is placed on the use of the auxiliary object.
5 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

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

6 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.

7 Copyright

Copyright (C) Grid Forum 2000. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Grid Forum or other Internet organizations, except as needed for the purpose of developing Internet standards, in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into.
Appendix

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

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 GridComputeResource. Note that a secondary table is created to ensure that data is kept within an appropriate normal form, e.g., 3NF. This additional table, GridComputeResource-diskDrive also defines a relationship, via a foreign key, with the GridComputeResource. 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 GridComputeResource
);
```

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 GridComputeResource 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 )
)

References


