GWD-R, GWD-I or GWD-C
Authors:

Sergio Andreozzi*, INFN
Stephen Burke, RAL
Felix Ehm, CERN
Laurence Field*, CERN
Gerson Galang, ARCS
David Horat (editor), CERN
Balazs Konya*, Lund University
Maarten Litmaath, CERN
Paul Millar, DESY
JP Navarro, ANL
*co-chairs

GLUE WG
http://forge.ogf.org/sf/sfmain/do/viewProject/projects.glue-wg

July 4, 2010

**GLUE v. 2.0 – Reference Implementation of an LDAP Schema**

## Status of This Document

This document provides information to the Grid community regarding the LDAP Schema implementation of the GLUE information model (v.2.0). Distribution is unlimited. This implementation is derived from the specification document "GLUE Specification v. 2.0". This document is a draft.



## Abstract

The GLUE specification is an information model for Grid entities described in natural language enriched with a graphical representation using UML Class Diagrams. This document presents an implementation of this information model as an LDAP Schema, and includes explanations of the major design decisions made during the implementation process.

Contents

## 1. Introduction

The GLUE 2.0 information model defined in [glue-2] is a conceptual model of Grid entities. In order to be used by Grid middleware an implementation in terms of a concrete data model is required.

This document describes the normative implementation of the GLUE 2.0 conceptual model in terms of an LDAP Schema. The approach followed to map the entities and relationships in the conceptual model to the concrete data model is also described.

## 2. Notational Conventions

The key words 'MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" are to be interpreted as described in RFC 2119 (see http://www.ietf.org/rfc/rfc2119.txt).

References to classes and attributes in the abstract schema are in *italic*, and to Object Classes and attributes in the concrete LDAP schema are in **bold**.

## 3. LDAP Schema Implementation

### 3.1 Approach

There are many possible approaches to realise the GLUE conceptual model as an LDAP Schema. The approach followed here is informed by practical experience with the LDAP implementation of the GLUE 1 schema [glue-1], and by general considerations relating to the efficiency and simplicity of likely queries. Conversely, the GLUE 2 schema itself was designed in the expectation that LDAP would be one of the main implementation technologies, and hence there is in many cases a natural way to translate the schema concepts into LDAP.

The GLUE LDAP implementation needs to map each entity in the GLUE information model to a specific LDAP entry defined in terms of Object Classes. We have chosen the most straightforward mapping in which there is a one to one correspondence between LDAP Object Classes and GLUE entities, with inheritance in the abstract schema represented explicitly by Object Class inheritance in LDAP.

In the following sections we discuss the detailed design decisions that have been made while converting the GLUE model into LDAP.

### 3.2 Prefix conventions

LDAP allows the same descriptor to refer to different object identifiers in certain cases and the registry supports multiple registrations of the same descriptor (each indicating a different kind of schema element and different object identifier). However, multiple registrations of the same descriptor are to be avoided if possible [rfc4520].

In practical experience with version 1 of the GLUE schema it has generally been the case that the schema does not need to coexist with other schemas, but it nevertheless seems useful to allow for this as a possibility. As Object Classes and attributes might have the same names in different schemas (there is only a global namespace), in order to make schemas compatible and able to coexist with other schemas in the same LDAP server we have decided that all Object Class and attribute names should be prefixed with a concrete string.

Given that GLUE 2.0 represents a major version change which may be required to cohabit with older versions for some time, **GLUE2** is used as a clean short prefix for all schema elements in the model – this compares with the prefix of **Glue** used for the version 1.x schemas.

## 3.3 Object Class and attribute naming conventions

The name of each LDAP Object Class is simply the name of the schema entity prefixed as described above, e.g. the Object Class representing the *Service* entity is called **GLUE2Service**. Each object-attribute pair in the abstract schema has a new attribute type in LDAP, with a name which is composed from the name of the object and the name of the attribute, for example **GLUE2ServiceType**. This gives a clear separation of attributes per Object Class, making it less prone to mistakes if changes are made and in the construction of queries.

## 3.4 Object Class types and inheritance

The abstract schema uses inheritance to derive some entities from others. LDAP is not object-oriented in the usual sense, but it allows inheritance to be represented explicitly by composing Object Classes [rfc4512]. However, it would also be possible to define standalone Object Classes including all inherited attributes directly. We have chosen to use explicit inheritance, both as the most natural representation of the schema and because it simplifies some queries. For example, it enables a generic query to be made for the *URL* attribute of every *Endpoint* without any special treatment for *Computing Endpoint*s, *Storage Endpoint*s or any other specialised classes which may be defined in the future. The main disadvantage of this approach is more complexity in the naming of attributes within an object, for example a **GLUE2ComputingEndpoint** object can include attributes called **GLUE2ComputingEndpointRunningJobs**, **GLUE2EndpointURL** and **GLUE2EntityName**, but in practice this seems unlikely to cause significant problems.

A separate case could be made for the *Entity* class, since it is unlikely that queries for the attributes of all objects will be common. In general we conclude that consistency both with the abstract schema and the general principles for the LDAP schema nevertheless make an explicit **GLUE2Entity** Object Class the best solution.

We have however made an exception for the *ID* attribute. All LDAP objects have an attribute which is used to construct its Distinguished Name (DN), and for the GLUE schema the natural attribute to use is clearly the *ID*. If we simply followed the rules described above the name of the *ID* attribute for every object would be **GLUE2EntityID**, and the DN of every object would be of the form **GLUE2EntityID**=x, **GLUE2EntityID**=y, **GLUE2EntityID**=z. We consider that this would be unduly opaque, and therefore introduce an additional rule that the *ID* attribute is defined in the Object Classes representing the classes derived immediately from *Entity*, and the naming then follows the standard rules. So for example the *ID* attribute for all of the **GLUE2Service**, **GLUE2ComputingService** and **GLUE2StorageService** objects is called **GLUE2ServiceID**.

One final point is that the schema document defines *Policy*, *Domain*, *Share*, *Manager* and *Resource* as being abstract classes which MUST NOT be instantiated, but should only be used to derive specialised entities. However, this rule is based on the fact that these objects in themselves contain no useful information, rather than that there is any structural flaw caused by instantiating them. Modifying an LDAP schema is a complex and time-consuming operation, so it may be useful at some point to prototype a new class derived from, for example, *Share* using a concrete **GLUE2Share** Object Class together with **GLUE2Extension** objects to carry the putative new attributes, and only define a new specialised Object Class once the definition of the new entity is stable. We have therefore decided to make these Object Classes concrete and instantiable. However, it should be emphasised that such objects MUST NOT be regarded as strictly compliant with the schema, that schema validation tools SHOULD reject such objects, and that tools to translate the LDAP schema to another representation MAY reject or ignore them.

To summarise, the following rules were employed:

- The **GLUE2Entity** Object Class in LDAP should carry all attributes defined in *Entity* except *ID*.
- All classes immediately deriving from *Entity* will have their own *ID* attribute named after the class name. For example, the Object Class **GLUE2Location** will have the attribute **GLUE2LocationID**.

- All classes deriving from *Entity* in GLUE will also inherit from the **GLUE2Entity** class in LDAP.
- The **GLUE2Entity** class will be of type “Abstract”.
- All classes deriving from *Entity* will be of type “Structural”.
- All other classes will be of type “Auxiliary”.

## 3.5 Data types

LDAP does not have an extensive range of data types, and there is little overlap with the types defined in the GLUE schema. For the implementation of the different data types, just two different types of the standard LDAP v3 attribute set referred to in [rfc4517] are used:

- **DirectoryString**, with OID 1.3.6.1.4.1.1466.115.121.1.15
- **Integer**, with OID 1.3.6.1.4.1.1466.115.121.1.27

“**Integer**” is used for types *UInt32* and *UInt64* of the original GLUE 2.0 Specification and “**DirectoryString**” is used for every other type.

This also means that data type integrity will largely not be checked in the LDAP implementation itself, but must be ensured by other means, for example external validation tools.

The attribute multiplicity in the schema maps naturally to LDAP since it supports both optional and multi-valued attributes directly, and hence the constraints implied by the schema (**MUST/MAY** and **SINGLE-VALUE**) are imposed directly in the LDAP attribute definitions.

Note that there are two principle changes from the LDAP representation used for GLUE 1. One is that in that case we chose **IA5String** (OID=1.3.6.1.4.1.1466.115.121.1.26) as the string type. However, this is basically 7-bit ASCII which does not allow text in various non-English languages to be represented, and moreover the presence of such strings may cause the entire object to be rejected by an LDAP server. We have therefore decided to use **DirectoryString** for GLUE 2, which is basically the UTF-8 encoding of Unicode which includes ASCII as a subset. Potentially it would be possible to use **IA5String** for the majority of attributes where the permitted values could be restricted and only use **DirectoryString** for attributes which represent free text, but in practice it seems simpler to use a uniform representation. We note that the schema document itself does not define the string type in any detail, which also implies that we should use the broadest possible type.

The second change concerns case sensitivity. The GLUE 1 schema defines strings not to be case-sensitive (a matching rule of **caseIgnoreIA5Match**), and to some extent this makes queries simpler. However, many external tools are case-sensitive, and for the GLUE 2 schema we explicitly defined strings to be case-sensitive. We have therefore followed this in the LDAP schema by defining the matching rules to be **caseExact**. This also supports the change to **DirectoryString**, since case-matching rules are more complex for extended character sets. However, this will be the most visible change in behaviour relative to GLUE 1, and hence may require some education for users.

The existence of mandatory attributes also represents a partial change from GLUE 1 which had essentially all attributes as optional. This may require more care in the writing of information providers, but also helps to ensure the quality of the published data.

## 3.6 Relationships

LDAP is not a relational database, but a directory. Thus, LDAP neither provides nor ensures relationships other than the parent-child relations implied by the hierarchical DN.

To implement relationships between objects in LDAP, for each relationship a new attribute therefore needs to be defined. In the GLUE 1 schema we defined two such attributes, **GlueChunkKey** pointing to parent objects in the DN hierarchy and **GlueForeignKey** pointing to objects outside the hierarchy. These attributes contain ID-value constraints of the form **GlueClusterUniqueID**=xyz.

In GLUE 2 we have two differences that imply a change in the way that relationships are represented. In GLUE 1 the need for the **ChunkKey** is related to the fact that some objects have only a non-unique **LocalID**, and there is therefore a need to relate those objects explicitly to their parent in order for them to be identified. For example, a **GlueSA** object can only be identified relative to its parent **GlueSE** object. By contrast, in GLUE 2 all entities (other than *Extension*) have a unique *ID* and hence can be identified uniquely, which removes the need for something similar to the **ChunkKey**.

Secondly, in GLUE 1 the unique ID attributes are only unique within objects of the same type, so for example a **GlueClusterUniqueID** and a **GlueSubClusterUniqueID** may be identical. However, in GLUE 2 we require *ID* attributes to be globally unique even across object types. It is therefore possible for the relationship value to simply be the ID.

In terms of the attribute names we felt that it would be clearer and more explicit for the name to specify the relation it represents, rather than using a generic name such as **GLUE2ForeignKey**. This also prevents the accidental publication of relationships not defined in the schema. The naming convention chosen is to have the prefix and Object Class name as for other attributes, followed by the name of the Object Class to which the reference points, and finally a suffix **ForeignKey**. (We also considered using **FK** as a more compact suffix, but decided that the longer string is likely to be easier to understand.) As an example, this means that a relation from **GLUE2Endpoint** to **GLUE2Service** is called **GLUE2EndpointServiceForeignKey**, and will have a value which is the corresponding **GLUE2ServiceID**. These attributes are inherited in the same way as any other attribute, so for example a **GLUE2StorageEndpoint** will be related to a **GLUE2StorageService** via an attribute with the same name.

Relational attributes need to be defined in the LDAP schema corresponding to every relation defined in the abstract schema, and with multiplicities as defined in the schema document. Relations are bidirectional, but there is no general need to define an attribute for both ends of the relation since LDAP queries can be performed in either direction. That is, it is possible either to query for an object which has a particular **ID** in its **ForeignKey** attribute, or for an object with an **ID** which has been extracted from a **ForeignKey**. Depending on the circumstances there may be differences in efficiency or ease of use, for example queries which return multiple **ID**s are likely to be more complex, but in general we decided to define a **ForeignKey** only for one end of a relationship.

There were two main considerations taken into account in deciding which end of the relationship to use. In many cases there is a natural parent-child relation, for example *Service* is a parent of *Endpoint*, and it is likely to be better for the relation to point from child to parent. This is both for likely ease of coding of information providers – create the parent and then loop over the children – and because the most likely query direction is to find the children of a given parent rather than vice versa.

The second consideration is multiplicity. For one-to-many relations it will normally be better to have one attribute per object than many, and even for many-to-many relations it will often be the case that one of the multiplicities is likely to be substantially more than the other. For example, the relation between *Share* and *Endpoint* is many-to-many, but in most cases there will be many more *Share*s than *Endpoint*s.

In general this mechanism is similar to the one used in relational databases with foreign keys, except for a few key points:

- In a relational database, when implementing a one-to-many relationship, the foreign key attribute is included in the “many” object since a database cell can only have one value. In LDAP attributes can be multivalued, so this may depend on the needs for each object.
- In a relational database, when implementing a many-to-many relationship, a new table is created that holds all relations due to the fact that a table cell cannot hold multivalued

attributes. LDAP supports multivalued attributes directly so there is no need for any intermediate table.

- Relational databases ensure relationship integrity, LDAP does not.

We then considered each of the schema relations individually to decide which end should carry the foreign key attribute in the light of the considerations described above, and the result is shown in Table 1. In the vast majority of cases the decision was obvious. The only exception to the "one end" rule is for the two peer relations *Service-Service* and *Activity-Activity* where the keys need to be at both ends.

| Relation 1 | Mult 1 | Mult 2 | Relation 2 | Object with key | Name |
|---|---|---|---|---|---|
| Entity | 1 | 0..* | Extension | Extension | GLUE2ExtensionEntity ForeignKey |
| Location | 0..1 | 0..* | Service | Location | GLUE2LocationService ForeignKey |
| Location | 0..1 | 0..* | Domain | Location | GLUE2LocationDomain ForeignKey |
| Contact | 0..* | 0..* | Service | Contact | GLUE2ContactService ForeignKey |
| Contact | 0..* | 0..* | Domain | Contact | GLUE2ContactDomain ForeignKey |
| AdminDomain | 1 | 0..* | Service | Service | GLUE2ServiceAdminDomain ForeignKey |
| AdminDomain | 0..1 | 0..* | AdminDomain | AdminDomain (child) | GLUE2AdminDomain AdminDomain ForeignKey |
| UserDomain | 1..* | 0..* | Policy | Policy | GLUE2PolicyUserDomain ForeignKey |
| UserDomain | 0..1 | 0..* | Activity | Activity | GLUE2ActivityUserDomain ForeignKey |
| UserDomain | 0..1 | 0..* | UserDomain | UserDomain (child) | GLUE2UserDomainUserDomain ForeignKey |
| Service | 1 | 0..* | Endpoint | Endpoint | GLUE2EndpointService ForeignKey |
| Service | 1 | 0..* | Share | Share | GLUE2ShareService ForeignKey |
| Service | 1 | 0..* | Manager | Manager | GLUE2ManagerService ForeignKey |
| Service | 0..* | 0..* | Service | Service (both) | GLUE2ServiceService ForeignKey |
| Endpoint | 0..* | 0..* | Share | Share | GLUE2ShareEndpoint ForeignKey |
| Endpoint | 1 | 0..* | AccessPolicy | AccessPolicy | GLUE2AccessPolicyEndpoint ForeignKey |
| Endpoint | 0..1 | 0..* | Activity | Activity | GLUE2ActivityEndpoint ForeignKey |
| Share | 0..* | 0..* | Resource | Share | GLUE2ShareResource ForeignKey |
| Share | 0..1 | 0..* | Activity | Activity | GLUE2ActivityShare ForeignKey |

| | | | | | |
|---|---|---|---|---|---|
| Share | 1 | 0..* | MappingPolicy | MappingPolicy | GLUE2MappingPolicyShare ForeignKey |
| Manager | 1 | 1..* | Resource | Resource | GLUE2ResourceManager ForeignKey |
| Resource | 0..1 | 0..* | Activity | Activity | GLUE2ActivityResource ForeignKey |
| Activity | 0..* | 0..* | Activity | Activity (both) | GLUE2ActivityActivity ForeignKey |
| ComputingService | 1 | 0..* | ToStorageService | ToStorageService | GLUE2ToStorageService ComputingService ForeignKey |
| ComputingManager | 1 | 0..* | Application Environment | ApplicationEnvironment | GLUE2ApplicationEnvironment ComputingManager ForeignKey |
| ComputingManager | 0..1 | 0..* | Benchmark | Benchmark | GLUE2Benchmark ComputingManager ForeignKey |
| Benchmark | * | 0..1 | Execution Environment | Benchmark | GLUE2Benchmark ExecutionEnvironment ForeignKey |
| ExecutionEnvironment | 0..* | 0..* | Application Environment | ApplicationEnvironment | GLUE2ApplicationEnvironment ExecutionEnvironment ForeignKey |
| ApplicationEnvironment | 1 | 0..* | ApplicationHandle | ApplicationHandle | GLUE2ApplicationHandle ApplicationEnvironment ForeignKey |
| ToStorageService | - | 1 | StorageService | ToStorageService | GLUE2ToStorageService StorageService ForeignKey |
| StorageService | 1 | 0..* | StorageAccess Protocol | StorageAccessProtocol | GLUE2StorageAccessProtocol StorageService ForeignKey |
| StorageService | 1 | 0..* | StorageService Capacity | StorageServiceCapacity | GLUE2StorageServiceCapacity StorageService ForeignKey |
| StorageAccessProtocol | 0..* | 0..* | ToComputingService | ToComputingService | GLUE2ToComputingService StorageAccessProtocol ForeignKey |
| StorageShare | 1 | 0..* | StorageShare Capacity | StorageShareCapacity | GLUE2StorageShareCapacity StorageShare ForeignKey |
| ToComputingService | - | 1 | ComputingService | ToComputingService | GLUE2ToComputingService ComputingService ForeignKey |
| ToComputingService | - | 1 | StorageService | ToComputingService | GLUE2ToComputingService StorageService ForeignKey |

**Table 1: Foreign Key attributes**

## 3.7 Directory Information Tree

In LDAP, object instances are arranged in a hierarchical structure called the Directory Information Tree (DIT). Each object has a unique Distinguished Name (DN) constructed from an ordered series of Relative Distinguished Names (RDNs), each of which consists of an attribute name and its value. The RDNs are required to be unique only to the extent that the full DN of every object needs to be unique, but in GLUE it is natural to use the unique *ID* of the object to form the RDN. The list of RDNs which form a DN then correspond to a tree of objects.

To some extent LDAP allows the tree structure to be used to specify queries. In particular, queries can be restricted to a subtree below a given point, and there are also so-called extensible queries which include contraints on components of the DN. However, the tree structure is specific to LDAP and will generally not be reflected in implementations using other technologies. Also it may be convenient for LDAP implementations to use a variable DIT, and for the tree to be restructured as information is aggregated for different purposes. The schema design ensures that any object can be uniquely referenced by its unique *ID* irrespective of its DN, and the implementation of references between objects described above also allows all object relations to be followed without reference to the DN. We have therefore decided that implementations should be free to use whatever DIT they find convenient, subject to the following restrictions:

- All **GLUE2Extension** objects MUST appear immediately below the object they extend, since they are logically part of the object.
- If the DIT contains a **GLUE2Service** object then all objects which represent components of the corresponding *Service* MUST be placed in the subtree below the **GLUE2Service** object, and unrelated objects MUST NOT appear there. However, isolated component objects MAY be published before aggregation into complete *Service*s.
- If objects which are related to a given *AdminDomain* have a **GLUE2AdminDomain** object as a parent in the tree it SHOULD be the *AdminDomain* to which they relate. This effectively means that sites should publish their own *Service*s where possible.
- Some LDAP servers may aggregate information from other servers, for example to combine information from many sites to form a view of an entire Grid. In such cases objects MAY be added or removed as part of the aggregation process, but the DIT relations between existing objects SHOULD be preserved.
- Implementations MAY impose additional constraints on the construction of the DIT.

The corresponding restriction on clients is that queries MUST NOT make assumptions about the DIT except in accordance with these principles. This implies that in general clients SHOULD NOT assume anything about the number or existence of RDNs in the DN of a given object, but they MAY restrict the scope of a query to a subtree at the level of **AdminDomain**s or **Service**s.

The LDAP tree requires a root DN - for GLUE 1 we chose **o=grid** for compatibility with the Globus MDS [globus-mds]. There are no strong guidelines on the choice of root [ldap-root], so we chose **o=glue** as being compact, similar to the existing usage and having reasonable justification to claim "glue" as an organization name. This enables both GLUE 1 and GLUE 2 information to be present in the same LDAP server. Having different base DNs means that it is not possible to perform single queries across both schemas, but the schema structure is sufficiently different that this is not likely to be useful, and the separation ensures that existing GLUE 1 clients will not be affected by the presence of GLUE 2 information in the same server.

Finally, we decided to define one auxiliary Object Class, called **GLUE2Group** and with a single attribute **GLUE2GroupID**. This is a "local" ID, i.e. it only needs to be sufficiently unique to ensure the uniqueness of DNs. **GLUE2Group** objects MAY be inserted at any point in the DIT. Their purpose is to group objects together in the tree, both to improve visual presentation (e.g. avoiding very long object lists in an LDAP browser) and to facilitate queries by restricting the query to the subtree below the **GLUE2Group** object. As a concrete example, *ComputingActivity* objects represent jobs in a computing system, and hence may have a very large multiplicity. It may therefore be useful to have one or more **GLUE2Group** objects as their parents in the tree to allow them to be manipulated and displayed as a unit. Implementations MAY define circumstances in which **Group**s will always be used, and MAY also define how the **GroupID**s are constructed. However it should be emphasized that **Group**s are specific to an LDAP implementation and there will in general be no corresponding entity in other representations.

As described above, implementations are broadly free to define the DIT as they choose. However, we consider it useful to define a RECOMMENDED reference implementation, as illustrated in Figure 1. This includes one use of a **Group** with **GLUE2GroupID=grid** at a level immediately below the root, which enables information relating to an entire Grid to be separated from other local information which may be published by the same server. Below this there MAY be any number of **Domain** objects, which represent the hierarchical nature of the *Domain*s in the Grid environment. For example, a computing center C, participating in a national Grid infrastructure N, which is part of a wider international infrastructure Z SHOULD construct the following DN:

**GLUE2DomainID**=C, **GLUE2DomainID**=N, **GLUE2DomainID**=Z, **GLUE2GroupID=grid, o=glue**

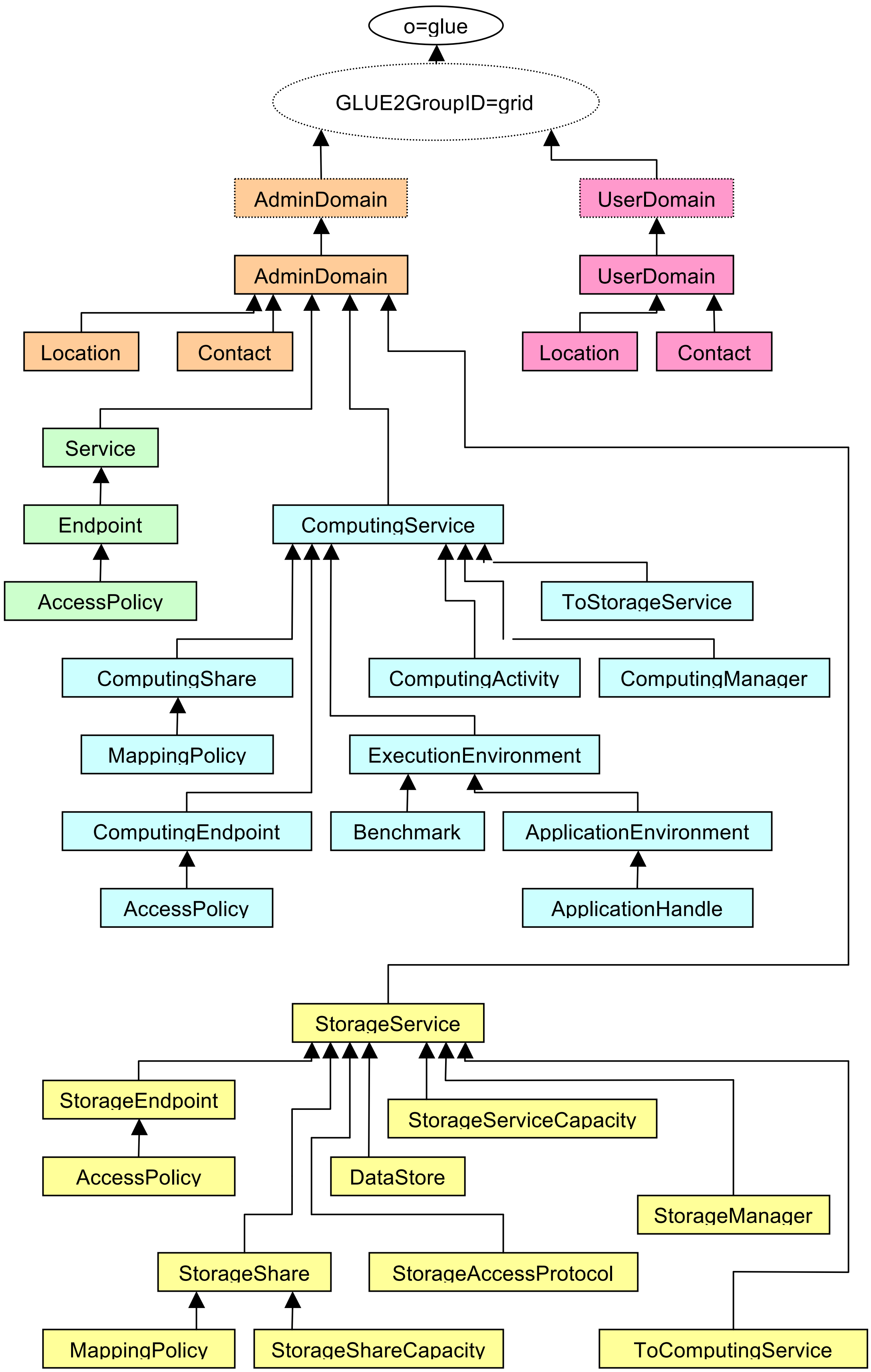


Figure 1: The recommended LDAP DIT

## 3.8 OID Assignments

The GLUE 2.0 LDAP implementation utilizes the sub tree of 1.3.6.1.4.1.6757 which is assigned to the Global Grid Forum. An overview of the main use of the sub tree is given in Tables 2, 3 and 4 representing the main entities, Computing Service entities and Storage Service entities respectively.

Since it is recommended that each attribute type should be linked to an object, we can clearly identify attributes as parts of an object OID subtree. In the case of inherited objects, we can also identify them as the parent's object OID subtree. The suggested order is that attribute types should appear first in the OID tree and object children should appear later in a concrete Object OID subtree.

Note that the OID numbers include the concrete chapter number in which the entity for that OID is referenced in the GLUE 2.0 specification. (I.e. Entity is described in chapter 5.1, thus its OID is 1.3.6.1.4.1.6757.100.1.1.5.1).

| Main Entities | |
|---|---|
| **OID** | **Entity** |
| 1.3.6.1.4.1.6757.100.1.1.5.1 | Entity <<abstract>> |
| 1.3.6.1.4.1.6757.100.1.1.5.2 | Extension |
| 1.3.6.1.4.1.6757.100.1.1.5.3 | Location |
| 1.3.6.1.4.1.6757.100.1.1.5.4 | Contact |
| 1.3.6.1.4.1.6757.100.1.1.5.5 | Domain <<abstract>> |
| 1.3.6.1.4.1.6757.100.1.1.5.5.7 | AdminDomain |
| 1.3.6.1.4.1.6757.100.1.1.5.5.8 | UserDomain |
| 1.3.6.1.4.1.6757.100.1.1.5.6 | Service |
| 1.3.6.1.4.1.6757.100.1.1.5.7 | Endpoint |
| 1.3.6.1.4.1.6757.100.1.1.5.8 | Share <<abstract>> |
| 1.3.6.1.4.1.6757.100.1.1.5.9 | Manager <<abstract>> |
| 1.3.6.1.4.1.6757.100.1.1.5.10 | Resource <<abstract>> |
| 1.3.6.1.4.1.6757.100.1.1.5.11 | Activity |
| 1.3.6.1.4.1.6757.100.1.1.5.12 | Policy <<abstract>> |
| 1.3.6.1.4.1.6757.100.1.1.5.12.5 | AccessPolicy |
| 1.3.6.1.4.1.6757.100.1.1.5.12.6 | MappingPolicy |

**Table 2: Main Entities**

| Computing Service | |
|---|---|
| **OID** | **Entity** |
| 1.3.6.1.4.1.6757.100.1.1.6.1 | ComputingService |
| 1.3.6.1.4.1.6757.100.1.1.6.2 | ComputingEndpoint |
| 1.3.6.1.4.1.6757.100.1.1.6.3 | ComputingShare |
| 1.3.6.1.4.1.6757.100.1.1.6.4 | ComputingManager |
| 1.3.6.1.4.1.6757.100.1.1.6.5 | Benchmark |
| 1.3.6.1.4.1.6757.100.1.1.6.6 | ExecutionEnvironment |
| 1.3.6.1.4.1.6757.100.1.1.6.7 | ApplicationEnvironment |
| 1.3.6.1.4.1.6757.100.1.1.6.8 | ApplicationHandle |
| 1.3.6.1.4.1.6757.100.1.1.6.9 | ComputingActivity |
| 1.3.6.1.4.1.6757.100.1.1.6.10 | ToStorageService |

**Table 3: Computing Service**

| Storage Service | |
|---|---|
| **OID** | **Entity** |
| 1.3.6.1.4.1.6757.100.1.1.7.1 | StorageService |
| 1.3.6.1.4.1.6757.100.1.1.7.2 | StorageServiceCapacity |
| 1.3.6.1.4.1.6757.100.1.1.7.3 | StorageAccessProtocol |
| 1.3.6.1.4.1.6757.100.1.1.7.4 | StorageEndpoint |
| 1.3.6.1.4.1.6757.100.1.1.7.5 | StorageShare |
| 1.3.6.1.4.1.6757.100.1.1.7.6 | StorageShareCapacity |
| 1.3.6.1.4.1.6757.100.1.1.7.7 | StorageManager |
| 1.3.6.1.4.1.6757.100.1.1.7.8 | DataStore |
| 1.3.6.1.4.1.6757.100.1.1.7.9 | ToComputingService |

**Table 4: Storage Service**

## 4. Security Considerations

Using LDAP to implement the GLUE 2.0 specification raises several considerations especially in the field of data integrity.

LDAP is not a relational database, thus it can not ensure relationship integrity. This must be ensured by other means.

LDAP can not ensure most data types referred in the GLUE 2.0 specification, thus this implementation uses the generic types “DirectoryString” and “Integer” specified in [rfc4517].

## 5. Author Information

Sergio Andreozzi, INFN

Stephen Burke, RAL

Felix Ehm, CERN

Laurence Field, CERN

Gerson Galang, ARCS

David Horat, CERN

Balazs Konya, Lund University

Maarten Litmaath, CERN

Paul Millar, DESY

JP Navarro, ANL

## 6. Contributors & Acknowledgements

We gratefully acknowledge the contributions made to this document.

## 7. Intellectual Property Statement

available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

## 8. Disclaimer



## 9. Full Copyright Notice



## 10. References

[glue-2] "GLUE 2.0 Specification". http://forge.gridforum.org/sf/go/doc15519

[glue-1] GLUE Schema Specification version 1.3. http://forge.gridforum.org/sf/go/doc14185

[rfc4512] RFC 4512. "Lightweight Directory Access Protocol (LDAP): Directory Information Models". http://tools.ietf.org/html/rfc4512

[rfc4517] RFC 4517. "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules". http://tools.ietf.org/html/rfc4517

[rfc4520] RFC 4520. "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)". http://tools.ietf.org/html/rfc4520

[ldap-root] "LDAP: Root Name Angst". http://www.zytrax.com/books/ldap/apa/ldap-root.html

[globus-mds] "MDS 2.4 in the Globus Toolkit 2.4 Release". http://www.globus.org/toolkit/docs/2.4/mds/