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
Florido Paganelli, Lund University
*co-chairs

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

June 13,2012

GLUE v. 2.0 – Reference Realization ~~Implementation of~~ to ~~an~~ LDAP Schema

Status of This Document

This document provides information to the Grid community regarding the LDAP Schema realization~~implementation~~ of the GLUE information model (v.2.0). Distribution is unlimited. This implementation is derived from the proposed recommendation GFD.147 ~~specification document~~ “GLUE Specification v. 2.0”. This document is a draft to be submitted for public comment after working group agreement ~~.~~(wg-internal version 6)

Copyright Notice

Copyright © Open Grid Forum (2010). All Rights Reserved.

Trademark

Open Grid Services Architecture and OGSA are trademarks of the Open Grid Forum.

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 realization~~implementation~~ of this information model as an LDAP Schema, and includes explanations of the major design decisions made during the rendering ~~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 adopted by technology providers, a realization in terms of a concrete data model is needed. ~~In order to be used by Grid middleware an implementation in terms of a concrete data model is required.~~

This document describes the normative realization~~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 LDAP~~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 entities~~classes~~ and attributes in the abstract model of GFD.147~~schema~~ are in *italic*, and to Object Classes and attributes in the concrete LDAP schema are in **bold**.

3. LDAP Schema Realization~~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 model~~schema~~ entity prefixed as described above, e.g. the Object Class representing the *Service* entity is called **GLUE2Service**. Each ~~object~~ attribute of an entity~~pair~~ in the abstract model~~schema~~ ~~has a new~~ is rendered as an LDAP attribute ~~type in LDAP,~~ with a name ~~which is~~ composed from the names of the entity~~object~~ ~~and the name of~~ and the attribute, for example **GLUE2ServiceType** corresponds to the *Type* attribute of the *Service* entity. 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 LDAP rendering is defined by following the most straightforward mapping in which there is a one to one correspondence between the model entities and the LDAP Object Classes. Every entity is represented by an LDAP Object Class with the set of mapped attributes affected by the explicit inheritance.

The abstract model~~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 model~~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 GLUE2 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**.

Comment [B1]: This strange choice is not motivated and explained. What is the benefit e.g. to not to have GLUE2ComputingEndpointID or GLUE2StorageSErvice ID while all the less important entities such as Benchmark have their own IDs?

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 entity~~class~~ name. For example, the Object Class **GLUE2Location** will have the attribute **GLUE2LocationID**.
- All classes deriving from *Entity* in GLUE2 will also inherit from the **GLUE2Entity** Object C~~c~~lass in LDAP.
- The **GLUE2Entity** Object C~~c~~lass will be of type “Abstract”.
- All classes deriving from *Entity* will be of type “Structural”.
- All other classes will be of type “Auxiliary”.

Comment [B2]: Seems to restrictive to allow only the first child of the Entity to be structural. Choice of structural/auxiliary should be revised . For example ComputingService is not structural.

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:

- **~~Directory~~IA5String**, with OID 1.3.6.1.4.1.1466.115.121.1.26~~15~~
- **Integer**, with OID 1.3.6.1.4.1.1466.115.121.1.27
- **Boolean,** with OID 1.3.6.1.4.1.1466.115.121.1.7

“**Integer**” is used for types *UInt32* and *UInt64* of the original GLUE 2.0 Specification and “**~~Directory~~IA5String**” and “**Boolean**” 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 model~~schema~~ maps naturally to LDAP since it supports both optional and multi-valued attributes directly, and hence the constraints implied by the model~~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.

Comment [B3]: This is not true in the latest schema that uses IA5String

The second change concerns case sensitivity. The Glue~~LUE~~ 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.

Comment [B4]: This is not true in the latest schema

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~~LUE~~ 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~~LUE~~ 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~~LUE~~ 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*.

Formatted: Font: Italic

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 model~~schema~~, and with multiplicities as defined in the model~~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 (entries) are arranged in a hierarchical structure called the Directory Information Tree (DIT). An LDAP entry consists of set of attributes taken from the object classes associated with the object. An LDAP entry MUST be composed of at least one structural objectclass and MAY use several auxiliary object classes. Each entry~~object~~ MUST have~~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 taken from a structural object class. The RDNs are required to be unique only to the extent that the full DN of every object needs to be unique.The DNs through their hierarchically ordered RDNs define the tree structure.

In the GLUE2 rendering approach the GLUE2 entries are built from objectclasses that are grouped into the same entry only if there is an inheritance relationship in the model. This results in an LDAP tree of GLUE2 entries where each non-related model entities have their own separate node. As for the DN it is natural to use the unique *ID* of the model entity to form the RDN.In case of entries with multiple objectclasses, the ID of the objectclass derived immediately from the abstract objectclass **GLUE2Entity** is used.

For example, GLUE2ComputingEndpoint entry consists of attributes of the objectclassses **GLUE2Entity**,**GLUE2Endpoint** and **GLUE2ComputingEndpoint**. Following the example the unrelated model entities *ComputingEndpoint* and *ComputingManager* rendered as **GLUE2ComputingEndpoint** and **GLUE2ComputingManager**, are separate nodes in the tree. The RDNs of **GLUE2ComputingEndpoint** and **GLUE2ComputingManager** are **GLUE2EndpointID=X** and **GLUE2ManagerID=Y.**

~~in GLUE it is natural to use the unique *ID* of the object to form th The list of RDNs which form a DN then correspond to a tree of obj~~~~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.~~ It was also decided to define an additional auxiliary Object Class not derived from the model to facilitate grouping of same type of entries under a single node in the tree. This grouping node SHOULD have a **GLUE2Group** object class with a single attribute **GLUE2GroupID**. The attribute specifies the entity class of the entries to be grouped. A **GLUE2Group** grouping entry MAY be inserted at any point in the DIT. The grouping facilitates queries by restricting the query to the subtree below the **GLUE2Group** entry and also improves visual presentation (e.g. avoiding very long object lists in an LDAP browser). 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 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 introduce a **GLUE2Group** entry with attribute **GLUE2GroupID=ComputingActivities** as their parent in the tree to allow them to be manipulated and displayed as a unit.

When deciding the DIT for the GLUE2 rendering following considerations were taken: ~~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 conceptual model~~schema design~~ ensures that any entity~~object~~ can be uniquely referenced by its unique *ID* ~~irrespective of its DN~~,

Comment [B5]: LDAP is a hierarchical data structure, no need to explain the opposite :)

Comment [B6]: This entry is used as a structural one, therefore shouldn't this be a structural object class instead of auxiliary?

Comment [B7]: This attribute is NOT an ID, furthermore it SHOULD carry information, therefore it is more like a type or name. GLUE2GroupClass or GLUE2GroupEntities

Comment [B8]: This is not true. Very similar grouping is created in the XML rendering.

and the rendering choices for the associations of the model via GLUE2ForeignKey ~~implementation of references between objects~~ described earlier~~above~~ ~~also~~ allows all entity ~~object~~ relations to be followed directly. This means DNs are not needed to identify entries or to express associations~~without reference to the DN~~.

Therefore it was agreed that as part of the GLUE2 LDAP rendering the DIT structure is not mandated. Nevertheless a set of restrictions and a proposed structure is presented. The proposed DIT structure (see figure) SHOULD be followed by those implementations that plan to benefit from the hierarchical data model of LDAP.

The minimum restrictions regarding the structure of a GLUE2 DIT to be followed are~~We have therefore decided that implementations should be free to use whatever DIT they find convenient, subject to the following restrictions~~:

- The LDAP tree requires a root DN, also called the base DN. For the DIT of a LDAP tree containing GLUE2 information the **o=glue** MUST be used. This enables both GLUE2 and Glue 1 information to be present in the same LDAP server since the Glue 1 tree has the root **o=grid.** Having different base DNs means that it is not possible to perform single queries across both trees and the separation also ensures that Glue 1 clients are not affected by the presence of GLUE 2 information in the same server.

- When aggregating DITs from different LDAP servers entries MAY be added or removed as part of the aggregation process but the DIT relations between existing entries SHOULD be preserved. Aggregation is used to combine information taken from different trees into a single DIT. ~~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.~~

- GLUE2Admindomain entries SHOULD only aggregate services as their child entries in the tree when those services are managed by that domain. ~~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 *Services* where possible.~~

- A GLUE2Service entry SHOULD aggregate all service related entries describing the specific service via placing all those entries under its subtree, unrelated entries MUST NOT appear there~~. 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 *Services*~~

- All **GLUE2Extension** entries~~objects~~ MUST appear immediately below the object they extend, since they are logically part of the object.

- ~~Implementations MAY impose additional constraints on the construction of the DIT.~~

As a consequence, ~~The corresponding restriction on~~ client~~s~~ ~~is that~~ queries SHOULD NOT~~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 position of an entry in the tree~~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 **GLUE2AdminDomain**s or **GLUE2Service**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~~

Comment [B9]: Add rerefence to figures

Comment [B10]: Empty statement

Comment [B11]: In a hierarchical data structure MUST is not the write phrase

~~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 *Domains* 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**~~

Following the presentation of the general considerations and the minimal restrictions on the DIT, below we describe the proposed GLUE2 LDAP DIT structure. The GLUE2 model can be used to describe grid information on different levels:

- Local: At the bottom there are the set of services operated on local resources within a same box or local network;
- Domain: local information from different sources then MAY be merged or aggregated at an intermediate level called the domain information;
- Global: finally information taken from the domain or the local level are aggregated on a global top level. The DIT representing the global level naturaly accommodates the domain and local level trees as subtrees.

The DITs for the three levels are shown on the figures below.

Local-level DIT:

- Right beneath the o=glue root it MUST contain a GLUE2Group entry with GLUE2GroupID=local. This entry SHOULD accommodate all the local *services*. With their complete subtrees
- Right beneath the o=glue root it MAY contain GLUE2AdminDomain or GLUE2UserDomain entries.
- The following groupings were introduced and MUST be placed into the tree as shown on the figure: GLUE2GroupID=ComputingActivities, GLUE2GroupID=ExecutionEnvironments, GLUE2GroupID=ApplicationEnvironments

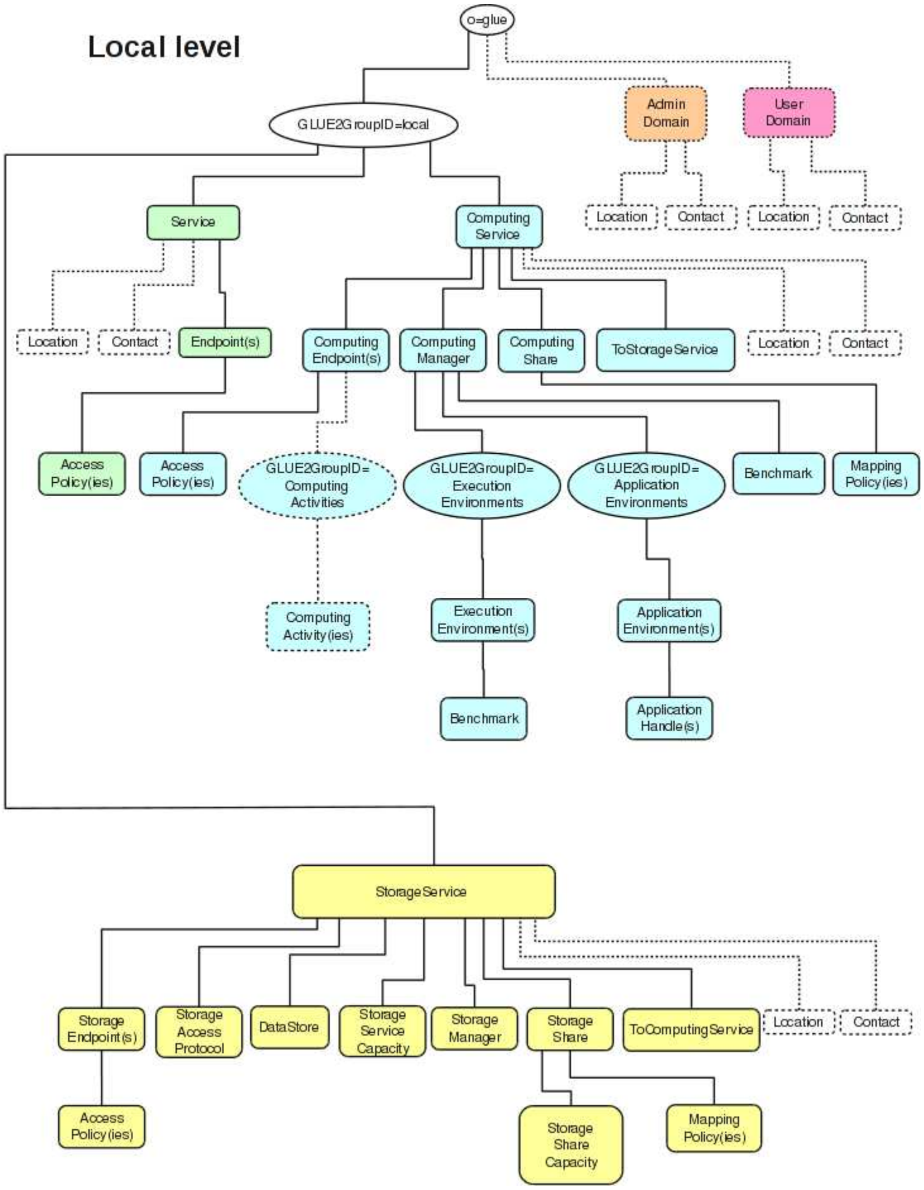

Figure 1: Proposed DIT structure of a local information source publishing *Service*, *Computing Service* and *Storage Service* together with *Domain* information**.**

Domain-level DIT:

- Right beneath the o=glue root it MUST contain only one GLUE2AdminDomain and MAY have a GLUE2UserDomain
- The AdminDomain entry MUST have a GLUE2Group entry with GLUE2GroupID=Services node that SHOULD contain all the service trees from the local level belonging to that domain.
- Right beneath the o=glue root it MAY contain a GLUE2Group entry with GLUE2GroupID=local. This entry MAY be used to describe the LDAP service itself.

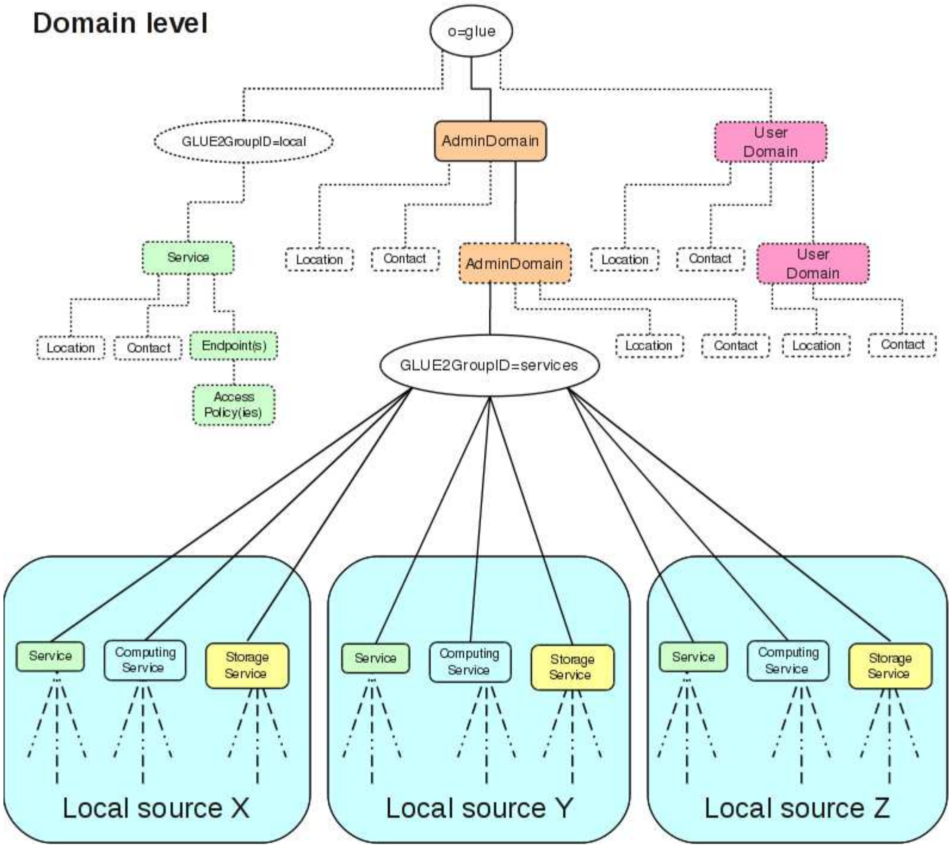

Figure 2: Proposed DIT structure for Domain-level aggregation of local information sources. Aggregation of the services from multiple local sources is done by moving every subtrees of the *GLUE2groupID=local* entries under a common *GLUE2GroupID=services* entry.

Figure 1: The recommended LDAP DIT

Global-level DIT:

- Right beneath the o=glue root it MUST contain a GLUE2Group entry with GLUE2GroupID=grid. This entry SHOULD accommodate all the local *AdminDomains and UserDomains* with their complete subtrees
- Right beneath the o=glue root it MAY contain a GLUE2Group entry with GLUE2GroupID=local. This entry MAY be used to describe the LDAP service itself.

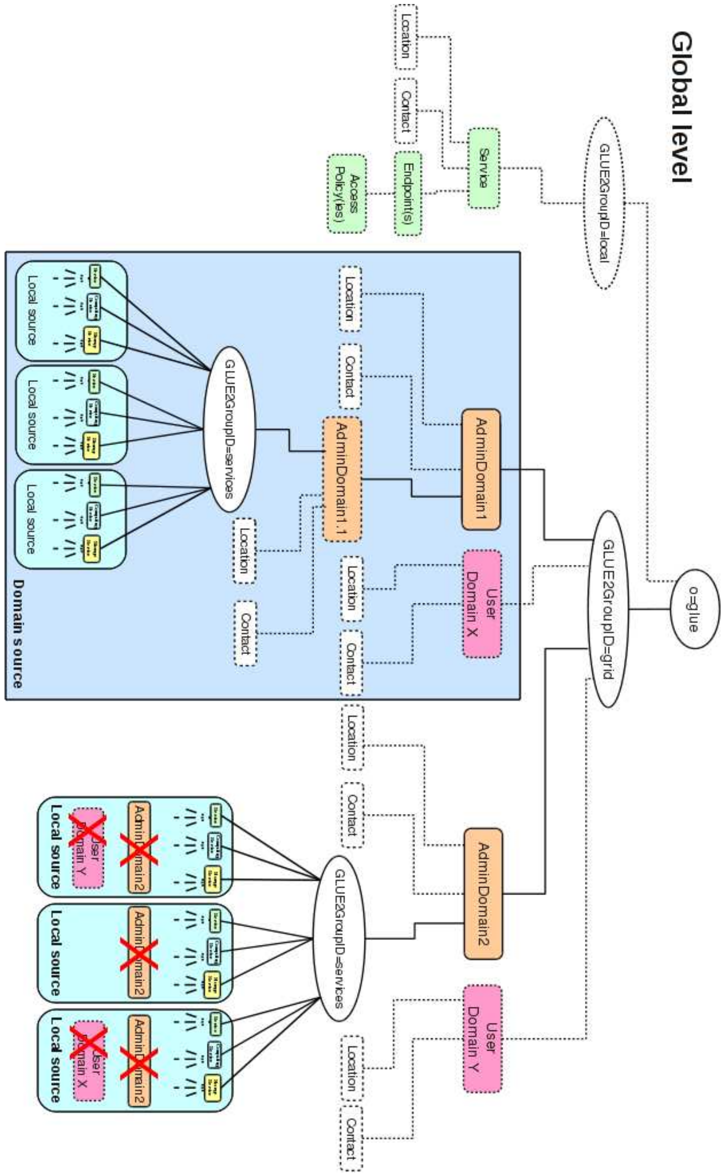

Figure 3: Proposed DIT structure for global level aggregation from both domain and local level information sources.

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.

Comment [B12]: Inheritance rules are not applied consistently, it used only for policy and domain.

Comment [B13]: The proposed order is not extensible when it comes to adding new attributes

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].

Comment [B14]: Not consistent with the latest schema

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

Florido Paganelli, Lund University

6. Contributors & Acknowledgements

We gratefully acknowledge the contributions made to this document.

7. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made 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

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

9. Full Copyright Notice

Copyright (C) Open Grid Forum (2008). 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 OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

10. References

[glue-2] “GFD.147, GLUE 2.0 Specification”. http://ogf.org/documents/GFD.147.pdf ~~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/