GWD-R-P

Authors:
Sergio Andreozzi, INFN
Stephen Burke (editor), STFC
Felix Ehm, CERN
Laurence Field, CERN
Gerson Galang, ARCS
David Horat, CERN
Balazs Konya, Lund University
Maarten Litmaath, CERN
Shiraz Memon*, JSC
Paul Millar, DESY
JP Navarro*, ANL
Florido Paganelli (editor), Lund University
*co-chairs

GLUE WG
http://redmine.ogf.org/projects/glue-wg

June 2, 2014

GLUE v. 2.0 – Reference Realisation to LDAP Schema

Status of This Document

This document provides information to the Grid community regarding the LDAP Schema realisation of the GLUE information model (v.2.0). Distribution is unlimited. This implementation is derived from the proposed recommendation GFD.147 “GLUE Specification v. 2.0”.

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 a realisation of this information model as an LDAP Schema, and includes explanations of the major design decisions made during the rendering process.

JP Navarro
Deleted: G

Contents

1. Introduction

The GLUE 2.0 information model defined in [glue-2] is an abstract information model of Grid entities. For technology providers to adopt it, it has to be realized into a concrete data model. This document describes the normative realisation of the GLUE 2.0 abstract information model in terms of an LDAP Schema. The approach followed to map the entities and relationships in the abstract information model to the LDAP 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).

This document adopts the following terminology:

"Entities" and "Classes" as defined in the abstract model of GFD.147 are referred with the terms "entity" or "entities"; "attributes" as defined in GFD.147 are called "attribute" or "attributes". Entity and attribute names as defined in GFD.147 are shown in *italic.* For example, the entity with name "Service" is represented as *Service*.

Note that the term "entity" with no typesetting is used in this document for any generic entity or class defined in GFD147, and MUST NOT be confused with the term "*Entity"* with italic typesetting, that identifies the specific entity in GFD whose name is "Entity".

All the terminology relative to LDAP is taken from definitions in RFC 4510 (see http://tools.ietf.org/html/rfc4150); object classes and attributes in the concrete LDAP schema will have typesetting **bold**. For example, **GLUE2Entity** is an LDAP object class and **GLUE2EntityID** is a LDAP attribute.

3. LDAP Schema Realisation

3.1 Approach

There are many possible approaches to realise the GLUE 2.0 abstract information 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 with 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 implementing 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 LDAP server 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 entity prefixed as described above, e.g. the object class representing the *Service* entity is called **GLUE2Service**. Each attribute of an entity in the abstract model is rendered as an LDAP attribute with a name composed from the names of the entity 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 schema objects in the rendering are 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 whose set of mapped attributes is constituted only of those attributes explicitly defined in the model as belonging to that entity, and that are NOT inherited from other entities.

The abstract model 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] into LDAP entries. 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 entity which may be defined in the future. The main disadvantage of this approach is more complexity in the naming of attributes within an LDAP entry, for example a **GLUE2ComputingEndpoint** entry can include attributes called **GLUE2ComputingEndpoint-RunningJobs**, **GLUE2EndpointURL** and **GLUE2EntityName**, but in practice this seems unlikely to cause significant problems.

A separate case could be made for the *Entity* entity, since it is unlikely that queries for the attributes of all LDAP entries will be common. In general, we conclude that consistency both with the abstract model 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 entries have an attribute that 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**.

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 entities immediately deriving from *Entity* will have their own *ID* attribute named after the entity 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 class in LDAP.
- The **GLUE2Entity** object class will be of type “Abstract”.
- All classes except *Entity* are STRUCTURAL.

3.4.1 Composition of LDAP object classes into entries

An LDAP entry is formed out of a collection of LDAP object classes. Please note that since **GLUE2Entity** is abstract, no **GLUE2EntityID** is defined for it (see discussion above), hence there is no need to include such object class. Inheritance in the schema allows to instantiate its attributes into any GLUE2 LDAP object class.

Each entry which is a realization of an entity in GFD.147 MUST include:

- An object class that has a defined **GLUE2<entityname>ID** attribute, e.g. the object class **GLUE2Service** has attribute **GLUE2ServiceID** attribute.
- Any additional object class added MUST be consistent with respect to inheritance as defined in GFD.147. For example, a **GLUE2ComputingEndpoint** entry MUST have **GLUE2Endpoint** object class as well, but SHOULD NOT have a **GLUE2StorageEndopoint** object class, as it would break the model.
- Attributes can belong to any of the included object classes, plus those of the abstract object class GLUE2Entity
- Attributes included in the RDN MUST be of the form GLUE2<entityname>ID, that is, ONLY GLUE2 ID or Local IDs as defined in GFD.147 MUST appear in the RDN, with the exception of the special **o=glue** suffix.

Example of a GLUE2ComputingService entry:

```
dn: GLUE2ServiceID=<URI>,…,o=glue
objectClass GLUE2Service
ObjectClass GLUE2ComputingService
# from GLUE2Entity
GLUE2EntityName: Human readable Service Name
GLUE2EntityCreationTime: 2013-11-26T08:28:16Z
GLUE2EntityValidity: 120
# from GLUE2Service
GLUE2ServiceID: <URI>
```

```
GLUE2ServiceType: org.ogf.glue.someservice
GLUE2QualityLevel: testing
GLUE2ServiceCapability: information.discovery.resource
# From GLUE2ComputingService
GLUE2ComputingServiceTotalJobs: 1
GLUE2ComputingServiceRunningJobs: 1
…
```

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.

DirectoryString type does NOT allow empty strings. This means that if an attribute has no value the corresponding LDAP attribute entry will NOT be published by any LDAP server. Information consumers and validation tools MUST take this into account.

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

Due to the fact that this document has been in DRAFT status for long time, some of the existing implementations of the LDAP schema prior to the current revision of this document do not yet follow the above recommendations. In particular:

- The Boolean object type (OID 1.3.6.1.4.1.1466.115.121.1.7) was used for all attributes with type *ExtendedBoolean_t* in GFD147
- The string type IA5String (OID 1.3.6.1.4.1.1466.115.121.1.26) was used for all objects of different type from GFD.147 *UInt32, UInt64, and ExtendedBoolean_t*

The GLUE WG recommends updating all renderings and adopting version 2.0 of the schema as soon as this document is out. The attribute multiplicity in the model maps naturally to LDAP since it supports both optional and multi-valued attributes directly, and hence the constraints implied by the model (**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. First, in the case of choosing **IA5String** (OID=1.3.6.1.4.1.1466.115.121.1.26) as the string type. However, this is basically 7-bit ASCII that 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 that 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 that 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 implementation 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 must 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 GLUE2 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 GLUE2 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 GLUE2 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 that 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, and with multiplicities as defined in the model 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, but 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 Structuring the LDAP entries

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 object class and MAY use several auxiliary object classes. Each entry MUST have 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 object classes 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 object classes, the ID of the object class derived immediately from the abstract object class **GLUE2Entity** is used.

For example, GLUE2ComputingEndpoint entry consists of attributes of the object classes **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.**

It was also decided to define an additional structural 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. Please note that this attribute is NOT a GLUE2 ID as defined in GFD.147 (i.e. does not have to be universally unique and not a URI) but is rather a Local ID, meant to identify the grouping, and follows the restrictions of LocalID_t as in GFD.147. 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.

For 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 the following considerations were taken: it may be convenient for different LDAP implementations to use a different DIT, and for the tree to be restructured as information is aggregated for different purposes. The GLUE 2 abstract information model ensures that any entity can be uniquely referenced by its unique *ID*, and the rendering choices for the associations of the model via GLUE2ForeignKey described earlier allows all entity relations to be followed directly. This means DNs are not needed to identify entries or to express associations.

Therefore it was agreed that as part of the GLUE2 LDAP rendering the DIT structure is not mandated. Implementers are free to build their own DIT. Some of the existing production implementations are presented in Appendix A.

Nevertheless a single restriction is presented: 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.

An additional restriction is MANDATORY:

- All **GLUE2Extension** entries MUST appear immediately below the object they extend, since they are logically part of the object, it would actually not make sense to do otherwise.

3.8 Aggregation

In this context, aggregation means composition of different LDAP trees under the same tree. At the time of writing, there is no defined way of aggregating different GLUE2 documents into one. In this section we will attempt to define a minimal structure that has proven to be effective in some implementation,

The GLUE2Group object class can also be used to do aggregation, instead of being used to group object the same kind. This object can be seen as an “insertion point”, that is, a node of the LDAP DIT under which subtrees of other LDAP trees can be merged.

In the purpose of reaching a minimum common aggregation strategy, it is decided that the special **GLUE2Group** with **GLUE2GroupID**=grid is reserved for merging *AdminDomain*s hosting grid *Service*s from different sources. **GLUE2GroupID**=grid is an insertion point for *Domain* aggregation.

Additionally, the other two special GLUE2GroupID strings “resource”, “services” are reserved for aggregation of services coming from different sources. A ”resource” can be used for backward compatibility with Glue1 service aggregation, while “services” can be used by those who are not bound to any Glue1 implementation. **GLUE2GroupID**=resource/services is an insertion point for *Service* aggregation.

Hence, an implementer willing to realise *Domain* aggregation MUST follow the restrictions written below:

- The aggregation GLUE2Group entry MUST appear right after the o=glue. That is, the DN of such entry MUST be exactly: GLUE2GroupID=grid,o=glue
- The aggregated trees should appear right after such entry. That is, each LDAP entry belonging to the aggregated tree with its own RDN MUST be concatenated with the suffix GLUE2GroupID=grid,o=glue, i.e. <RDN>,GLUE2GroupID=grid,o=glue

Examples of how production implementers realised aggregation are shown in Appendix A.

3.9 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. The sub tree OIDs mapped to each LDAP object class are shown in Table 2. Table 3 gives an example of how OIDs are assigned to attributes for *AdminDomain* and its ancestor LDAP object classes.

OID are important for the LDAP server, to uniquely identify and validate object classes and attributes contained in the LDIF entries that will populate the LDAP database. These are placed in the LDAP schema as part of the definition of each LDAP object class and attribute.

Information consumers and LDAP clients will search data using more human readable query parameters, and can therefore completely avoid knowing these OIDs.

It is however important on the server side to have a simple and consistent way of assigning these OIDs so that future extensions of the GLUE2 schema are easy to rollout.

For this reason the numbering schema in previous version of this document was revised. LDAP Object classes and their attributes OID follow these simple rules:

- For each entity *E* in GFD.147, an OID is assigned, in the form:
  1.3.6.1.4.1.6757.100.1.1.X
  Where X uniquely identifies the entity *E,* X is a positive integer, X>0.
- For each attribute *A* that belongs to the entity *E* with OID 1.3.6.1.4.1.6757.100.1.1.X, an OID is assigned of the form:
  1.3.6.1.4.1.6757.100.1.1.X.Y
  Where Y uniquely identifies the attribute *A*. Y is a positive integer, Y>0.

3.9.1 Adding new object classes or attributes

Within this framework, an implementer willing to add a new attribute to an existing LDAP object class 1.3.6.1.4.1.6757.100.1.1.X has to look for the attribute with OID 1.3.6.1.4.1.6757.100.1.1.X.Y with the highest Y, and define a new attribute with OID 1.3.6.1.4.1.6757.100.1.1.X.Z where Z=Y+1. Always add attributes increasing the Y number.

> Example: Since **GLUE2AdminDomain** OID is 1.3.6.1.4.1.6757.100.1.1.6 (see table2), the new attribute will have OID 1.3.6.1.4.1.6757.100.1.1.6.Y. To choose Y, let's look at the defined attributes for **GLUE2AdminDomain**: the highest Y is 3 (see table 3), hence the new attribute will have OID 1.3.6.1.4.1.6757.100.1.1.6.4.

Similarly, an implementer willing to add a new LDAP object class and its attributes will have to define a new OID 1.3.6.1.4.1.6757.100.1.1.W where W=X+1, X being the highest used value within all the existing object classes OIDs 1.3.6.1.4.1.6757.100.1.1.X. Then he will use the OIDs 1.3.6.1.4.1.6757.100.1.1.W.Y, Y>0, for each of the new attributes of the object class.

> Example: Check table 2. Let OIDs in that table be 1.3.6.1.4.1.6757.100.1.1.X. The highest X used for object class OIDs is 36. A new object must have W=X+1, hence one can define the new LDAP object class to have OID 1.3.6.1.4.1.6757.100.1.1.37 and all its attributes to be OID 1.3.6.1.4.1.6757.100.1.1.37.1 OID=1.3.6.1.4.1.6757.100.1.1.37.2 , … , and so on.

Main Entities		
OID	**LDAP object class NAME**	**GFD.147 entity**
1.3.6.1.4.1.6757.100.1.1.1	**GLUE2Entity <<abstract>>**	*Entity <<abstract>>*
1.3.6.1.4.1.6757.100.1.1.2	**GLUE2Extension**	*Extension*
1.3.6.1.4.1.6757.100.1.1.3	**GLUE2Location**	*Location*
1.3.6.1.4.1.6757.100.1.1.4	**GLUE2Contact**	*Contact*
1.3.6.1.4.1.6757.100.1.1.5	**GLUE2Domain**	*Domain*
1.3.6.1.4.1.6757.100.1.1.6	**GLUE2AdminDomain**	*AdminDomain*

1.3.6.1.4.1.6757.100.1.1.7	**GLUE2UserDomain**	*UserDomain*
1.3.6.1.4.1.6757.100.1.1.8	**GLUE2Service**	*Service*
1.3.6.1.4.1.6757.100.1.1.9	**GLUE2Endpoint**	*Endpoint*
1.3.6.1.4.1.6757.100.1.1.10	**GLUE2Share**	*Share*
1.3.6.1.4.1.6757.100.1.1.11	**GLUE2Manager**	*Manager*
1.3.6.1.4.1.6757.100.1.1.12	**GLUE2Resource**	*Resource*
1.3.6.1.4.1.6757.100.1.1.13	**GLUE2Activity**	*Activity*
1.3.6.1.4.1.6757.100.1.1.14	**GLUE2Policy**	*Policy*
1.3.6.1.4.1.6757.100.1.1.15	**GLUE2AccessPolicy**	*AccessPolicy*
1.3.6.1.4.1.6757.100.1.1.16	**GLUE2MappingPolicy**	*MappingPolicy*
Computing Service Entities		
OID	**LDAP object class NAME**	**GFD.147 entity**
1.3.6.1.4.1.6757.100.1.1.17	**GLUE2ComputingService**	*ComputingService*
1.3.6.1.4.1.6757.100.1.1.18	**GLUE2ComputingEndpoint**	*ComputingEndpoint*
1.3.6.1.4.1.6757.100.1.1.19	**GLUE2ComputingShare**	*ComputingShare*
1.3.6.1.4.1.6757.100.1.1.20	**GLUE2ComputingManager**	*ComputingManager*
1.3.6.1.4.1.6757.100.1.1.21	**GLUE2Benchmark**	*Benchmark*
1.3.6.1.4.1.6757.100.1.1.22	**GLUE2ExecutionEnvironment**	*ExecutionEnvironment*
1.3.6.1.4.1.6757.100.1.1.23	**GLUE2ApplicationEnvironment**	*ApplicationEnvironment*
1.3.6.1.4.1.6757.100.1.1.24	**GLUE2ApplicationHandle**	*ApplicationHandle*
1.3.6.1.4.1.6757.100.1.1.25	**GLUE2ComputingActivity**	*ComputingActivity*
1.3.6.1.4.1.6757.100.1.1.26	**GLUE2ToStorageService**	*ToStorageService*
Storage Service Entities		
OID	**LDAP object class NAME**	**GFD.147 entity**
1.3.6.1.4.1.6757.100.1.1.27	**GLUE2StorageService**	*StorageService*
1.3.6.1.4.1.6757.100.1.1.28	**GLUE2StorageServiceCapacity**	*StorageServiceCapacity*
1.3.6.1.4.1.6757.100.1.1.29	**GLUE2StorageAccessProtocol**	*StorageAccessProtocol*
1.3.6.1.4.1.6757.100.1.1.30	**GLUE2StorageEndpoint**	*StorageEndpoint*
1.3.6.1.4.1.6757.100.1.1.31	**GLUE2StorageShare**	*StorageShare*
1.3.6.1.4.1.6757.100.1.1.32	**GLUE2StorageShareCapacity**	*StorageShareCapacity*
1.3.6.1.4.1.6757.100.1.1.33	**GLUE2StorageManager**	*StorageManager*
1.3.6.1.4.1.6757.100.1.1.34	**GLUE2DataStore**	*DataStore*
1.3.6.1.4.1.6757.100.1.1.35	**GLUE2ToComputingService**	*ToComputingService*
1.3.6.1.4.1.6757.100.1.1.36	**GLUE2Group**	*Group*

Table 2: OID assigned to LDAP object classes and their corresponding entity in GFD.147.

Entity attributes		
OID	**LDAP Attribute Name**	**GFD.147 Attribute name**
--	**GLUE2EntityID**	*ID*
1.3.6.1.4.1.6757.100.1.1.1.2	**GLUE2EntityName**	*Name*
1.3.6.1.4.1.6757.100.1.1.1.3	**GLUE2EntityOtherInfo**	*OtherInfo*
1.3.6.1.4.1.6757.100.1.1.1.4	**GLUE2EntityCreationTime**	*CreationTime*
1.3.6.1.4.1.6757.100.1.1.1.5	**GLUE2EntityValidity**	*Validity*
Domain attributes		
OID	**LDAP Attribute Name**	**GFD.147 Attribute name**
1.3.6.1.4.1.6757.100.1.1.5.1	**GLUE2DomainID**	*ID*
1.3.6.1.4.1.6757.100.1.1.5.2	**GLUE2DomainDescription**	*Description*
1.3.6.1.4.1.6757.100.1.1.5.3	**GLUE2DomainWWW**	*WWW*
AdminDomain attributes		
OID	**LDAP Attribute Name**	**GFD.147**

		Attribute name
1.3.6.1.4.1.6757.100.1.1.6.1	**GLUE2AdminDomainDistributed**	*Distributed*
1.3.6.1.4.1.6757.100.1.1.6.2	**GLUE2AdminDomainOwner**	*Owner*
1.3.6.1.4.1.6757.100.1.1.6.3	**GLUE2AdminDomainAdminDomainForeignKey**	*AdminDomain* to *AdminDomain* association

Table 3: OID assigned to LDAP attributes of the AdminDomain entity and its ancestors. Note that GLUE2EnityID does not exist, as it is an abstract entity, so there is no corresponding LDAP attribute and OID.

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 cannot ensure relationship integrity. This must be ensured by other means.

LDAP cannot 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

Florido Paganelli, Lund University

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

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

10. References

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

[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

11. Appendix A: Example LDAP tree structures in existing server side implementations

This section gives a detailed explanation of how the DIT has been realised by the gLite and ARC grid middlewares.
Historically, these two decided to follow different approaches, but through agreements on how aggregation is done, it was possible to connect the different trees.

Aggregation is used to combine information taken from different trees into a single DIT. 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 in that tree SHOULD be preserved.

Following the presentation of the general considerations and the minimal restrictions on the DIT as stated in Section 3.7, gLite and ARC defined the following GLUE2 LDAP DIT structure. The GLUE2 model can be used to describe Grid information on different levels, from lower to top:

1) gLite "Resource" or ARC "local" level
The bottom level, where no aggregation is usually done, contains a set of services operated on local resources within a same box or local network;

In the **gLite** model, all *Service*s are aggregated under the **GLUE2GroupID=resource,o=glue** LDAP entry. The software component that generates such data is called Resource BDII. Resource BDIIs are usually not exposed to information consumers directly. For this reason gLite Resource BDII does NOT contain **GLUE2AdminDomain** entries. These entries are added later by another component, Site BDII, described later. Resource BDIIs do not publish ComputingActivities for performance reasons.

In the **ARC** model, all *Service*s are aggregated under the **GLUE2GroupID=services,o=glue** LDAP entry. The software component that generates such data is called ARIS. ARIS is usually exposed to information consumers directly. This is a key difference with gLite BDII, as this service level tree MAY contain an LDAP entry with DN **GLUE2DomainID=<domainID>,o=glue** which contains information about the *AdminDomain* to which the services belong, where ID is a URI as mandated by GFD.147. ARIS tree includes the following groupings: **GLUE2GroupID=ComputingActivities**, **GLUE2GroupID=ExecutionEnvironments**, **GLUE2GroupID=ApplicationEnvironments**. These are placed in the trees according to Figure 1.

2) gLite "Site" or Domain level
In the **gLite** model, local information from different sources like Resource BDIIs or ARIS local trees then MAY be merged or aggregated at an intermediate level by a software component called Site BDII. The top of its tree is labeled with DN

GLUE2DomainID=<domainName>,o=glue
where domainName is a free form string.

Right under this DN, Resource BDII's contents under the entity
GLUE2GroupID=resource,o=glue
is aggregated with the DN suffix
GLUE2GroupID=resource,GLUE2DomainID=<domainName>,o=glue.

As a consequence of agreements on the structure of the DIT finalized in 2012, ARIS's local content under
GLUE2GroupID=services,o=glue
is also aggregated as
GLUE2GroupID=services,GLUE2DomainID=<domainName>,o=glue.

All these DNs suffixes listed above are usually referred to as “insertion points”, to indicate that a tree coming from a different source with DN **GLUE2GroupID=resource,o=glue** or **GLUE2GroupID=services,o=glue** will be inserted starting from the entity with the same **GLUE2GroupID** attribute value.

The content aggregated by a Site BDII is usually available for information consumers.
ARC has no equivalent of a Site BDII and therefore no Domain level. The same information is distributed in ARIS, at the local level.

3) gLite “Top” or Global level

In the **gLite** model, information taken from the Site or the Resource/local levels is aggregated on a global top level by a component called Top BDII. The DIT representing the global level naturally accommodates the domain and local level trees as subtrees.

In this context, the insertion point **GLUE2GroupID=grid,o=glue** aggregates **GLUE2DomainID=<domainName>,o=glue** trees coming from different Site BDIIs, as

GLUE2DomainID=<domainName>,GLUE2GroupID=grid,o=glue.

*Service*s contained in such trees will be aggregated with suffix

GLUE2GroupID=resource,GLUE2DomainID=<domainName>,GLUE2GroupID=grid,o=glue

ARC does not have an equivalent to a Top BDII. All the information needed for the aggregation should be fetched by Top BDII directly from ARIS. While an agreement on how to achieve this has been already established, work with this respect is still ongoing.

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

Notable properties of the common DIT structure are as follows:

- **GLUE2AdminDomain** entries only aggregate services as their child entries in the tree when those services are managed by that Domain.
- 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

Local-level DIT (gLite Resource BDII or ARC ARIS):

- gLite and ARC: Right beneath the o=glue root it contains a **GLUE2Group** entry with **GLUE2GroupID=resource** or **GLUE2GroupID=services** . This entry usually accommodates all the local *Services* with their complete subtrees.
- In ARC, right beneath the o=glue root it MAY contain GLUE2AdminDomain or GLUE2UserDomain entries.

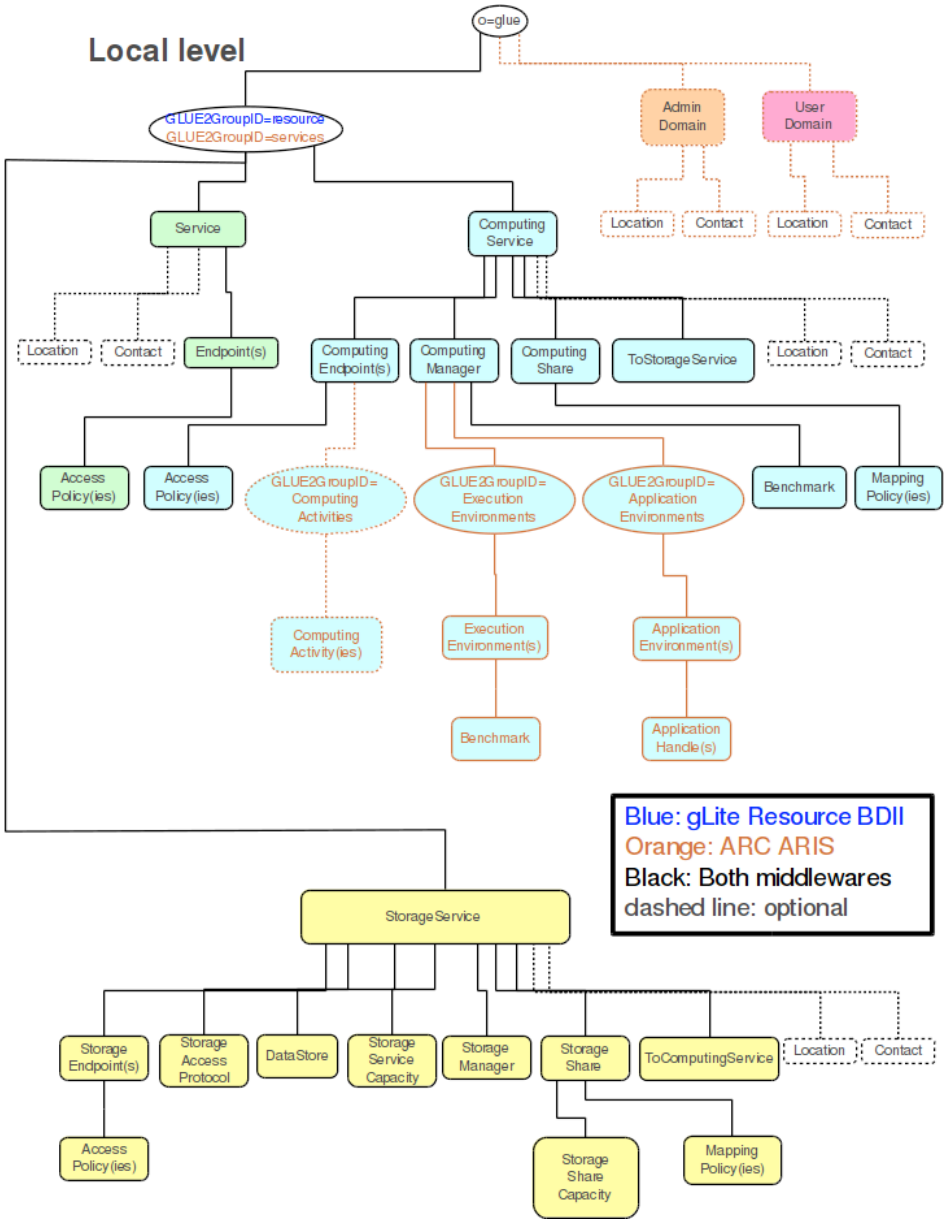

Figure 1: Example DIT structure of a local information source (gLite Resource BDII or ARIS ARC) publishing *Service*, *Computing Service* and *Storage Service* together with *Domain* information.

Domain-level DIT (gLite Site-BDII):

- Right beneath the o=glue root it contains only one **GLUE2AdminDomain** entry and might have a **GLUE2UserDomain**
- The **GLUE2AdminDomain** entry can have a **GLUE2Group** entry with **GLUE2GroupID=resource** or **GLUE2GroupID=services** attribute that contains all the service trees from the local level belonging to that domain.
- Right beneath the o=glue root it contains a **GLUE2Group** entry with **GLUE2GroupID=resource**. This entry is used to describe the LDAP service itself.

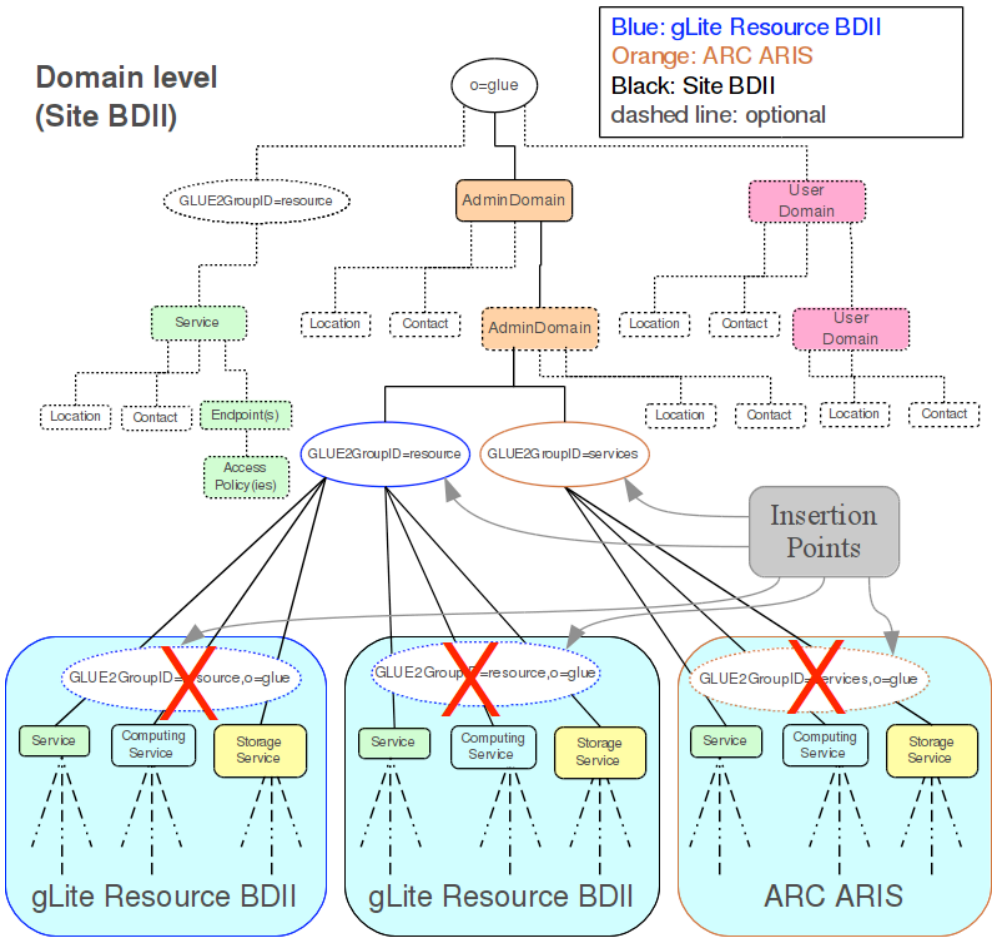

Figure 2: Example DIT structure for Site BDII aggregation of local information sources. Aggregation of the services from multiple local sources is done by moving every subtrees of the **GLUE2GroupID=resource** or **GLUE2groupID=services** entries under the **GLUE2AdminDomain** entry.

Global-level DIT (gLite Top BDII):

- Right beneath the o=glue root it contains a **GLUE2Group** entry with **GLUE2GroupID=grid**. This entry accommodates all the local *AdminDomains and UserDomains* with their complete subtrees
- Right beneath the o=glue root it contains a **GLUE2Group** entry with **GLUE2GroupID=resource**. This entry is used to describe the LDAP service itself.

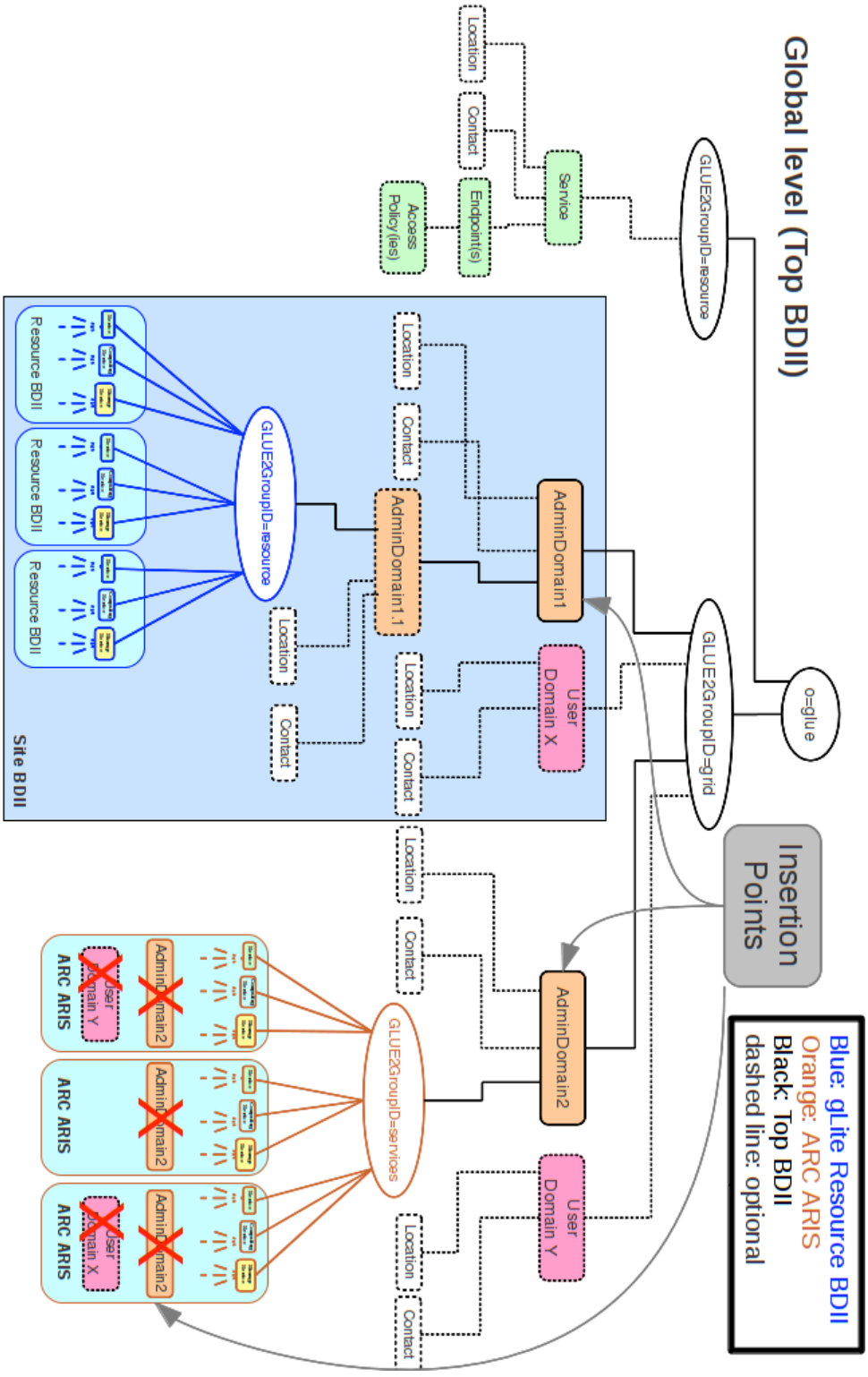

Figure 3: Example DIT structure for Top BDII aggregation from both Site BDII and local level information sources.