GOCDB5 Grid Topology Information System

David.meredith@stfc.ac.uk (corresponding author)
John.Casson@stfc.ac.uk
George.Ryall@stfc.ac.uk
James.McCarthy@stfc.ac.uk

Contents

Executive Summary

GOCDB5 is an information system for recording Grid topology data such as service endpoints, sites, downtimes and users. GOCDB is the central topology database for the EGI project. Funded development will continue until March 2015 and there is interest in continuing the activity in Horizon 2020 and the UK GridPP5 project. The GOCDB5 codebase uses standards and best practices (MVC, services architecture, unit testing, ORM) to provide a stable, easily customisable product.

GOCDB5 supports multiple projects and is used to manage the relationships between different Grid entities using a well constrained relational schema. It includes a comprehensive role-based permissions model and each instance supports project specific business rules. Users request roles over entities such as sites or projects. Role requests can be subsequently granted or revoked by those holding a role over the entity. The core domain model closely resembles a sub-set of the GLUE 2[1] information model and includes Projects, Admin-Domains, Sites, ServiceGroups/VirtualSites, Services, Downtimes, Users and Roles (there are a number of key differences as explained later on).

The tool provides a web portal for editing information and a REST style programmatic interface (PI) for querying data in XML. A flexible tag-cloud mechanism allows Grid entities to be tagged with one or more scope-tags. This allows resources to be tagged and grouped into multiple categories without duplication of information – this is essential to maintain the integrity of topology information across different infrastructures and projects. Different scope tags can be defined according to requirement, for example, tags can be defined to reflect different projects, infrastructure groupings and sub-projects or ventures. Resources can be flexibly 'filtered-by-tag' when querying for data via the programmatic interface (PI).

GOCDB5 supports multiple databases out-of-the-box through the use of the Doctrine Object Relational Mapping library (ORM). A comprehensive DBUnit test suite ensures out of the box compatibility with Oracle and MySQL. Other databases such as Postgres should also be supported via Doctrine with a change to the DB connection settings. GOCDB also provides an administration interface for common admin tasks.

An authentication abstraction layer has been integrated to allow different authentication mechanisms to be supported using a pluggable 'AuthenticationProvider' interface. Implementations are provided for x509 and username/password.

GOCDB5 was released into production in EGI on 2nd October. Post v5 release work is focussing on a flexible extensibility mechanism for defining custom properties using key-value pairs and a GLUE 2 XML[2] rendering of the GOCDB5 data. New attributes will be added to the GOCDB5 entities to give greater coverage of GLUE 2 attributes. Work is also necessary to 'productise' and package the GOCDB5 application.

[1] http://www.ogf.org/documents/GFD.147.pdf
[2] http://redmine.ogf.org/projects/editor-pubcom/boards/9

Domain Model Overview

The GOCDB5 domain closely resembles a sub-set of the GLUE 2 Grid entity model. GOCDB5's core entities are described below and their main relationships are summarised in Figure 1. The full entity relationship diagram (ERD) is provided in Appendix1. Since the structure of the GOCDB5 data model closely resembles GLUE 2, especially in terms of the entities and their relationships, we expect the structure of the data model to remain largely static. However, it is expected that new attributes will be added to the existing entities to populate more of the GLUE 2 attributes.

- **OwnedEntity**: An abstract super class that allows user Role objects to be linked to objects that extend OwnedEntity. This currently includes Project, NGI, Site and ServiceGroup.

- **Project** (extends OwnedEntity): A Project is a generic entity and can be defined multiple times in GOCDB5. A Project can aggregate zero or more 'NGI' objects and is used to cascade Project level roles over its child NGIs.

- **NGI** (extends OwnedEntity): An 'NGI' is an EGI specific term but it simply reflects an administrative domain, corresponding to a GLUE 2 AdminDomain. An NGI aggregates zero or more Sites and can belong to one or more Projects. Users with roles over an NGI can have different permissions cascading over its child Sites. The GOCDB team have experience removing references to "NGI"s and other EGI specifics while collaborating with EUDAT on their GOCDB instance.

- **Site** (extends OwnedEntity): A Site represents a physical site with a location. A Site hosts zero or more Services. A Site also corresponds to a GLUE 2 AdminDomain. Users with roles over the Site have various permissions over a Site's services.

- **ServiceGroup** (extends OwnedEntity): A ServiceGroup is also known as a 'Virtual-Site'. It is used to group existing services that are physically distributed across multiple hosting sites into a virtual service grouping. Users with roles over a ServiceGroup do not (currently) have permissions over the aggregated services. Rather, a user must apply for a role over the service's hosting Site.

- **Service:** Represents an instance of a specific service and has a defined service-type enum value (e.g. 'org.service.type.X' or 'org.service.type.Y'). GOCDB5 does not currently distinguish between ComputingService and StorageService like GLUE 2. A Service defines an Endpoint which can be linked to zero or more Downtimes.

- **Downtime:** A Downtime object can be linked to one or more service endpoints and is used to declare a downtime for all of the joined services. A key difference between GOCDB5 and GLUE 2 is that in GOCDB5, a service endpoint can be linked to zero or many downtimes. This is required so that a history of past and pending downtimes can be recorded. In contrast, a GLUE 2 service can only publish a single set of downtime information for a particular service instance (usually the current or future downtime).

- **User**: Represents a user account. A User object owns one or more Role objects.

- **Role**: A Role joins and User and an OwnedEntity. It is used to define user permissions over the joined OwnedEntity. A user can own many Roles.

- **Scope**: A Scope entity defines a tag/label that can be associated to any entity that defines the 'IScopedEntity' interface. Entities implementing this interface include Site, NGI, Service (and by extension Downtime), ServiceGroup and Project.

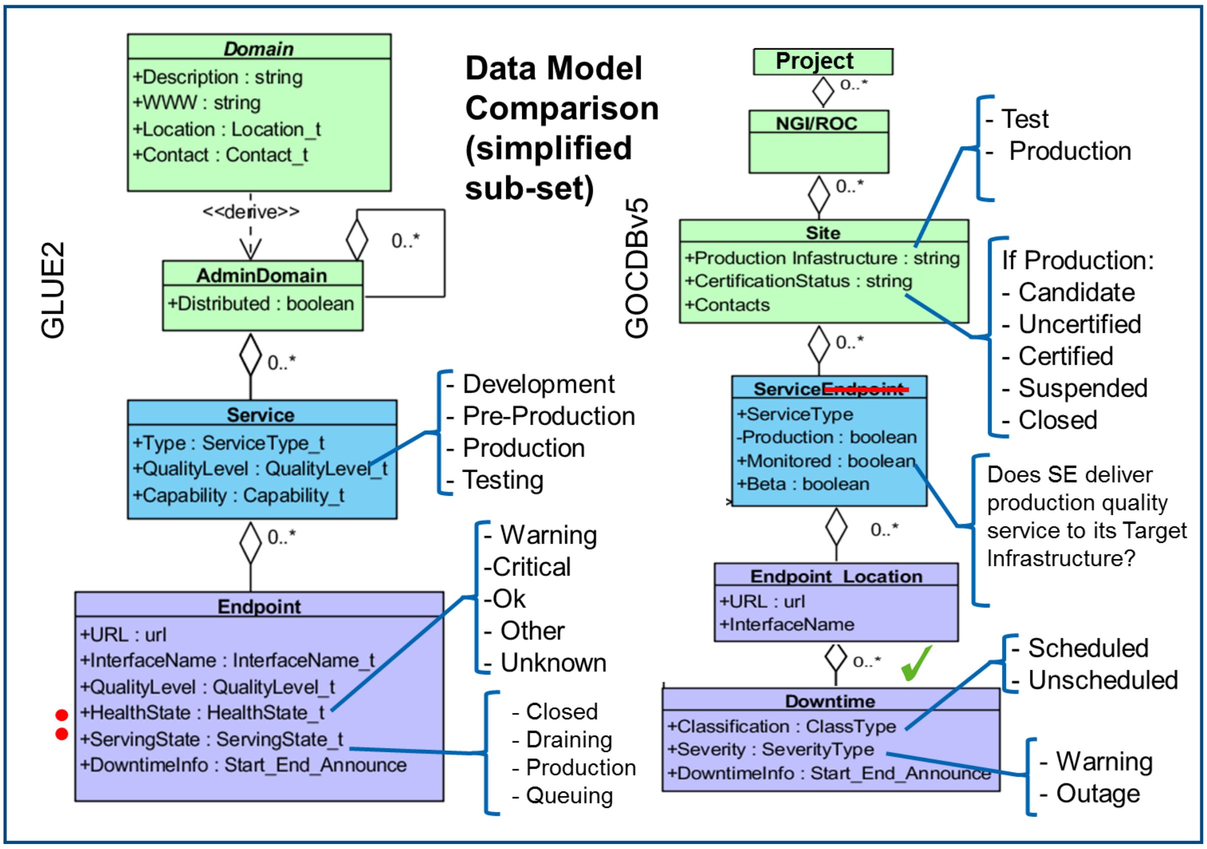


Figure 1. The GOCDB5 domain model (showing simplified sub-set) closely resembles a sub-set of the GLUE 2 Grid model, especially in terms of the entities and their corresponding relationships. To support GLUE 2, new attributes will need to be added to the existing GOCDB5 entities as/when required.

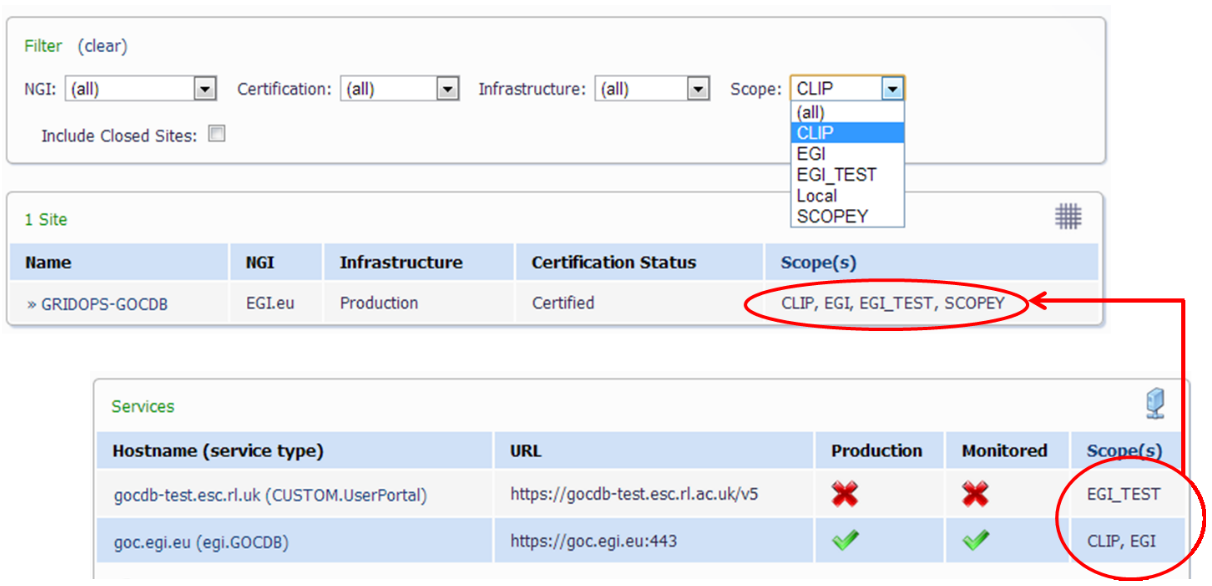
Multiple Projects and Scope Tags

GOCDB5 can host multiple projects. Projects are used to cascade project-level user roles and permissions over its child objects such as NGIs.

Conversely, scope tags help organise resources into different categories like a tag-cloud. New scope tags are added by admins on request. This allows users to tag their own resources with one or more scope-tags as necessary. The GOCDB admins control which scope tags are made available to avoid proliferation of tags (user defined tags are reserved for the extensibility mechanism). As shown in Figure 2, a site's scope list could aggregate all of the scopes defined by its child services. In doing this, the site scope list becomes a union of its service scopes plus any other site specific tags defined by the site. By defining scope tags, resources can be 'filtered-by-scope-tag' when querying for data in the PI using the 'scope' and 'scope_match' parameters (see the section on PI for details).

Clear Separation of Concerns

- It is important to understand that scopes and projects are distinct:
    - Projects are used to cascade roles and permissions over child objects
    - Scope-tags are used to filter resources into flexible categories
- In doing this, it is easy to define scope tags to mirror the projects hosted in GOCDB5. For example, assuming two projects (e.g. EGI.eu and EU-DAT), two corresponding tags may be defined ('EGI' 'EUDAT').
- In addition, it is also possible define additional scopes for finer grained resource filtering e.g. 'CLIP' and 'EGI_TEST'.
- The key benefit: A clear separation of concerns between cascading permissions and resource filtering.



NGI: (all) Certification: (all) Infrastructure: (all) Scope: CLIP

1 Site

Name	NGI	Infrastructure	Certification Status	Scope(s)
» GRIDOPS-GOCDB	EGI.eu	Production	Certified	CLIP, EGI, EGI_TEST, SCOPEY

Services

Hostname (service type)	URL	Production	Monitored	Scope(s)
gocdb-test.esc.rl.uk (CUSTOM.UserPortal)	https://gocdb-test.esc.rl.ac.uk/v5	✗	✗	EGI_TEST
goc.egi.eu (egi.GOCDB)	https://goc.egi.eu:443	✓	✓	CLIP, EGI

Figure 2. Sites and Services can be associated with one or more scope tags (other entities can also be tagged as required). In this example, the hosting site aggregates all of the scope tags defined by its child services. In doing this, the site scope list effectively becomes a union of the service scope tags + additional site specific scopes.

The Role Model and Permissions

- A user can request different Roles over different OwnedEntities.
- A Role object has a status of GRANTED or PENDING and links a User object to an OwnedEntity (see Figure 3).
- OwnedEntity is an abstract super class. Implementations currently include Project, Site, NGI and ServiceGroup.
- Role requests can be granted or revoked by users who already own the necessary roles over the specified OwnedEntity, or by users who have higher level roles over parent objects. The GOCDB5 administrators bootstrap this process by granting the initial role requests. Projects and users then subsequently manage their own role requests.
- A Role has a defined type. Different role types are used to define varying permissions, each enabling different 'Actions' over a target OwnedEntity. Role types include 'Site Administrator,' 'Site Ops Manager,' 'NGI Security Officer,' 'Chief Operations Officer' etc (not all listed here).
- Current Actions include EDIT_OBJECT, DELETE_OBJECT, GRANT_ROLE, REVOKE_ROLE, NGI_ADD_SITE (not all Actions are listed here).
- The role model is flexible and can be customised by adding new Role types and defining new Actions.
- The role model is simple to use by invoking the 'authoriseAction()' method and passing the requesting user and the required action that affects the target OwnedEntity, e.g. 'authoriseAction(Action::EDIT_OBJECT, $site, $user)'.

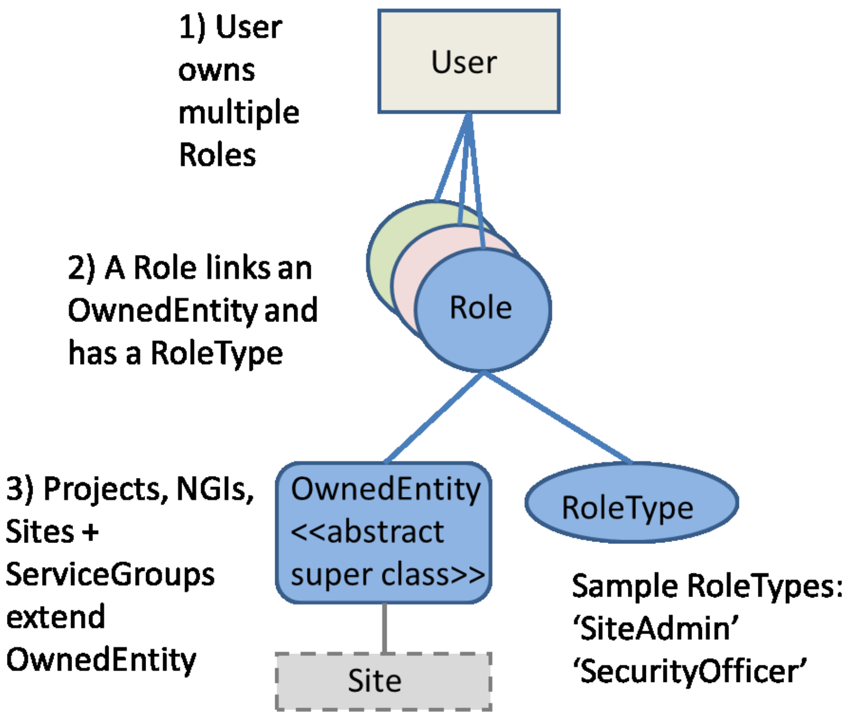

1) User owns multiple Roles
User
2) A Role links an OwnedEntity and has a RoleType
Role
3) Projects, NGIs, Sites + ServiceGroups extend OwnedEntity
OwnedEntity <<abstract super class>>
RoleType
Site
Sample RoleTypes: 'SiteAdmin' 'SecurityOfficer'

Figure 3. GOCDB5 Role model (simplified)

Programmatic Interface

- GOCDB5 provides a comprehensive REST style programmatic interface (PI[3]) for querying the data in XML. Methods include; get_downtime, get_site, get_ngi, get_service_endpoint, get_site_contacts, get_service_types (not all methods are listed here).
- Queries can be refined by passing different URL parameters to narrow results, e.g. 'get_site&sitename=RAL_LCG2' to return just the specified site or 'get_site&country=UK' to return all UK sites.
- A number of core methods support the 'scope' and 'scope_match' parameters. The 'scope' parameter is used to specify a comma separated list of scope tags, and the 'scope_match' parameter is used to specify the value 'any' or 'all' ('any' means match all resources that define any of the specified scope tags, 'all' means only match those resources that define all of the specified scope tags). For example:
    - 'get_service_endpoint&scope=EGI,CLIP,PROJX&scope_match=any' (return all services that define either EGI, CLIP and/or PROJX scope tags)
    - 'get_site&scope=EGI,EUDAT&scope_match=all' (return all sites that define both EGI and EUDAT scope tags)

Authentication Abstractions

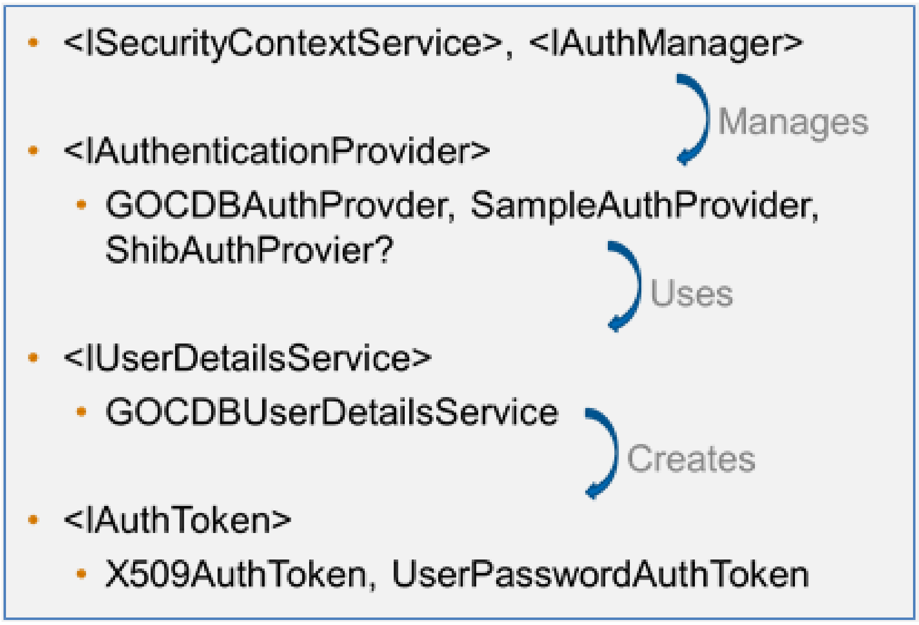
GOCDB5 includes core authentication abstractions to facilitate the plug-in of different user authentication mechanisms such as x509 certificates and username/passwords. These core abstractions have been copied from the Spring Security 3 framework for use in GOCDB5 (disclaimer: in no way is the GOCDB5 PHP security module a full implementation of Spring Security 3[4], rather, it is a micro-super simplification).

In brief (and summarised in Figures 4 and 5):

- The ISecurityContextService is used to store the users IAuthToken in HTTP session so that re-authentications are not necessary across different page requests (this requires cookies are enabled in the browser).
- The IAuthenticationProvider interface authenticates the user if their IAuthToken is null.
- The IUserDetailsService abstracts the local credential store (e.g. a local database that stores user accounts identified by certificate DN or username).

The authentication module does not perform authorisation; this is delegated to the role model. Sample usage of the authentication API is shown in Figure 5. The EGI implementation uses x509 certificates and so in this scenario, we do not use the authentication abstractions in the EGI GOCDB5 instance as this allows us to bypass the use of cookies and avoid EU cookie-law regulations. This would not be possible however if using username and password authentication as the user's AuthToken would need to be stored in HTTP session which requires cookies. There is plenty of scope to further refine and improve the authentication abstractions for use in different projects.

[3] https://wiki.egi.eu/wiki/GOCDB/PI/Technical_Documentation
[4] http://docs.spring.io/spring-security/site/index.html



<ISecurityContextService>, <IAuthManager>
Manages
<IAuthenticationProvider>
GOCDBAuthProvder, SampleAuthProvider, ShibAuthProvier?
Uses
<IUserDetailsService>
GOCDBUserDetailsService
Creates
<IAuthToken>
X509AuthToken, UserPasswordAuthToken

Figure 4. Core abstractions in the authentication module

```
$auth = SecurityContextService::getAuthentication();
if($auth == null) {
   // Here you could re-direct the user to some form of auth/login pag
   // to construct an Auth token from the user, then call:
   AuthenticationManagerService::authenticate($anIAuthenticationObj);
} else {
    $auth = SecurityContextService::getAuthentication();
    $auth->getPrinciple(); // to get the authenticated principle objec
    $auth->getDetails(); // to get an application specific user detail
    $auth->getAuthorities(); // to list granted roles
}

//An explicit authentication can be achieved using the following code:
SecurityContextService::setAuthentication($anIAuthenticationObj);

//An explicit logout (removal of the security context):
SecurityContextService::setAuthentication(null);
```

Figure 5. Sample usage of the Authentication API

Future Work

Current and future developments[5] are focussing on an extensibility mechanism[6] and a GLUE 2 rendering of the GOCDB5 data[7].

Extensibility Mechanism

This will allow property bags (custom key-value pairs) to be added to different entities such as Sites and Services. In doing this, queries could be filtered by custom key-value pairs in the PI. This is similar to the scope-tag parameters but differs in that custom key-value pairs can be freely added and modified by users. Conversely, scope-tags can only be added by the GOCDB admins for subsequent use by users (users cannot define new scope tags).

GLUE 2 Rendering of GOCDB5 Data

The structure of the core entity relationship diagram (ERD) closely resembles a sub-set of the GLUE 2 data model, especially in terms of the entities each model defines and their relationships. However, for a GLUE 2 rendering, new GLUE 2 attributes will need to added to the existing GOCDB5 entities such as the 'interfaceName' attribute for service endpoints (there are many more).

Useful Links

GOCDB Production EGI release: https://goc.egi.eu

GOCDB Primary failover: https://goc.dl.ac.uk

GOCDB Secondary failover: https://goc.itwm.fraunhofer.de

GOCDB Wiki and user documentation: https://wiki.egi.eu/wiki/GOCDB_Documentation_Index

Future Developments: https://wiki.egi.eu/wiki/GOCDB/Release4/Development

Programmatic Interface doc: https://wiki.egi.eu/wiki/GOCDB/PI/Technical_Documentation

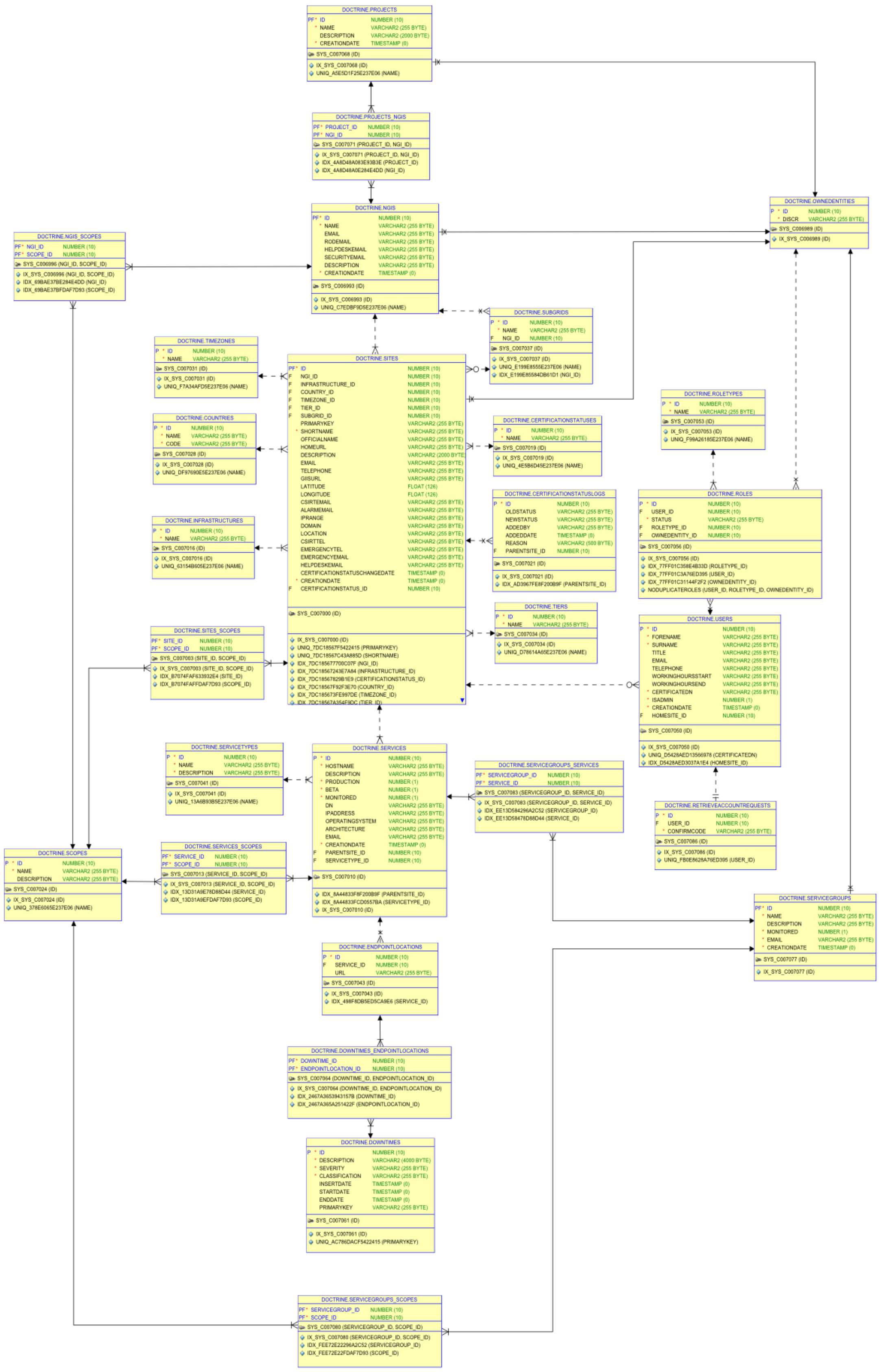
GOCDB5 SVN repo/src:

https://www.sysadmin.hep.ac.uk/svn/grid-monitoring/branches/gocdb/Doctrine%20Web%20Portal

(Please note, a packaged and tagged release will be made available for download soon, in the meantime, the code is available only through the SVN repo)

[5] https://wiki.egi.eu/wiki/GOCDB/Release4/Development
[6] https://wiki.egi.eu/wiki/GOCDB/Release4/Development/ExtensibilityMechanism
[7] https://wiki.egi.eu/wiki/GOCDB/Release4/Development/GLUE2Compatibility

Appendix 1 – GOCDB5 Entity Relationship Diagram (ERD)


DOCTRINE.PROJECTS
DOCTRINE.PROJECTS_NGIS
DOCTRINE.NGIS
DOCTRINE.NGIS_SCOPES
DOCTRINE.OWNEDENTITIES
DOCTRINE.SUBGRIDS
DOCTRINE.TIMEZONES
DOCTRINE.SITES
DOCTRINE.COUNTRIES
DOCTRINE.CERTIFICATIONSTATUSES
DOCTRINE.ROLETYPES
DOCTRINE.INFRASTRUCTURES
DOCTRINE.CERTIFICATIONSTATUSLOGS
DOCTRINE.ROLES
DOCTRINE.TIERS
DOCTRINE.SITES_SCOPES
DOCTRINE.USERS
DOCTRINE.SERVICETYPES
DOCTRINE.SERVICES
DOCTRINE.SERVICEGROUPS_SERVICES
DOCTRINE.RETRIEVEACCOUNTREQUESTS
DOCTRINE.SCOPES
DOCTRINE.SERVICES_SCOPES
DOCTRINE.SERVICEGROUPS
DOCTRINE.ENDPOINTLOCATIONS
DOCTRINE.DOWNTIMES_ENDPOINTLOCATIONS
DOCTRINE.DOWNTIMES
DOCTRINE.SERVICEGROUPS_SCOPES