GAARDS Architecture Specification

*Revision: 1.3*

Table of Contents

Table of Contents ii

Revision History iv

Approved By iv

List of Figures 1

1. Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Overview 3

1.4 Definitions and Acronyms 4

2. Architectural Considerations 6

2.1 SOA 7

2.2 Web service 7

2.3 SAML 8

2.4 Globus Toolkit 8

2.5 GSI 8

2.6 WS-Resource Framework 9

2.7 Metadata 11

2.7.1 ServiceMetadata 12

2.7.2 SecurityMetadata 13

2.8 Advertisement, Discovery and Index Service 14

3. Architecture 15

3.1 Overview 15

3.2 Authentication Service 18

3.2.1 AuthenticationProfiles 18

3.2.2 AuthenticationProvider 22

3.2.3 SubjectProvider 22

3.2.4 SAMLProvider 22

3.2.5 Configuration 23

3.2.6 Interfaces 23

3.2.7 Operations 24

3.2.8 Data Types 25

3.2.9 Error Handling 25

3.2.10 Client API 27

3.3 Dorian 28

3.3.1 Dorian Service 30

3.3.1.1 Use cases 30

3.3.1.2 Interfaces 33

3.3.1.3 Metadata 34

3.3.1.4 Operations 35

3.3.1.5 Data types 48

3.3.1.6 Error Handling 49

3.3.1.7 Data Model 51

3.3.1.8 Identity Federation Service (IFS) 54

3.3.1.9 Dorian Identity Provider 60

3.3.1.10 Dorian Certificate Authority 64

3.3.1.11 Host Certificates 68

3.3.1.12 Dorian Client API 69

3.3.2 Dorian Administration 69

3.3.2.1 Trusted Identity Providers 70

3.3.2.2 Grid User Management 70

3.3.2.3 Dorian IdP Users 70

3.3.3 Dorian Configuration 71

3.3.4 SAML Assertion 73

3.3.4.1 Authentication Statement 73

3.3.4.2 Attribute Statement 74

3.3.4.3 Sample Dorian SAML Assertion 76

3.4 Grid Grouper 77

3.4.1 Use cases 79

3.4.2 Interfaces 82

3.4.3 Operations 82

3.4.4 Data Types 95

3.4.5 Error handling 95

3.4.6 Data Model 98

3.4.7 Client API 100

3.4.8 Configuration 100

3.5 Grid Trust Service 101

3.5.1 Level of Assurance 102

3.5.2 Use cases 103

3.5.3 Interfaces 105

3.5.4 Operations 105

3.5.5 Data Model 114

3.5.6 Data Types 116

3.5.7 Error Handling 118

3.5.8 Client API 121

3.5.9 Federation 122

3.5.10 Configuration 123

3.5.11 SyncGTS 123

3.6 Credential Delegation Service 127

3.6.1 Use cases 129

3.6.2 Interfaces 130

3.6.3 Operations 133

3.6.4 DelegatedCredentialService 137

3.6.5 Delegating a Credential 138

3.6.6 Obtaining a delegated credential 141

3.6.7 Administration 143

3.6.8 Data types 143

3.6.9 Error handling 143

3.6.10 Configuration 144

3.6.11 Data Model 145

3.6.12 Client API 146

3.7 WebSSO 147

3.7.1 JA-SIG's CAS 147

3.7.2 WebSSO Workflow 149

3.7.3 Configuration 152

3.7.4 Secure Communication 153

3.8 Auditing 154

3.8.1 Dorain 154

3.8.2 CDS 156

4. Deployment 157

5. References 159

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Name** | **Reason For Changes** | **Date** |
| *DRAFT* | Prasad Konka | Initial Version | 10/30/2009 |
| *DRAFT* | Prasad Konka | Changes based on John Eisenschmidt comments | 11/16/2009 |
| *DRAFT* | Prasad Konka | Changes based on Scott Oster comments  Removed implementation details  Added decoupled nature of GAARDS | 11/19/2009 |
| *DRAFT* | Stephen Langella | Initial Review of Document | 11/20/2009 |
| *DRAFT* | Prasad Konka | Changes based on Stephen Langella, Scott Oster and Marty Humphrey comments | 12/03/2009 |
| *1.3* | Prasad Konka | Final Version | 12/09/2009 |

Approved By

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Signature** | **Department** | **Date** |
|  |  |  |  |
|  |  |  |  |

# List of Figures

[*Figure 1 Grid Security Infrastructure* 10](#_Toc247950399)

[*Figure 2 WSRF Resource* 11](#_Toc247950400)

[*Figure 3 ServiceMetadata class diagram* 13](#_Toc247950401)

[*Figure 4 ServiceMetadata common components* 13](#_Toc247950402)

[*Figure 5 SecurityMetadata type* 14](#_Toc247950403)

[*Figure 6 GAARDS Security Infrastructure* 16](#_Toc247950404)

[*Figure 7 GAARDS components and roles* 18](#_Toc247950405)

[*Figure 8 AuthenticationService use case* 19](#_Toc247950406)

[*Figure 9 AuthenticationService activity diagram* 20](#_Toc247950407)

[*Figure 10 Authentication Service Sequence Diagram* 21](#_Toc247950408)

[*Figure 11 Authentication Service Interfaces Class Diagram* 23](#_Toc247950409)

[*Figure 12 Credential type* 25](#_Toc247950410)

[*Figure 13 SAMLAssertion type* 25](#_Toc247950411)

[*Figure 14 AuthenticationService client API* 27](#_Toc247950412)

[*Figure 15 Dorian usage scenario* 29](#_Toc247950413)

[*Figure 17 Dorian Architecture* 30](#_Toc247950414)

[*Figure 18 Dorian user use case diagram* 30](#_Toc247950415)

[*Figure 19 Dorian Admin use case diagram* 31](#_Toc247950416)

[*Figure 20 IFS Activity diagram* 32](#_Toc247950417)

[*Figure 21 Dorian Service class diagram* 33](#_Toc247950418)

[*Figure 22 DorianService class diagram* 48](#_Toc247950419)

[*Figure 23 Dorian Identity Provider data model* 52](#_Toc247950420)

[*Figure 24 IFS data model* 53](#_Toc247950421)

[*Figure 25 IFS Class diagram* 55](#_Toc247950422)

[*Figure 26 TrustedIdPManager class diagram* 56](#_Toc247950423)

[*Figure 27 UserManager class diagram* 57](#_Toc247950424)

[*Figure 28 UserCertificateManager class diagram* 58](#_Toc247950425)

[*Figure 29 HostCertificateManager class diagram* 59](#_Toc247950426)

[*Figure 30 Dorian Identity Provider class diagram* 61](#_Toc247950427)

[*Figure 31 IdentityProvider class diagram* 62](#_Toc247950428)

[*Figure 32 AssertionCredentialsManager class diagram* 62](#_Toc247950429)

[*Figure 33 Dorian CA class diagram* 66](#_Toc247950430)

[*Figure 34 CertificateAuthority class diagram* 67](#_Toc247950431)

[*Figure 35 Dorian Client API class diagram* 69](#_Toc247950432)

[*Figure 36 Dorianconfiguration class diagram* 71](#_Toc247950433)

[*Figure 37 Authentication statement* 73](#_Toc247950434)

[*Figure 38 Authentication method* 74](#_Toc247950435)

[*Figure 39 Subject* 74](#_Toc247950436)

[*Figure 40 Attribute statement* 76](#_Toc247950437)

[*Figure 41 Grid Grouper* 78](#_Toc247950438)

[*Figure 42 GridGrouper use case digram* 79](#_Toc247950439)

[*Figure 43 GridGrouper use case diagram* 80](#_Toc247950440)

[*Figure 44 Grid Grouper use case diagram* 81](#_Toc247950441)

[*Figure 45 Grid Grouper Interfaces* 82](#_Toc247950442)

[*Figure 46 GridGrouper class diagram* 95](#_Toc247950443)

[*Figure 47 Grouper data model* 99](#_Toc247950444)

[*Figure 48 Grid Trust Service* 102](#_Toc247950445)

[*Figure 49 GTS Administrator use cases* 104](#_Toc247950446)

[*Figure 50 GTS CA Administrator use cases* 104](#_Toc247950447)

[*Figure 51 GTS use case* 104](#_Toc247950448)

[*Figure 52 GTS Interfaces* 105](#_Toc247950449)

[*Figure 53 GTS class diagram* 106](#_Toc247950450)

[*Figure 54 GTS data model* 114](#_Toc247950451)

[*Figure 55 GTS TrustedAuthority type* 116](#_Toc247950452)

[*Figure 56 GTS permission type* 116](#_Toc247950453)

[*Figure 57 GTS TrustedAuthorityFilter type* 117](#_Toc247950454)

[*Figure 58 GTS TrustLevel type* 117](#_Toc247950455)

[*Figure 59 GTS AuthorityGTS type* 118](#_Toc247950456)

[*Figure 60 GTS client class diagram* 121](#_Toc247950457)

[*Figure 61 GTS Federation* 122](#_Toc247950458)

[*Figure 62 SyncGTS* 124](#_Toc247950459)

[*Figure 63 SyncGTS activity diagram* 125](#_Toc247950460)

[*Figure 64 SyncGTS class diagram* 126](#_Toc247950461)

[*Figure 65 Credential Delegation service usage* 127](#_Toc247950462)

[*Figure 66 Credential Delegation service architecture* 128](#_Toc247950463)

[*Figure 67 Credential delegation use case* 129](#_Toc247950464)

[*Figure 68 CDS Interfaces* 130](#_Toc247950465)

[*Figure 69 CredentialDelegationService class diagram* 133](#_Toc247950466)

[*Figure 70 CredentialDelegation sequence diagram* 139](#_Toc247950467)

[*Figure 71 DelegationRequest* 140](#_Toc247950468)

[*Figure 72 DelegationSigningRequest* 141](#_Toc247950469)

[*Figure 73 DelegationSigningResponse* 141](#_Toc247950470)

[*Figure 74 Obtaining delegated credential sequence diagram* 142](#_Toc247950471)

[*Figure 75 DelegationRecordFilter type* 142](#_Toc247950472)

[*Figure 76 CDS data model* 145](#_Toc247950473)

[*Figure 77 CDS client API* 146](#_Toc247950474)

[*Figure 78 CAS workflow* 148](#_Toc247950475)

[*Figure 79 WebSSO architecture* 149](#_Toc247950476)

[*Figure 80 WebSSO workflow* 150](#_Toc247950477)

[*Figure 81 GAARD Production Deployment* 157](#_Toc247950478)

[*Figure 82 GAARDS dependencies* 158](#_Toc247950479)

# Introduction

## Purpose

The purpose of this document is to provide architectural specification for caGrid GAARDS 1.3 security framework.

## Scope

This specification focus is on the GAARDS 1.3 framework components and their architectural details as they are working. This document is not exhaustive in presenting all of the Unified Modeling Language (UML) diagrams that might be included in software architecture documentation. Instead, it includes only the most important diagrams that will enable the developer to sufficiently understand the software architecture to work with it.

This document is targeted at the stakeholder – management, architect, developer and 3rd party implementer – to aid in developing a high-level familiarity with the GAARDS architecture as well as a more detailed understanding to support implementation. The GAARDS framework provides rich set of UI tools for users and administrators are not part of this specification.

## Overview

Grid computing refers to the notion of using distributed resources hosted at multiple institutions to solve large scale, challenging problems in science and engineering. In “The Anatomy of the Grid” by Ian Foster, Carl Kesselman and Steven Tuecke wrote, “The real and specific problem that underlies the Grid concept is *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a *virtual organization*.”

caGrid is a grid computing platform, made up of standards, tools, and middleware infrastructures, for sharing data and analytical resources as well as computation and storage systems. caGrid is an open source platform aimed at enabling secure data sharing and analysis among institutions. caGrid supports a wide range of use cases in basic, translational, and clinical research.

In a federated, collaborative environment, information security plays a critical role in safeguarding privileges while maintaining confidentiality, integrity, and availability of information. On the caGrid, security requirements exist both for protecting intellectual property and ensuring protection and privacy of patient data and other sensitive information. caGrid provides comprehensive security support through its Grid Authentication and Authorization with Reliably Distributed Services (GAARDS) infrastructure. GAARDS provides services and tools for the administration and enforcement of security policy in an enterprise Grid. GAARDS, like all of caGrid, was developed on top of the [Globus Toolkit](http://www.globus.org/), and extends Globus Grid Security Infrastructure ([GSI](http://www.globus.org/security/overview.html)).

GAARDS services include:

**Dorian** – Allows users to login to the caGrid

**Authentication Service** – Integrates existing institutional login capabilities with the caGrid

**Grid Grouper** – Allows institutions to implement group-based security policies

**Grid Trust Service** – Establishes and maintains trust relationships

**Credential Delegation Service** – Enables users/services (delegator) to delegate their Grid credentials to other users/services (delegatee) such that the delegatee(s) may act on the delegator's behalf.

**Web Single Sign-On** – Allows a single login to provide access to multiple web applications that utilize Grid services

## Definitions and Acronyms

**Architecture**

1. The software architecture of a program or computing system is the structure or structures of the system. This structure includes software components, the externally visible properties of those components, the relationships among them and the constraints on their use. (based on the definition of architecture in [[Soft Arch Pract]](http://www.w3.org/TR/ws-gloss/#SAP))
2. A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture. [[Fielding]](http://www.w3.org/TR/ws-gloss/#RoyFieldingThesis)

**Authentication**

Authentication is the process of verifying that a potential partner in a conversation is capable of representing a person or organization [W3C].

**Authorization**

The process of determining, by evaluating applicable access control information, whether or not a subject is permitted to access a particular resource. Usually, authorization is in the context of authentication. Once a subject is authenticated, it may be authorized to perform different types of access [W3C].

**Configuration**

A collection of properties, which may be changed. A property may influence the behavior of an entity [W3C].

**Certification Authority (CA)**

The entity / system that issues X.509 identity certificates (places a subject name and public key in a document and then digitally signs that document using the private key of the CA).

**CRL**

A certificate revocation list (CRL) is a list of certificates (or more specifically, a list of serial numbers for certificates) that have been revoked or are no longer valid, and therefore should not be relied upon.

**Encryption**

Cryptographic transformation of data (called "plaintext") into a form (called "ciphertext") that conceals the data's original meaning to prevent it from being known or used. If the transformation is reversible, the corresponding reversal process is called "decryption", which is a transformation that restores encrypted data to its original state. [[RFC 2828]](http://www.w3.org/TR/ws-gloss/#RFC2828)

**Grid**

A Grid is an interconnected set of computing resources for the purpose of data sharing and analysis by a virtual organization. For caGrid and caBIG® specifically, a Grid links computer and data resources of multiple organizations to securely share and analyze vast quantities of data. Computers, servers, or databases are shared as Grid services and join a target Grid by conforming to the Grid's data sharing and security policies.

**Grid Identity**

Grid identity is a global unique identifier on caGrid assigned by Dorian. It is the combination of “Dorian CA subject”, “Identity Provider Name” and “Local User Id”. Local user id is a unique string in authentication authority domain.

Example: /O=OSU/OU=BMI/OU=caGrid/OU=Dorian/OU=localhost/OU=OSU/CN=jdoe

Dorian CA Subject IdP Name Local User Id

**IdP**

Identity Provider (IdP) is an entity that asserts the identity of a user of an electronic information system.

**J2EE**

The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications. The J2EE platform simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

**Loose coupling**

Coupling is the dependency between interacting systems. This dependency can be decomposed into real dependency and artificial dependency:

1. Real dependency is the set of features or services that a system consumes from other systems. The real dependency always exists and cannot be reduced.
2. Artificial dependency is the set of factors that a system has to comply with in order to consume the features or services provided by other systems. Typical artificial dependency factors are language dependency, platform dependency, API dependency, etc. Artificial dependency always exists, but it or its cost can be reduced.

Loose coupling describes the configuration in which artificial dependency has been reduced to the minimum [W3C].

**MVC**

Model-View-Controller is an architectural pattern used in software engineering. The pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each.

**Spring Framework**

Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

**Web service**

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [W3C].

**XML**

XML (Extensible Markup Language) is a set of rules for encoding documents electronically. It is defined in the [XML 1.0 Specification](http://www.w3.org/TR/REC-xml) produced by the W3C and several other related specifications; all are fee-free open standards. It is a textual data format, with strong support via Unicode for the languages of the world.

**XSD**

XML Schema Definition, a Recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) document. XML Schema Definition expresses shared vocabularies and allows machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents in more detail.

# Architectural Considerations

caGrid is a service oriented system. In a service oriented system, each resource is made available to the (Grid) environment as a service. A service wraps the functionality of the resources in a set of well-defined interfaces. These interfaces (and the associated client side application programming interfaces) are used by client applications to interact with the resource. The inherent benefit of Service Oriented Architecture (SOA) is the loose coupling between the producer and the consumer, which eases the construction of component-based solutions and promotes abstraction. To preserve loose coupling, security must also be implemented as a service - to avoid tightly bound security and thereby tight binding of the services themselves. Tightly bound security scheme however will not work for SOA where services consumers and producers are distributed outside a single security domain and loosely coupled. Key to this loose coupling is Federated Identity Management with standardized mechanisms and formats for the communication of identity information between the domains; the Security Assertion Markup Language (SAML) defines such a standard.

The current GAARDS implementation uses SAML 1.1 assertions as the enabling mechanism for federating users from local institutions to the caGrid.

GAARDS framework is based on service-oriented-architecture design principles and web service interfaces. This architecture enables individual components to be replaced by custom solutions as long as they adhere to the defined web service interface specifications. It also allows implementations to be hosted on different hardware and software platforms, as well as services to be implemented using different programming languages.

GAARDS framework has been built on top of the Globus Toolkit that supports security via Grid Security Infrastructure (GSI). GSI leverages X.509 certificates for authentication. X.509 certificates are an implementation of PKI, which binds public keys with respective user identities by means of a certificate authority (CA). The user identity must be unique across CA participating in a federation. The binding is established through the registration and issuance process, which, depending on the level of assurance the binding has, may be carried out by software at a CA, or under human supervision. For each user, the user identity, the public key, their binding, validity conditions and other attributes are made unforgeable in public key certificates issued by the CA. With PKI, a user's credentials consist of a public X.509 certificate and a private key. These credentials are generally contained in two encoded files, one for the certificate and one for the private key. It is the responsibility of the party owning the credential to keep their private key a secret.

Following sections describes overview of SOA, Webservices, SAML, Globus Toolkit, GSI, WS Resource Framework, Metadata and Index service. These are important foundational components / frameworks / architecture on which GAARDS framework is built up on. You can skip to [section 3](#_Architecture) if you are already familiar with these topics.

## SOA

Service Oriented Architecture (SOA) is an architecture framework in which the functionality of a software component is accessible remotely and programmatically via well-defined interfaces. A service oriented architecture environment consists of software components (e.g., applications, tools, databases) that are loosely coupled to other software components and exchange information with each other and clients through messages. The most common realization of SOA is Web Services.

In contrast, SOA is a form of distributed systems architecture that is typically characterized by the following properties:

* Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.
* Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.
* Description orientation: A service is described by machine-processable metadata. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.
* Granularity: Services tend to use a small number of operations with relatively large and complex messages.
* Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.
* Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

## Web service

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. These standards provide a common approach for defining, publishing, and using web services.

Web services are the preferred standards-based way to realize SOA, where the basic unit of communication is a message, rather than an operation.

GAARDS infrastructure exposes its functionality as web service interfaces.

## SAML

SAML, developed by the Security Services Technical Committee of the Organization for the Advancement of Structured Information Standards (OASIS), is an XML-based framework for communicating user authentication, entitlement, and attribute information. As its name suggests, SAML allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject (an entity that is often a human user) to other entities, such as a partner company or another enterprise application. SAML is a flexible and extensible protocol designed to be used – and customized if necessary – by other standards. SAML has emerged as the gold standard for federated identity. By defining standardized mechanisms for the communication of security and identity information between business partners, SAML makes federated identity, and the cross domain transactions that it enables a reality.

SAML can act to push responsibility for proper management of identities to the identity provider, which is more often compatible with its business model than that of a service provider. Using SAML to "reuse" a single act of authentication (such as logging in with a username and password) multiple times across multiple services can reduce the cost of maintaining account information. This burden is transferred to the identity provider.

An *assertion* is a package of information that supplies one or more statements made by a SAML authority. SAML defines three different kinds of assertion statement that can be created by a SAML authority.

• **Authentication:** The specified subject was authenticated by a particular means at a particular time. This kind of statement is typically generated by a SAML authority called an identity provider, which is in charge of authenticating users and keeping track of other information about them.

• **Attribute:** The specified subject is associated with the supplied attributes.

• **Authorization Decision:** A request to allow the specified subject to access the specified resource has been granted or denied.

## Globus Toolkit

The open source Globus® Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when.

GT4.0 supports both message-level (support for the WS-Security standard and the WSSecureConversation) and transport-level security (authentication via TLS with support for X.509 proxy certificates).

## GSI

Grid Security Infrastructure (GSI) is part of Globus toolkit uses public key cryptography (also known as asymmetric cryptography) as the basis for its functionality. A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service.

A GSI certificate includes four primary pieces of information:

* A subject name, which identifies the person or object that the certificate represents.
* The public key belonging to the subject.
* The identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject.
* The digital signature of the named CA.

The primary motivations behind the GSI are:

* The need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid.
* The need to support security across organizational boundaries, thus prohibiting a centrally-managed security system.
* The need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

GSI may be thought of as being composed of four distinct functions: message protection, authentication, delegation, and authorization. Implementations of different standards are used to provide each of these functions:

* TLS (transport-level) or WS-Security and WS-SecureConversation (messagelevel) are used as message protection mechanisms in combination with SOAP.
* X.509 End Entity Certificates or Username and Password are used as authentication credentials
* X.509 Proxy Certificates and WS-Trust are used for delegation
* SAML assertions are used for authorization

Overview of the GT4 Grid Security Infrastructure and standards used for different functions is as shown in the diagram below. The left two figures show message-level security, with X.509 credentials and username/password authentication. The figure on the right shows transport-level security with X.509 credentials.



*Figure 1 Grid Security Infrastructure*

## WS-Resource Framework

The Web Service Resource Framework (WSRF), a part of Globus Toolkit, is a set of six Web services specifications that define what is termed the WS-Resource approach to modeling and managing state in a Web services context. The purpose of the WSRF is to define a generic framework for modeling and accessing persistent resources using Web services so that the definition and implementation of a service and the integration and management of multiple services is made easier.

WSRF introduces the idea of an XML document description, called the Resource Properties document schema, which is referenced by the WSDL description of the service and which explicitly describes a view of the shopping cart, printer, print job, or whatever, with which the client interacts. A resource described in this way is called a WS-Resource. By exploiting the Resource Properties document schema, WSRF enables the mechanical definition of simple, generic messages which interact with the WS-Resource.

A WS-Resource is defined as the composition of a Web service and a stateful resource that is (i) expressed as an association of an XML document with defined type with a Web services portType, and (ii) addressed and accessed according to the implied resource pattern, a conventional use of WS-Addressing endpoint references. In the implied resource pattern, a stateful resource identifier is encapsulated in an endpoint reference and used to identify the stateful resource to be used in the execution of a Web service message exchange. The WS-Resource framework allows WS-Resources to be declared, created, accessed, monitored for change, and destroyed via conventional Web services mechanisms, but does not require that the Web service component of the WS-Resource that provides access to the associated stateful resources be implemented as a stateful message processor.

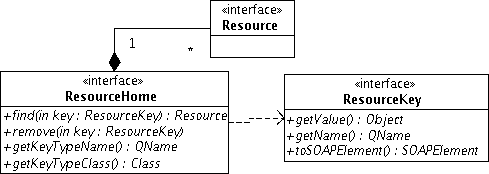
WS-Addressing standardizes the endpoint reference construct used to represent the address of a Web service deployed at a given network endpoint. An endpoint reference may contain, in addition to the endpoint address of the Web service, other metadata associated with the Web service such as service description information and *reference properties*, which help to further qualify the use of the Web service address. The reference properties of the endpoint reference play an important role in the implied resource pattern.

The implied resource pattern defines a conventional use of WS-Addressing in which a stateful resource is treated as an implied input for the processing of message exchanges implemented by a Web service. An endpoint reference that is described as following the implied resource pattern may include a ReferenceProperties child element that identifies the stateful resource to be used in the execution of all message exchanges performed using this EndpointReference. This type of endpoint reference is referred to as a *WS-Resourcequalified endpoint reference*. A request message directed to a Web service designated by a WS-Resource-qualified endpoint reference must include the ReferenceProperties information from the endpoint reference, as specified by WSAddressing.

Thus, the WS-Resource framework uses a WS-Resource-qualified endpoint reference to represent a “network-wide pointer” to a WS-Resource. A WS-Resource-qualified endpoint reference may be returned as a result of a Web service message request to a factory to create a new WS-Resource or, alternatively, from the evaluation of a search query on a service registry, or as a result of some application-specific Web service request.

The WS-Resource definition codifies the relationship between Web services and stateful resources in terms of the implied resource pattern, a set of conventions on Web services technologies, particularly XML, WSDL, and WS-Addressing [WSAddressing]. These conventions allow the state of a resource that participates in the implied resource pattern to be defined and associated with the description of a Web service interface. The state of a resource is defined in terms of a resource properties document.

A resource is represented by a Resource interface. It is a marker interface without any method defined. All resource objects must implement this interface. Resources are managed by an object that implements the ResourceHome interface. The ResourceHome interface provides methods for finding and removing resources as well as methods for identifying the SOAP header element and class for the resource key. A resource key is represented by a ResourceKey interface. It is a combination of a key name and the actual key value. In addition to the methods specified by the interface, ResourceHome implementations will generally provide an implementation-specific create() call or any other methods that operate on a set of resources.



*Figure 2 WSRF Resource*

Resources may have resource properties. Resource properties are declared in the WSDL of the service as elements of a resource property document. The ResourceProperties interface contains a single accessor method for retrieving the ResourcePropertySet from a resource. It must be implemented by all resources that want to expose resource properties. The ResourcePropertySet is the representation of the resource property document associated with the resource. It contains methods for managing the set of resource properties, e.g. adding and removing resource properties, and for discovering properties of the document itself, e.g. its name. The ResourceProperty interface needs to be implemented by all resource properties. It contains methods for: managing the set of values associated with the resource property, discovering properties of the resource property element, and serializing the resource property to a array of SOAP or DOM elements. The ResourcePropertyMetaData interface contains metadata information about a ResourceProperty such as resource property name, cardinality, etc.

Once metadata items are exposed as ResourceProperties, they can be queried using standard web service operations defined by the WS-ResourceProperties specification. Consult the specification for more details, but a synopsis of the operations is provided here:

* GetResourceProperty: allows access to the value of any resource property given its QName.
* GetMultipleResourceProperties: allows access to the value of several resource properties at once, given each of their QNames.
* QueryResourceProperties: allows complex queries on the resource properties document. Currently, the query language used is XPath.

## Metadata

In caGrid, both the client and service APIs are object oriented, and operate over well-defined and curated data types. Clients and services communicate through the grid using respectively Globus grid clients and service infrastructure. The grid communication protocol is XML, and thus the client and service APIs must transform the transferred objects to and from XML. This XML serialization of caGrid objects is restricted in that each object that travels on the grid must do so as XML which adheres to an XML schema registered in the Global Model Exchange (GME). GME defines the syntax of the XML serialization of them. Furthermore, Globus services are defined by the Web Service Description Language (WSDL). The WSDL describes the various operations the service provides to the grid. The inputs and outputs of the operations, among other things, in WSDL are defined by XML schemas (XSDs). As caBIG requires that the inputs and outputs of service operations use only registered objects, these input and output data types are defined by the XSDs which are registered in GME. In this way, the XSDs are used both to describe the contract of the service and to validate the XML serialization of the objects which it uses.

The caGrid metadata infrastructure consists of numerous components and services to:

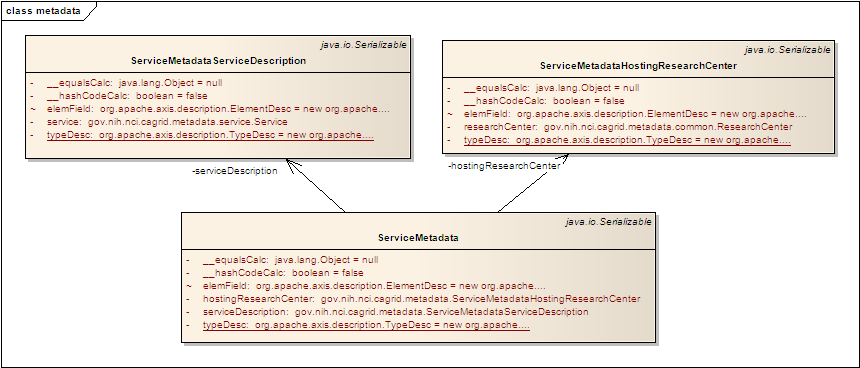
* Standardized service metadata models.
* Provides ability to generate and semantically annotate standard metadata using an external metadata repository like the caDSR.
* Acts as the authoritative repository the XML Schemas used on the grid.
* Provides the white and yellow pages of the grid.
* caDSR provides access to registered models and their semantic annotations.
* EVS provides access to controlled terminology.
* Introduce Support for caGrid metadata.
* Provides the means to advertise services.
* Provides the means to locate services and data of interest on the grid.

### ServiceMetadata

All caGrid Services publish a set of standard metadata, details the functionality of the service, and the institution providing it, and the points of contacts.  The Service Metadata describes the grid service, its hosting environment, and the underlying semantics of the data models used by the service's operations.

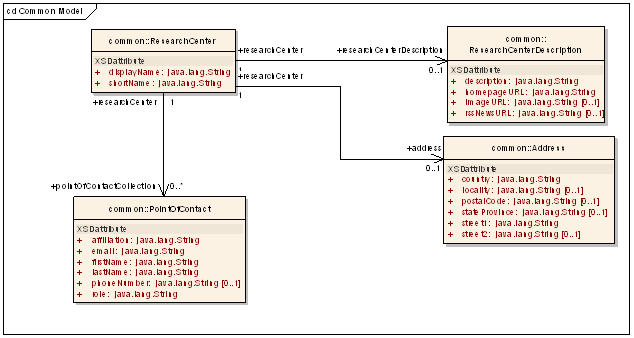
Several metadata-related features of caGrid manifest themselves as functionality in Introduce. Introduce is an open-source, extensible toolkit supporting easy development and deployment of WS/WSRF compliant Grid services. Introduce aims to reduce the service development and deployment effort by hiding low level details of the Globus Toolkit and to enable the implementation of strongly-typed Grid services. Introduce is the graphical service development environment used in caGrid, and supports an extension framework, whereby functionality can be plugged into Introduce dynamically. While such functionality could have alternatively been implemented directly in Introduce, this approach promotes a loose coupling between the components without lose of functionality of any difference to the end user.

ServiceMetadata class is the main entry point for the standard service metadata as shown below.



*Figure 3 ServiceMetadata class diagram*

Common components of service metadata are as given below in the diagram.

*Figure 4 ServiceMetadata common components*

ServiceMetadata schema can be referred at

<https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/metadata/schema/cagrid/types/>[caGridMetadata.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/metadata/schema/cagrid/types/caGridMetadata.xsd)

### SecurityMetadata

All caGrid grid services allow clients to dynamically determine the security requirements for a service such that they may configure themselves to appropriately connect to the service.

Service security metadata is made available anonymously via the grid service interface g*etServiceSecurityMetadata()* or through the grid service’s resource properties.

g*etServiceSecurityMetadata* ()

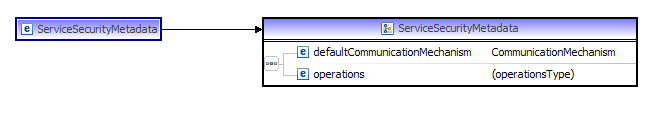
**Description:** Provides the client with the security information it needs to connect to the service.

**Input schema type:** GetServiceSecurityMetadataRequest

**Namespace:** http://security.introduce.cagrid.nci.nih.gov/ServiceSecurity

**Output schema type:** GetServiceSecurityMetadataResponse

**Namespace:** http://security.introduce.cagrid.nci.nih.gov/ServiceSecurity



*Figure 5 SecurityMetadata type*

Security schema can referred at

<https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/metadata/schema/cagrid/types/security/>[security.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/metadata/schema/cagrid/types/security/security.xsd)

More details on Metadata can be found at

<http://cagrid.org/display/metadata13/Documentation>

## Advertisement, Discovery and Index Service

The caGrid provides metadata infrastructure for a service advertisement, discovery and index services.

For the purposes of [Advertisement](http://cagrid.org/display/metadata13/Advertisement) and [Discovery](http://cagrid.org/display/metadata13/Discovery), caGrid leverages the Globus-provided Index Service. The Index Service implements the standard WS-ServiceGroup specification. When services are added to the service group, they specify what and how metadata should be accessed from them, and the Index Service performs this aggregation. Clients can then query this aggregated information using standard Resource Property operations. More information on these operations can be found [here](http://cagrid.org/display/metadata/caGrid+Metadata+in+WSRF).

caGrid services are expected to maintain soft-state registration to a well-known, Index Service instance, specifying polling of standard caGrid standard metadata. For more information, see the section on caGrid [Advertisement](http://cagrid.org/display/metadata13/Advertisement).

The Globus Information Services component, realized as the [Monitoring and Discovery System (MDS)](http://www.globus.org/toolkit/mds/) is a suite of web services to monitor and discover resources and services on Grids. This system allows users to discover what resources are considered part of a Virtual Organization (VO) and to monitor those resources. MDS services provide query and subscription interfaces to arbitrarily detailed resource data and a trigger interface that can be configured to take action when pre-configured trouble conditions are met. MDS is composed of the following three main components:

* WS MDS Index Service – This service contains a registry of grid resources and collects information from them, making it accessible and queryable from one location. Generally, a virtual organization deploys one or more index services, which then collect data on all of the grid resources available within that VO.
* WS MDS Trigger Service – This service collects data from grid resources and passes the data to appropriate programs to perform various actions in response to events. (not currently used by the caGrid metadata infrastructure).
* WS MDS Aggregator – This is the infrastructure on which the previous services are built. It collects, manages, and indexes data from an aggregator source and sends that data to an aggregator sink for processing.

For more information on the Index Service, see the Globus documentation (<http://www.globus.org/toolkit/docs/4.0/info/>).

Please visit following link for more details on caGrid Advertisement, Discovery and Index services.

<http://cagrid.org/display/metadata13/Design+Guide#DesignGuide-MetadataModelsOverview>

# Architecture

Identity management and federation is a critical problem in enabling access to resources across multiple security domains. In such an environment, it is highly unlikely that participating institutions use the same mechanism for managing the identities of their members. A common application of identity federation in the business domain is to enable single sign-on between two interacting companies that employ different internal authentication mechanisms, and vouch for their access to resources without needing to adopt the same security technologies or maintain a shared, centralized system for managing member identities. caGrid introduces additional challenges since it aims to facilitate the sharing of resources in a collaborative environment that spans multiple institutional boundaries. As a consequence, a Grid environment has to facilitate access by its users to resources at disparate institutions, while enforcing authentication and authorization policies set forth by the different institutions.

Identity federation provides a means for these organization services to agree on and establish common standards to refer to the user in order to share information across the organizational boundaries. The user is said to have a *federated identity*when organizations have established such an agreement on how to refer to the user. Identity Federation refers to the establishment of business agreements, cryptographic trust, and user identifiers or attributes across security and policy domains to enable more seamless cross-domain business interactions. Just as web services promise to enable integration between business partners through loose coupling at the application and messaging layer, federation does so at the identity management layer – insulating each domain from the details of the others authentication and authorization infrastructure.

"Think locally, act globally," nicely describes the federated model of identity management. In order to access protected resources at a service provider, users authenticate to their identity provider ("thinking locally" because they do not need to authenticate to a remote service provider, just an identity provider with which they have a closer trust relationship). Based on this authentication, they are then able to access resources at one or many service providers ("acting globally").

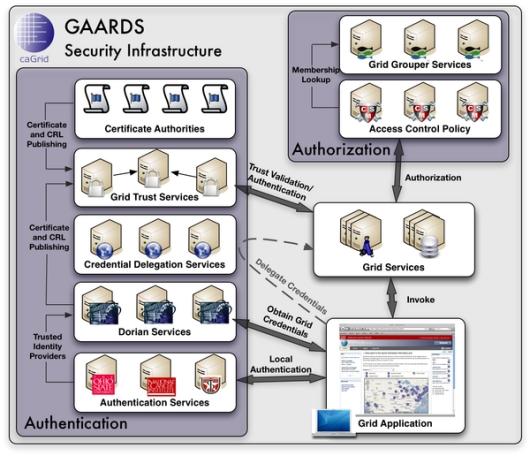
## Overview

Organizations invest a significant amount of resources into their existing identity management systems and already have processes in place for vetting user identities. In such settings, it would be more efficient to leverage existing identity management systems to provision Grid user accounts to access caGrid services. This scenario requires a mechanism to allow users to obtain Grid credentials using their existing organization-provided credentials. The mechanism should also remove the complications of using and managing Grid credentials.

Dorian is a Grid user management service that 1) hides the complexities of creating and managing Grid credentials from users and 2) provides a mechanism for users to authenticate using their institution's authentication mechanism. Dorian implements a complete, SOA solution for managing and federating user identities in a Grid environment based on public key certificates and SAML. Dorian uses SAML authentication assertions as the enabling mechanism for federating users from local institutions to the caGrid. The Security Assertion Markup Language (SAML) has been developed as a standard for exchanging authentication and authorization statements between security domains. Note that X.509 certificates and SAML assertions serve different purposes. SAML is mainly used between institutions for securely exchanging authentication information coming from trusted identity providers. The primary use of the certificates is to uniquely identify users and services, to facilitate authentication and authorization across multiple resource providers, and to enable secure delegation of credentials such that a service or a client program can access resources on behalf of the user.

The GAARDS framework supports security via Grid Security Infrastructure (GSI). GSI supports Grid identities in the form of X.509 certificates. When a user presents the certificate to a Grid service, the Grid service can use the public key embedded in the certificate to authenticate the user. The user keeps their private key secret, and uses their key to decode/encode messages during communication. X.509 certificates are used by both individuals (Grid users) and Grid services to identify themselves.

Following diagram shows GAARDS security infrastructure.



*Figure 6 GAARDS Security Infrastructure*

In order for users/applications to communicate with secure services, they first need X.509 credentials. Obtaining X.509 credentials requires having a User Account. Dorian provides two methods to register for a user account: 1) the user can register directly with Dorian, or 2) she can register indirectly via his/her existing user account obtained from a credential provider in another security domain. In order to obtain X.509 credentials via an existing user account, the credential provider must be registered in Dorian as a Trusted Identity Provider. While some users unaffiliated with an existing credential provider will register directly with Dorian, it is anticipated that most users will use their existing local credentials to obtain X.509 credentials. The advantages of this approach are: 1) users can use their existing credentials to access the Grid and 2) Administrators only need to manage a single account for a given user. In order to provision X.509 credentials, Dorian requires proof that local authentication succeeded, in the form of a SAML assertion. The GAARDS Authentication service provides a framework for existing credential providers to issue SAML assertions to Dorian. The authentication service also provides a uniform authentication interface upon which applications can be built. The user/application first authenticates with their local credential provider via the authentication service and obtains a SAML assertion as proof of successful authentication. They then use the SAML assertion to obtain X.509 credentials from Dorian. Assuming the local credential provider is registered with Dorian as a trusted identity provider and that the user’s account is in good standing with Dorian, it will issue X.509 credentials to the user. It should be noted that the use of the Authentication Service is not required; an alternative mechanism for obtaining the SAML assertion required by Dorian can be used. If a user is registered directly with Dorian, the user may contact Dorian directly to obtain X.509 credentials.

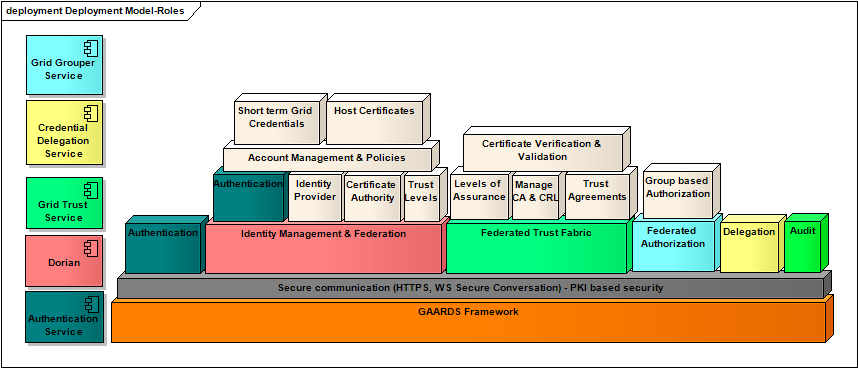
After a user has obtained X.509 credentials from Dorian, he/she may invoke a secure service. The user presents her credentials to the secure service. The service then validates user presented credentials. Part of the verification process is checking that the supplied Grid credentials were issued by a trusted Grid credential provider (i.e., Dorian or other certificate authorities). The Grid Trust Service (GTS) maintains a federated trust fabric of all the trusted digital signers in the Grid. Credential providers such as Dorian and other certificate authorities are registered as trusted digital signers and regularly publish new information (CRL, list of trusted authorities, etc) to the GTS. A service validates X.509 credentials against the trusted digital signers in a GTS.

Once a user has been authenticated to a secure service, the service determines if the user is authorized to call the desired operation. Services have many different options available to them for performing authorization. The GAARDS infrastructure provides two approaches that can either be used independently or together. It is important to note that, in addition to these two approaches, any other authorization approach

(i.e., user-developed authorization) can be used in conjunction with the GAARDS authentication/trust infrastructure. The first approach is group-based authorization provided by the Grid Grouper. In this approach, services and applications enforce authorization policy based on membership to globally accessible groups. Assuming the groups are provisioned by Grid Grouper, services can determine whether a caller is authorized by simply asking Grid grouper if the caller is in a given group. The second approach is role based authorization provided by the Common Security Module (CSM). In this approach,

Grid services ask CSM whether a user can perform a given operation on a specified resource. Based on the access control policy maintained in CSM, CSM decides whether or not a user is authorized. CSM enforces its group-based access control policy by asking Grid Grouper whether the caller is a member of the groups specified in the policy.

As explained above, GAARDS framework provides comprehensive security capabilities including authentication, identity federation, federated trust fabric, federated authorization and delegation with auditing capabilities. As shown in the below diagram, GAARDS provides AuthenticationService, Dorian, CredentialDelegationService, GTS, GridsGrouperService web service interfaces to serve its security capabilities via security communication. Combination of AuthenticationService and Dorian provides authentication, identity management and federation capabilities on the caGrid. GTS service provides the trust fabric between caGrid participants. It is a federated solution for registering and managing certificate authority certificates and CRLs, facilitating the enforcement of the most recent trust agreements, manage LOA, certificate verification and validation. CredentialDelegationService enables users/services to delegate their Grid credentials to other users/services to act on their behalf. This capability enables workflow conditions on the caGrid. GridGrouperService provides federated group/virtual organization based authorization to caGrid services and application based on group policies defined. Each of these services are described in detail in following sections. The order in which these services explained below doesn’t reflect the importance of it within GAARDS framework. Any of GAARDS components can be implemented independently based on the requirement. Dependencies between GAARDS components are explained in [Deployment](#_Deployment) section.



*Figure 7 GAARDS components and roles*

## Authentication Service

The Authentication Service enables existing identity providers to be seamlessly integrated into a production SOA environment such that users that are registered with an identity provider may use their existing credentials to obtain caGrid credentials. The Authentication Service provides a uniform web service interface providing applications with a single approach for authenticating users across a federation.

When a user authenticates with the Authentication Service, the Authentication Service returns a SAML Assertion. The SAML Assertion provides the application or consumer of the assertion the following:

* Proof that the user successfully authenticated.
* The method that the user authenticated with.
* Attributes containing information about the user.

Additional details and information about the contents of the SAML Assertion can be found at section 3.4.

Use case of authentication service is as given below. A user or service should be able to authenticate with local identity provider and assert the authentication with Dorian service to obtain X.509 credentials.



*Figure 8 AuthenticationService use case*

Together the Authentication Service and Dorian provide a solution for federating identity in a grid environment, allowing users to use their existing credentials to access secure grid resources. Details on Dorian specification can be found at section 3.3.

As it is mentioned above, one of the parts of SAML assertion is the method that the user authenticated with. User should be authenticated with a valid mechanism that AuthenticationService supports. AuthenticationService provides AuthenticationProfiles metadata element to discover valid authentication profiles supported. Following section describes AuthenticationProfiles metadata and its schema.

### AuthenticationProfiles

AuthenticationService supports multiple authentication profiles using WS Resource Framework. The AuthenticationProfiles metadata element specifies all authentication profiles that Authentication Service supports. An example of an authentication profile is username and password, another example is the Public Key Infrastructure (PKI) which uses X.509 certificates and private keys for authentication. Another example for AuthenticationProfile is one time password. The AuthenticationProfiles element provides a list of QName elements. Each QName represents the namespace and name of an XML schema element that specifies a credential type. Each QName listed in the AuthenticationProfiles metadata represent an authentication profile or credential that is supported by the AuthenticationService. In other words if the QName for a credential is listed in the AuthenticationProfiles metadata element, then that credential may used to authenticate using the Authentication Service's authenticateUser() operation. The authentication profiles are represented by a resource property (ex: (http://gaards.cagrid.org/authentication, AuthenticationProfiles)).

As it is explained in [section 2.6](#_WS-Resource_Framework), clients can access the value of AuthenticationProfiles resource property through standard web service operations defined by the WS-ResourceProperties specification. As it is explained in section 3.2.10, AuthenticationService client API provides convenient API, AuthenticationClient.getSupportedAuthenticationProfiles() to access supported authentication profiles.

Following is the AuthenticationProfiles schema.

<xs:complexType name="AuthenticationProfiles">

<xs:sequence>

<xs:element name="profile" minOccurs="1" maxOccurs="unbounded" type="xs:QName"/>

</xs:sequence>

</xs:complexType>

Following are the details on supported profiles by AuthenticationService:

QName: BASIC\_AUTHENTICATION

Namespace: http://gaards.cagrid.org/authentication

Schema type: BasicAuthentication

<xs:complexType name="BasicAuthentication">

<xs:complexContent>

<xs:extension base="gaards:Credential">

<xs:attribute name="userId" use="required" type="xs:string" />

<xs:attribute name="password" use="required" type="xs:string" />

</xs:extension>

</xs:complexContent>

</xs:complexType>

QName: *ONE\_TIME\_PASSWORD*

Namespace: http://gaards.cagrid.org/authentication

Schema type: OneTimePassword

<xs:complexType name="OneTimePassword">

<xs:complexContent>

<xs:extension base="gaards:Credential">

<xs:attribute name="userId" use="required" type="xs:string" />

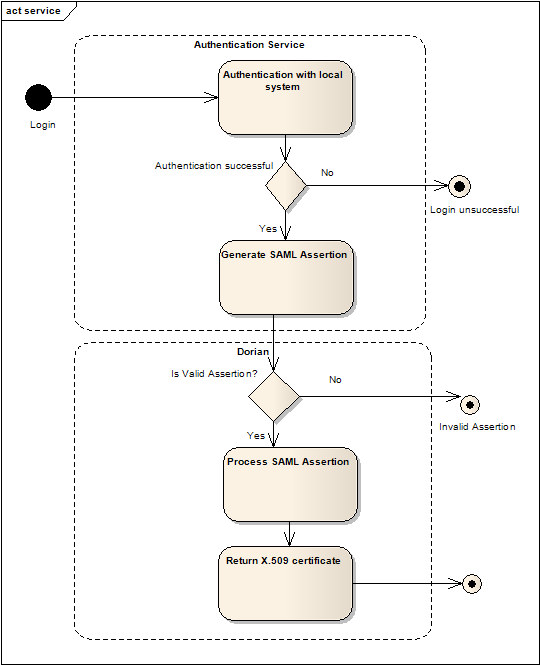
<xs:attribute name="oneTimePassword" use="required" type="xs:string" />

</xs:extension>

</xs:complexContent>

</xs:complexType>

Following activity diagrams shows activities in AuthenticationService and Dorian and their dependency.



*Figure 9 AuthenticationService activity diagram*

As shown in the sequence diagram below, a client communicate with the authentication service through its web service interface using HTTPS. The AuthenticationService implementation delegates all authentication requests to the AuthenticationProvider. The Authentication Service is built using the spring framework, allowing any of it modules to be easily replaced with custom implementation. The default AuthenticationProvider delegates the validation of the credential to the SubjectProvider. The SubjectProvider validates provided credential with organization's identity provider and obtains the Subject (user attributes). The AuthenticationProvider provides the Subject to the SAMLProvider which is responsible for encoding the user attributes provided by the AuthenticationProvider into a SAML Assertion. The SAMLProvider creates and signs the SAML Assertion and returns it to the AuthenticationProvider. The AuthenticationProvider returns the SAML Assertion to the web service interface, which then returns it to the client that made the authentication request.



*Figure 10 Authentication Service Sequence Diagram*

Partitioning the responsibility into three separate providers enables multiple integration options for identity providers.  For example: organizations whose Identity Provider can authenticate users and provide the required attributes, but do not provide support for issuing SAML assertions, should use the default AuthenticationProvider and SAMLProvider. They may be able to leverage the default SubjectProvider, if their identity provider is supported, otherwise they may choose to implement their own SubjectProvider.  Another example: if an organization's identity provider is able to issue the required SAML Assertions, they should provide their own implementation of the AuthenticationProvider, and ignore the SubjectProvider and SAMLProvider.

AuthenticationService component providers default implementation of AuthenticationProvider, SubjectProvider and SAMLProvider for faster and easy implementation. The Default Subject Provider provides out of the box capability for authenticating users and obtaining user attributes with several existing identity providers. The Default SAML Provider provides an out of the box solution for issuing the required SAML Assertions. The Default SAML Provider uses a user configured certificate and private key for issuing and signing SAML assertions. In cases where the Identity Provider does not have the ability to issue SAML Assertions, the Default SAML Provider will suffice the majority of the time.

### AuthenticationProvider

The AuthenticationProvider is called directly by the web service implementation and is the top most integration point for the Authentication Service. Following methods should be implemented by AuthenticationProvider implementation:

* **authenticate():** This method takes a credential as input and returns a SAMLAssertion. The credential should be used to authenticate the user with their organization's identity provider. The SAML Assertion should contain proof that the user successfully authenticated and the required a set of user attributes, mainly the user's local user id, first name, last name, and email address. For detailed information on the content required in the SAMLAssertion please refer the SAML Assertion section of this document.
* **getSupportedAuthenticationProfiles():** This method specifies the type of credentials that the AuthenticationProvider accepts as a valid means of authentication. This method does not require an input however returns a set of valid QName(s). Each QName corresponds to the XML schema definition for the credential.

### SubjectProvider

SubjectProvider is called by AuthenticationProvider to authenticate users with an identity provider. SubjectProvider is responsible for returning user attributes describing information about the user and method of authentication.

SubjectProvider interface requires implementation of following two methods:

* **getSubject():** This methodtakes a credential as input and returns a Subject. The credential should be used to authenticate the user with their organization's identity provider. Upon successfully authenticating the user, the SubjectProvider should create and return a Subject object. The Subject object should contain the following principles, each representing an attribute describing the user:

1. **gov.nih.nci.security.authentication.principal.LoginIdPrincipal** - The user's unique user id within their organization's identity provider.
2. **gov.nih.nci.security.authentication.principal.FirstNamePrincipal** - The user's first name.
3. **gov.nih.nci.security.authentication.principal.LastNamePrincipal** - The user's last name.
4. **gov.nih.nci.security.authentication.principal.EmailIdPrincipal** - The user's email address.

* **getSupportedAuthenticationProfiles()**: This method specifies the type of credentials that the AuthenticationProvider accepts as a valid means of authentication. This method does not require an input however returns a set of valid QName(s). Each QName corresponds to the XML schema definition for the credential.

### SAMLProvider

SAMLProvider is called by AuthenticationProvider to get SAMLAssertion for a Subject returned by SubjectProvider.

SAMLProvider interface requires implementation for following method:

* **getSAML():** This method takes a subject object containing the principles or user attributes needed to issue the SAML Assertion object.

### Configuration

The authentication-config.xml provides pluggable configuration to override default implementation of AuthenticationProvider, SubjectProvider and SAMLProvider. It is also placeholder to set properties related to private key, keystore used to sign a SAML assertion token.

Please visit following links for configuration details.

<https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/authentication-service/etc/>[authentication-config.xml](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/authentication-service/etc/authentication-config.xml)

<https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/authentication-service/etc/>[authentication.properties](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/authentication-service/etc/authentication.properties)

### Interfaces

Following class diagrams shows AuthenticationService dependency on different classes, their hierarchies and relations.



*Figure 11 Authentication Service Interfaces Class Diagram*

As shown above, the Authentication Service provides default implementations of AuthenticationProvider, Subjectprovider, and SAMLProvider. These interfaces must be implemented to change default behavior and to support customization. The DefaultAuthenticationProvider is made up of two sub-components; the SubjectProvider and the SAMLProvider. The SubjectProvider receives a credential and validates it, upon successful validation it returns the subject containing the attributes (UserId, First Name, Last Name, Email) that are required to be in the SAML Assertion. The SAML Provider consumes a subject returned by the Subject Provider, then creates and returns a Dorian-compliant SAML Assertion. The DefaultAuthenticationProvider isolates the required functionality of an AuthenticationProvider providing separate integration points. It requires the specification of a certificate and private key which it used to create and sign SAML Assertions based on subjects provided by the SubjectProvider.

### Operations

AuthenticationService provides following methods to interface with a client.

authenticateUser()

**Description***:* A client calling this method with Credential parameter would get authenticated and get a signed SAMLAssertion in return. Clients can determine which credentials are supported by the Authentication Service by obtaining the *AuthenticationProfiles* metadata from the AuthenticationService.

**Input:** org.cagrid.gaards.authentication.Credential

**Namespace:** http://cagrid.nci.nih.gov/1/authentication-service

**Schema type:** Credential

**Output:** gov.nih.nci.cagrid.opensaml.SAMLAssertion

**Namespace:** http://cagrid.nci.nih.gov/1/authentication-service

**Schema type:** SAMLAssertion

**Actors:** *User, Service*

**Exceptions:** AuthenticationProviderFault,

CredentialNotSupportedFault,

InsufficientAttributeFault,

InvalidCredentialFault

**Pre-condition:**

* + User has valid account with local authentication facility.

**Post-condition:**

* + User authentication is done locally with Trusted Identity Provider.
  + Signed SAMLAssertion is returned with attributes containing user information.
  + Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for AuthenticationService fault codes.

authenticate()

**Descrption:** This operation requires the client to specify a credential to authenticate with. **DEPRECATED** in version 1.3 of the Authentication Service

**Input:** gov.nih.nci.cagrid.authentication.bean.Credential

**Namespace:** http://cagrid.nci.nih.gov/1/authentication-service

**Schema type:** Credential

**Output:** gov.nih.nci.cagrid.authentication.bean.SAMLAssertion

**Namespace:** http://cagrid.nci.nih.gov/1/authentication-service

**Schema type:** SAMLAssertion

**Actors:** User, Service

**Exceptions:** InvalidCredentialFault,

InsufficientAttributeFault,

AuthenticationProviderFault

**Pre-condition:**

* + User has valid account with local authentication facility.

**Post-condition:**

* + User authentication is done locally with Trusted Identity Provider.
  + Signed SAMLAssertion is returned with attributes containing user information.
  + Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for AuthenticationService fault codes.

### Data Types

Following links provide different data types defined for AuthenticationService.

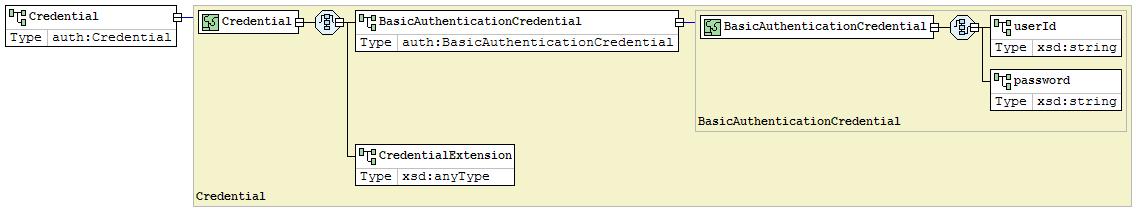
[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/authentication-core-types.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/authentication-core-types.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/authentication-service.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/authentication-service.xsd)

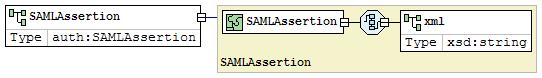
[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/AuthenticationServiceTypes.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/AuthenticationServiceTypes.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/cs-sstc-schema-assertion-1.1.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/authentication-service/schema/AuthenticationService/cs-sstc-schema-assertion-1.1.xsd)

Following diagrams shows data type definition of SAMLAssertion and Credential.



*Figure 12 Credential type*



*Figure 13 SAMLAssertion type*

### Error Handling

Following are the fault codes returned by AuthenticationService.

*AuthenticationProviderFault*: Returned if an internal error occurs during authenticateuser() operation

*<element name="AuthenticationProviderFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

*CredentialNotSupportedFault*: Returned if the supplied credentials format is not supported

*<element name="CredentialNotSupportedFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

*InsufficientAttributeFault*: Returned if the attributes returned by the underlying IdP are insufficient for identity federation

*<element name="InsufficientAttributeFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

*InvalidCredentialFault*: Returned if the supplied credentials are invalid.

*<element name="InvalidCredentialFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

### Client API

Authentication service framework provides following convenient client APIs to work with webservice interfaces. AuthenticationClient provides following methods:

setAuthorization():This method specifies an authorization policy that the client should us for authorizing the server that it connects to.

authenticate():This method authenticates with the authentication service using the supplied credential and returns SAMLAssertion.

getSupportedAuthenticationProfiles():This method obtains the authentication profiles supported by the

authentication service that the client is connecting to. The authentication profiles are represented by the resource property: (http://gaards.cagrid.org/authentication,AuthenticationProfiles). Client side authorization is not enforced when calling this method.



*Figure 14 AuthenticationService client API*

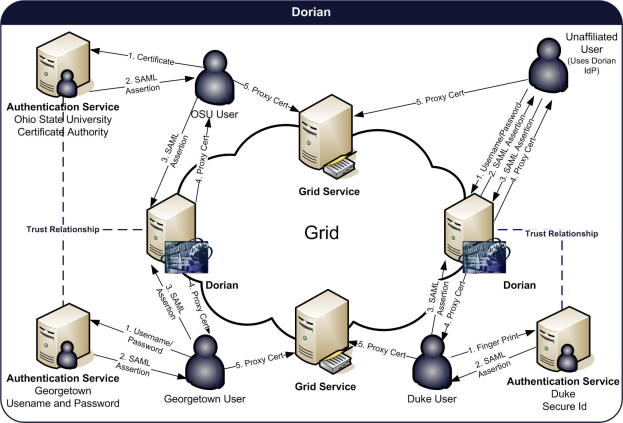
## Dorian

Dorian is an open source solution providing federated identity management. A Grid services environment generally spans across organizational boundaries with many users and services.  Identity vetting, authentication, and provisioning user and service credentials in large distributed environments are extremely complex and challenging problems.  Dorian alleviates these complexities and addresses these challenges by allowing organizations to integrate existing identity management systems into a Web/Grid services environment.   This enables users of these organizations to use their existing credentials to authenticate to Web/Grid Services.  In addition, Dorian provides a secure mechanism to issue and provision credentials for Web/Grid services.  This makes Dorian a complete identity management solution for Web/Grid service environments.  Below is a list of some of Dorian's salient features:

* Identity Federation
* Built-in Identity Provider
* Certificate Authority
* Compliance with [Federal e-Authentication Guidelineslinkext7](http://www.cio.gov/eauthentication/).
* Built-in Certificate Authority
* Integration with FIPS 140-2, Level 3, validated Hardware Security Module
* Auditing

To obtain X.509 credentials, users authenticate with their institution using the institution's conventional mechanism. Upon successfully authenticating the user, the local institution issues a digitally signed SAML assertion, vouching that the user has authenticated. The client then sends this SAML assertion to Dorian in exchange for X.509 credentials. Dorian will only issue X.509 credentials to users that supply a SAML assertion from a Trusted Identity Provider. Dorian's grid service interface provides mechanisms for managing trusted identity providers; this will be discussed in greater detail later in this document. For example, as shown in the Figure 15 below, where a Georgetown user wishes to invoke a grid service that requires credentials, they first supply the application with their username and password to the Georgetown IdP as they would normally do. The application client authenticates the Georgetown user with the Georgetown Authentication Service, receives a signed SAML assertion which it subsequently passes to Dorian in exchange for X.509 credentials. These credentials can then be used to invoke the services. This illustrates how Dorian can leverage an institution's existing authentication mechanism to federate its users to an SOA environment.

To facilitate smaller groups or institutions without an existing IdP, Dorian also has its own internal IdP. This allows users to authenticate to Dorian directly, thereby enabling them to authenticate with services. It provides administrators with facilities for approving and managing users. All of the Dorian IdP's functionality is made available through a service interface. Details of the Dorian IdP are provided later in this document. Figure 15 illustrates a scenario of a client using the Dorian IdP to authenticate to the Grid. In this scenario, the unaffiliated User wishes to invoke a secure service. Given that this unaffiliated user has registered and been approved for an account, she is able to authenticate with the Dorian IdP by supplying their username and password. Upon successfully authenticating the user, the Dorian IdP issues a SAML Assertion just like institutional IdPs, which can be presented to Dorian in exchange for X.509 credentials. The credentials can be used to invoke a secure service.



*Figure 15 Dorian usage scenario*

The high level architecture is illustrated in Figure 17. Dorian is built on top of the Globus Toolkit and runs as a WSRF [14] compliant grid service. Clients communicate with Dorian though its web service interface. All communication between clients and Dorian is secured via Transport Level Security (TLS) or WS-SecureConversation, depending on the deployment configuration.

The Dorian architecture consists of two core components: the Identity Federation Service (IFS) and the Dorian Identity Provider (IdP). The IFS component handles all the identity federation and management aspects for Dorian including the management of grid user accounts and Trusted IdPs. Dorian's internal IdP component provides the functionality for registering, authenticating, and managing users. Dorian provides a complete client API which provides complete programmatic access to all operations. Dorian also provides a complete graphical user interface.

Client



Dorian Grid Service Implementation

SOAP Handling Framework

Globus Container

**Identity Federation Service (IFS)**

Dorian Identity Provider (IDP)

Trusted IdP Manager

Grid User Manager

Grid Credentials Manager

Host Certificate Manager

Certificate Authority

Dorian IdP User Manager

SAML Asserter

IFS Requests

IdP Requests

Dorian Architecture

*Figure 17 Dorian Architecture*

### Dorian Service

As shown in figure 7, Dorian provides identity management and federation services to the caGrid participants. Dorian exposes all of its operations through its web service. It encapsulates all the functionality from all its subcomponents; Identity Federation and the Dorian Identity Provider.

#### Use cases

Following diagram shows different use cases for a user.



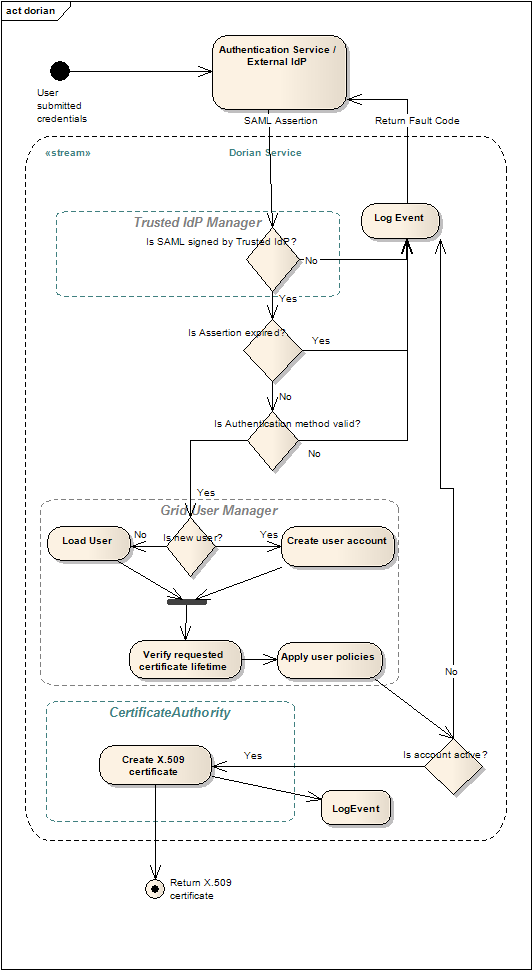
*Figure 18 Dorian user use case diagram*

Following diagram shows use cases for Dorian Administrator.

**

*Figure 19 Dorian Admin use case diagram*

As it is mentioned above, Dorian encapsulates all the functionality from all its subcomponents; Identity Federation and the Dorian Identity Provider. Following activity diagram shows the flow of activities involved with in different subcomponents generating a X.509 certificate for a SAML Assertion.



*Figure 20 IFS Activity diagram*

Users authenticate with grid services using a X.509 credential. Such “credential” consists of a private key and corresponding X.509 certificate, which is signed by Dorian’s certificate authority. Dorian facilitates the creation of X.509 credentials for its users. To create credential the user supplies a lifetime (generally short term) and the SAML assertion provided by their identity provider to the Dorian client. The Dorian client generates a new public/private key pair and sends the lifetime, public key, and SAML assertion to the Dorian Service. The Dorian Service validates the SAML assertion and creates an X.509 certificate based on the public key provide by the user. The created certificate is signed by the Dorian certificate authority and returned to the user. The certificate and locally generated private key can then be used as a credential to authenticate with secure services. It is important to note that throughout this process no sensitive information, i.e. private keys are passed over the network.

#### Interfaces

IFS (Identity Federation Service) and IdP (Identity Provider) services exposes their functionality through Dorian web service interface. Following class diagram shows Dorian web service relation with its sub components. IdentityFederationManager handles all the identity federation and management aspects for Dorian including the management of grid user accounts and Trusted IdPs. IdentityProvider provides the functionality for registering, authenticating, and managing users using UserManager. CertificateAuthority is responsible to store user certificates and issue proxy certificates.



*Figure 21 Dorian Service class diagram*

#### Metadata

Dorian publishes metadata about the identity providers it trusts to allow client interfaces to access. This also allows client interfaces to automatically include any new trusted identity providers to its list. Dorian also publishes metadata about AuthenticationProfiles of each trusted identity providers. This provides client interfaces to present valid authentication mechanism to end user for a selected trusted identity provider.

Trusted Identity Provider metadata includes following information.

* + Display Name
  + Authentication Service URL
  + Authentication Service Identity

Following is the schema and metadata of Trusted Identity Provider.

QName: TrustedIdentityProvider

Namespace: http://cagrid.nci.nih.gov/1/dorian-ifs

Schema type: TrustedIdentityProvider

<xsd:element name=*"TrustedIdentityProvider"* type=*"federation:TrustedIdentityProvider"* />

<xsd:complexType name=*"TrustedIdentityProvider"*>

<xsd:sequence>

<xsd:element name=*"name"* type=*"xsd:string"* />

<xsd:element name=*"displayName"* type=*"xsd:string"* />

<xsd:element name=*"authenticationServiceURL"* type=*"xsd:string"*

minOccurs=*"0"* />

<xsd:element name=*"authenticationServiceIdentity"* type=*"xsd:string"*

minOccurs=*"0"* />

</xsd:sequence>

</xsd:complexType>

Following is the AuthenticationProfiles schema and metadata.

<xs:complexType name="AuthenticationProfiles">

<xs:sequence>

<xs:element name="profile" minOccurs="1" maxOccurs="unbounded" type="xs:QName"/>

</xs:sequence>

</xs:complexType>

Following are the details on supported profiles by AuthenticationService:

QName: BASIC\_AUTHENTICATION

Namespace: http://gaards.cagrid.org/authentication

Schema type: BasicAuthentication

<xs:complexType name="BasicAuthentication">

<xs:complexContent>

<xs:extension base="gaards:Credential">

<xs:attribute name="userId" use="required" type="xs:string" />

<xs:attribute name="password" use="required" type="xs:string" />

</xs:extension>

</xs:complexContent>

</xs:complexType>

QName: *ONE\_TIME\_PASSWORD*

Namespace: http://gaards.cagrid.org/authentication

Schema type: OneTimePassword

<xs:complexType name="OneTimePassword">

<xs:complexContent>

<xs:extension base="gaards:Credential">

<xs:attribute name="userId" use="required" type="xs:string" />

<xs:attribute name="oneTimePassword" use="required" type="xs:string" />

</xs:extension>

</xs:complexContent>

</xs:complexType>

#### Operations

Following are the operations a Dorian service provides.

createProxy() - **DEPRECATED**

**Description**: Generates a X.509 certificate for user supplied public key and proxy life time after validating SAMLAssertion.

**Input**: org.cagrid.gaards.dorian.SAMLAssertion

**Namespace**: <http://cagrid.nci.nih.gov/1/dorian-common>

**Schema type**: SAMLAssertion

org.cagrid.gaards.dorian.federation.PublicKey

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** PublicKey

org.cagrid.gaards.dorian.federation.ProxyLifetime

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** ProxyLifetime

**Output**: org.cagrid.gaards.dorian.X509Certificate[]

**Namespace**: <http://cagrid.nci.nih.gov/1/dorian-common>

**Schema type**: X509Certificate

**Actors:** User, Service

**Exceptions**: DorianInternalFault

InvalidAssertionFault

InvalidProxyFault

UserPolicyFault

PermissionDeniedFault

**Pre-condition:**

* + - Identity provider is registered with Dorian as trusted.
    - Account approval policy has been setup for new users.
    - User is authenticated with external authentication service.
    - Signed SAML assertion with required user attributes has been created.
    - Public key of trusted identity provider that signed SAML Assertion and life time of requested certificate is provided.

**Post-condition:**

* With “Auto Approval” account policy, a new user is automatically registered and given access to the grid. With “Manual Approval” policy, a new user is automatically registered but not granted access, an administrator is required to grant access. (User status is set to pending when their account is created).
* Existing user account is updated with any changes in user attributes.
* Proxy certificate for requested life time and signed with requestor public key is returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

getCACertificate()

**Description:** Return Dorian CA pubic certificate

**Input:**

**Output:** org.cagrid.gaards.dorian.X509Certificate

**Namespace**: <http://cagrid.nci.nih.gov/1/dorian-common>

**Schema type**: X509Certificate

**Actors**: User, Service

**Exceptions:** DorianInternalFault

**Pre-condition:**

* None

**Post-condition:**

* Dorian CA public certificate is returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

getTrustedIdPs()

Namespace:

**Description:** Returns trusted identity providers registered with Dorian

**Input:**

**Output:** org.cagrid.gaards.dorian.federation.TrustedIdP[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema Type:** TrustedIdP

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.

**Post-condition:**

* Return all identity providers registered with Dorian for identity federation.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

addTrustedIdP()

**Description:** Add a trusted identity provider to Dorian

**Input:** org.cagrid.gaards.dorian.federation.TrustedIdP

**Output:** org.cagrid.gaards.dorian.federation.TrustedIdP

**Actors:** Admin

**Exceptions:** DorianInternalFault

InvalidTrustedIdPFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.
* Implement and operate AuthenticationService to authenticate locally and generate SAML assertions.

**Post-condition:**

* IdentityProvider is added to Dorian as trusted.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

updateTrustedIdP()

**Description:** Update a trusted identity provider exists in Dorian

**Input:** org.cagrid.gaards.dorian.federation.TrustedIdP

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema Type:** TrustedIdP

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

InvalidTrustedIdPFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.
* Implement and operate AuthenticationService to authenticate locally and generate SAML assertions.

**Post-condition:**

* IdentityProvider has been updated with given details.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

removeTrustedIdP()

**Description:** Remove a trusted identity provider exists in Dorian

**Input:** org.cagrid.gaards.dorian.federation.TrustedIdP

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema Type:** TrustedIdP

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

InvalidTrustedIdPFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.
* Implement and operate AuthenticationService to authenticate locally and generate SAML assertions.

**Post-condition:**

* Identity Provider has been removed from Dorian.
* All accounts associated with identity provider have been removed.
* User certificates that were revoked at the time the identity provider was deleted have been added permanently to Dorian's CRL
* Host certificates issued by Dorian associated with the identity provider have been permanently revoked.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

authenticate() - **DEPRECATED**

**Description:** Authenticate a Dorian local user using authentication service. Dorian provides identity provider services for users registered with Dorian directly. This method has been deprecated in version 1.3.

**Input:** gov.nih.nci.cagrid.authentication.bean.Credential

**Namespace:** http://cagrid.nci.nih.gov/1/authentication-service

**Schema type:** Credential

**Output:** gov.nih.nci.cagrid.authentication.bean.SAMLAssertion

**Namespace**: <http://cagrid.nci.nih.gov/1/dorian-common>

**Schema type**: SAMLAssertion

**Actors:** User, Service

**Exceptions:** InvalidCredentialFault

InsufficientAttributeFault

AuthenticationProviderFault

**Pre-condition:**

* User has valid account with local authentication facility.
* Credential confirming AuthenticationProfiles metadata is supplied as input.

**Post-condition:**

* User authentication is done locally.
* SAMLAssertion is returned with attributes containing user information.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

addAdmin()

**Description:** Add a grid user to Dorian administrator group.

**Input:** java.lang.String - gridIdentity

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.

**Post-condition:**

* User has been granted with Administrator privileges.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

removeAdmin()

**Description:** Remove a grid user from Dorian administrator group.

**Input:** java.lang.String - gridIdentity

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.

**Post-condition:**

* User has been revoked with Administrator privileges.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

getAdmins()

**Description:** Find all Dorian Admin users

**Input:**

**Output:** java.lang.String[] – Grid identities

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account with admin privileges.

**Post-condition:**

* Return all Dorian users with Administrator privileges.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

findLocalUsers()

**Description:** Find all local users registered with Dorian IdP for a given matching criteria.

**Input:** org.cagrid.gaards.dorian.idp.LocalUserFilter

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** LocalUserFilter

**Output:** org.cagrid.gaards.dorian.idp.LocalUser[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** LocalUser

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active local user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Return all local users registered with Dorian for the given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

registerWithIdP()

**Description:** Register a user with Dorian Identity Provider. Returns account status message.

**Input:** org.cagrid.gaards.dorian.idp.Application

**Output:** java.lang.String

**Actors:** User

**Exceptions:** DorianInternalFault

InvalidUserPropertyFault

**Pre-condition:**

* Registration application is filled with all required information.

**Post-condition:**

* Local user account has been registered with Dorian.
* Account is set with “Active” or “Pending” status based on account approval policy defined in Dorian configuration.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

updateLocalUser()

**Description:** Update a local user registered with Dorian IdP.

**Input:** org.cagrid.gaards.dorian.idp.LocalUser

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** LocalUser

**Output:**

**Actors:** Admin, User, Service

**Exceptions:** DorianInternalFault

NoSuchUserFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active local user account with Dorian with Administrator privileges.
* Input user account exists in Dorian with Active or suspended status.
* Password is in valid format. It has at least one uppercase, lowercase, numeric and symbol characters. Password length is less than 30 characters. Minimum password length is greater than Dorian configured setting. Password does not have dictionary word.

**Post-condition:**

* User account has been updated with changes.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

removeLocalUser()

**Description:** Remove a local user from Dorian IdP.

**Input:**

**Output:** java.lang.String – UserId

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active local user account with Dorian with Administrator privileges.
* Input user account exists in Dorian with Active status.

**Post-condition:**

* User account has been removed from Dorian IdP.
* All certificates issued for the users have been removed
* All active host certificates issued for the user account have been put in “Compromised” state.
* All host certificates with “pending” state have been set to “rejected” state.
* Remove user from all enrolled groups.
* IdP CRL has been updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

registerLocalUser()

**Description:** Register a user with Dorian Identity Provider. Returns account status message.

**Input:** org.cagrid.gaards.dorian.idp.Application

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** Application

**Output:** java.lang.String

**Actors:** User

**Exceptions:** DorianInternalFault

InvalidUserPropertyFault

**Pre-condition:**

* Registration application is filled with all required information.

**Post-condition:**

* Local user account has been registered with Dorian.
* Account is set with “Active” or “Pending” status based on account approval policy defined in Dorian configuration.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

requestHostCertificate()

**Description:** Request a host certificate to host services. A host credential consist of a X.509 certificate and private key.

**Input:** org.cagrid.gaards.dorian.federation.HostCertificateRequest

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** HostCertificateRequest

**Output:** org.cagrid.gaards.dorian.federation.HostCertificateRecord

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** HostCertificateRecord

**Actors:** User

**Exceptions:** DorianInternalFault

InvalidHostCertificateRequestFault

InvalidHostCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian.
* Valid Host name, caller public key has been provided.
* A host certificate with “Active” or “Pending” state does not exist for the host.
* Provided public key is not in “Compromised” state with Dorian.
* Provided public key size is equal to CA configured size.

**Post-condition:**

* Return host certificate to the caller.
* Host certificate is set to Active status, if auto host approval policy is set to true.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

getOwnedHostCertificates()

**Description:** Returns list of host certificates issued to the requestor.

**Input:**

**Output:** org.cagrid.gaards.dorian.federation.HostCertificateRecord[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** HostCertificateRecord

**Actors:** User

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian.

**Post-condition:**

* Return all host certificates issued to the caller.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

approveHostCertificate()

**Description:** Approve a host certificate request

**Input:** java.math.BigInteger - recordId

**Output:** org.cagrid.gaards.dorian.federation.HostCertificateRecord

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** HostCertificateRecord

**Actors:** Admin

**Exceptions:** DorianInternalFault

InvalidHostCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.
* Host certificate is in pending state.

**Post-condition:**

* Host certificate has been approved.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

findHostCertificates()

**Description:** List all host certificated issued and requested

**Input:**

**Output:** org.cagrid.gaards.dorian.federation.HostCertificateRecord[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** HostCertificateRecord

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Return all host certificates matched with given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

updateHostCertificateRecord()

**Description:** Update a host record

**Input:** org.cagrid.gaards.dorian.federation.HostCertificateUpdate

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** HostCertificateUpdate

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

InvalidHostCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.
* Host certificate exists with Dorian.
* Host certificate is not in “Compromised”, “Rejected” or “Pending” state.

**Post-condition:**

* Host certificate has been updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

renewHostCertificate()

**Description:** Renew a host certificate

**Input:** java.math.BigInteger recordId

**Output:**

**Actors:** User, Service

**Exceptions:** DorianInternalFault

InvalidHostCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.
* Host certificate is active.

**Post-condition:**

* Host certificate has been renewed for Dorian configured life time.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

changeIdPUserPassword

**Description:** Change Dorian local user password.

**Input:** org.cagrid.gaards.dorian.idp.BasicAuthCredential

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** BasicAuthCredential

java.lang.String - newPassword

**Output:**

**Actors:** Admin, User, Service

**Exceptions:** InvalidUserPropertyFault

DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider.
* Authentication credential has been provided to change password.
* New password should not be same as old password.

**Post-condition:**

* New password has been set for the user account.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

authenticateUser()

**Description:** Authenticate a Dorian local user using authentication service. Dorian provides identity provider services for users registered with Dorian directly.

**Input:** org.cagrid.gaards.authentication.Credential

**Namespace:** http://cagrid.nci.nih.gov/1/authentication-service

**Schema type:** Credential

**Output:** gov.nih.nci.cagrid.opensaml.SAMLAssertion

**Namespace**: <http://cagrid.nci.nih.gov/1/dorian-common>

**Schema type**: SAMLAssertion

**Actors:** User, Service

**Exceptions:** CredentialNotSupportedFault

InvalidCredentialFault

InsufficientAttributeFault

AuthenticationProviderFault

**Pre-condition:**

* User has valid account with local authentication facility.
* Credential confirming AuthenticationProfiles metadata has been supplied as input.

**Post-condition:**

* User authentication is done locally.
* SAMLAssertion is returned with attributes containing user information.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

changeLocalUserPassword()

**Description:** Change Dorian local user password.

**Input:** org.cagrid.gaards.authentication.BasicAuthentication

**Namespace:** http://gaards.cagrid.org/authentication

**Schema type:** BasicAuthentication

java.lang.String - newPassword

**Output:**

**Actors:** Admin,User, Service

**Exceptions:** InvalidUserPropertyFault

DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active local user account with Dorian Identity Provider.
* Authentication credential has been provided to change password.
* New password should not be same as old password.

**Post-condition:**

* New password has been set for the user account.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

doesLocalUserExist()

**Description**: Verify if a local user exist in Dorian.

**Input:** java.lang.String userId

**Output:** Boolean

**Actors:** Admin

**Exceptions:** DorianInternalFault

**Pre-condition:**

* None.

**Post-condition:**

* Return true if given local userId exists in Dorian.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

findGridUsers()

**Description:** Find all grid users based on given criteria

**Input:** org.cagrid.gaards.dorian.federation.GridUserFilter

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** GridUserFilter

**Output:** org.cagrid.gaards.dorian.federation.GridUser[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** GridUser

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* `Return all local grid user account details.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

updateGridUser()

**Description:** Update a grid user account

**Input:** org.cagrid.gaards.dorian.federation.GridUser

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** GridUser

**Output:**

**Actors:** Admin, User, Service

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian.
* Caller is updating his/her own account.
* First name and last name length is less than 255 characters.
* Email address is in valid format.

**Post-condition:**

* User has been updated with changes.
* IdP CRL has been updated for active user status.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

removeGridUser()

**Description:** Remove a grid user account

**Input:** org.cagrid.gaards.dorian.federation.GridUser

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** GridUser

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.
* Input user account exists in Dorian.

**Post-condition:**

* Grid user account has been removed
* All certificates issued for the users have been removed
* All active host certificates issued for the user account have been put in “Compromised” state.
* All host certificates with “pending” state have been set to “rejected” state.
* Remove user from all enrolled groups.
* Update IdP CRL.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

getGridUserPolicies()

**Description:** Get grid user policies defined

**Input:**

**Output:** org.cagrid.gaards.dorian.federation.GridUserPolicy[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** GridUser

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* `Return user account policies.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

updateUserCertificate

**Description:**

**Input:** org.cagrid.gaards.dorian.federation.UserCertificateUpdate

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** UserCertificateUpdate

**Output:**

**Exceptions:** DorianInternalFault

InvalidUserCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Update user certificate with given details.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

requestUserCertificate()

**Description**: Generates a proxy certificate for user supplied proxy life time after validating SAMLAssertion.

**Input:** gov.nih.nci.cagrid.opensaml.SAMLAssertion

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-common

**Schema type:** SAMLAssertion

org.cagrid.gaards.dorian.federation.PublicKey

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** PublicKey

org.cagrid.gaards.dorian.federation.CertificateLifetime

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** CertificateLifetime

**Output:** org.cagrid.gaards.dorian.X509Certificate

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-common

**Schema type:** X509Certificate

**Exceptions:** DorianInternalFault

InvalidAssertionFault

PermissionDeniedFault

UserPolicyFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Return all local user certificates for the given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

findUserCertificates()

**Description:** Find all proxy certificates issued

**Input:** org.cagrid.gaards.dorian.federation.UserCertificateFilter

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** UserCertificateFilter

**Output:** org.cagrid.gaards.dorian.federation.UserCertificateRecord[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** UserCertificateRecord

**Actors:** Admin

**Exceptions**: DorianInternalFault

InvalidUserCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Return all user certificates for the given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

removeUserCertificate()

**Description:** Remove a proxy certificate issued for a user.

**Input:** java.lang.String serialNumber

**Output:**

**Actors:** Admin

**Exceptions:** DorianInternalFault

InvalidUserCertificateFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Add certificate to blacklist if status is “compromised”.
* Remove user certificate from certificate store.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

performFederationAudit()

**Description:** Query Federation audit logs for a given criteria.

**Input:** org.cagrid.gaards.dorian.federation.FederationAuditFilter

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** FederationAuditFilter

**Output:** org.cagrid.gaards.dorian.federation.FederationAuditRecord[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-ifs

**Schema type:** FederationAuditRecord

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Return identity federation service audit records for the given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

performIdentityProviderAudit()

**Description:** Query IdP audit logs for a given criteria.

**Input:** org.cagrid.gaards.dorian.idp.IdentityProviderAuditFilter

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** IdentityProviderAuditFilter

**Output:** org.cagrid.gaards.dorian.idp.IdentityProviderAuditRecord[]

**Namespace:** http://cagrid.nci.nih.gov/1/dorian-idp

**Schema type:** IdentityProviderAuditRecord

**Actors:** Admin

**Exceptions:** DorianInternalFault

PermissionDeniedFault

**Pre-condition:**

* Caller has active user account with Dorian Identity Provider with Admin privileges.

**Post-condition:**

* Return identity provider audit records for the given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Dorian fault codes.

Following class diagram shows all the operations supported by a DorianService.



*Figure 22 DorianService class diagram*

#### Data types

Dorian service schema and WSDL can be referred at

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[AuthenticationService.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/AuthenticationService.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[DorianConfiguration.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/DorianConfiguration.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[dorian-common.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/dorian-common.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[dorian-conf.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/dorian-conf.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[dorian-idp.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/dorian-idp.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[dorian-ifs.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/dorian-ifs.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[xmldsig-core-schema.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/xmldsig-core-schema.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[AuthenticationService.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/AuthenticationService.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[AuthenticationServiceTypes.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/AuthenticationServiceTypes.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[authentication-core-types.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/authentication-core-types.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/)[authentication-service.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/authentication-service.xsd)

#### Error Handling

Following are the fault codes returned by a Dorian Service on an error processing a request.

DorianInternalFault: This is a generic fault returned by Dorian service.

*<element name="DorianInternalFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidUserPropertyFault: This fault is returned if you try to change a user property with invalid value.

*<element name="InvalidUserPropertyFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

PermissionDeniedFault: This fault is returned if user doesn’t have access permission or invalid credentials or account has been suspended, rejected or not not reviewed.

*<element name="PermissionDeniedFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

NoSuchUserFault: This fault is returned if user does not exist.

*<element name="NoSuchUserFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidAssertionFault: This fault is returned if Assertion is not valid.

*<element name="InvalidAssertionFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidProxyFault

*<element name="InvalidProxyFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

UserPolicyFault: This fault is returned if the request did not meet user policy requirements.

*<element name="UserPolicyFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidTrustedIdPFault: This fault is returned if an operation on query, add, update, delete operation on a trusted IdP is failed.

*<element name="InvalidTrustedIdPFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidUserFault: This fault is returned if an operation on query, add, update, delete operation on a user is failed.

*<element name="InvalidUserFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

AdminAlreadyExists: This fault is returned for duplicate Admin name.

*<element name="AdminAlreadyExists" type="tns:AdminAlreadyExists" />*

*<complexType name="AdminAlreadyExists">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

InvalidHostCertificateRequestFault: This fault is returned if host certificate request is not valid.

*<element name="InvalidHostCertificateRequestFault" type="tns:InvalidHostCertificateRequestFault" />*

*<complexType name="InvalidHostCertificateRequestFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

InvalidHostCertificateFault: This fault is returned if host certificate is not valid.

*<element name="InvalidHostCertificateFault" type="tns:InvalidHostCertificateFault" />*

*<complexType name="InvalidHostCertificateFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

InvalidUserCertificateFault: This fault is returned if user certificate is not valid.

*<element name="InvalidUserCertificateFault" type="tns:InvalidUserCertificateFault" />*

*<complexType name="InvalidUserCertificateFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

#### Data Model

Dorian Service uses following tables to store data to service client requests.

**CERTIFICATE\_AUTHORITY**

Dorian maintains its own certificate authority, which it uses to issues and manage user credentials, host credentials, and SAML asserting credentials. Depending on deployment requirements, Dorian can either be configured to use an existing certificate authority or it can generate a new certificate authority. CERTIFICATE\_AUTHORITY table stores information related to certificate authorities registered with Dorian. This table is not used if Hardware Security Module (HSM) is used to store certificates. Please see section [3.3.1.9](#_Dorian_Certificate_Authority) for details.

**CERTIFICATE\_BLACKLIST** table stores information related to black listed certificates.

**IDP\_USERS** table stores information related to users registered with Dorian directly.

**IDP\_PASSWORD\_SECU**RITY table stores information related to login attempts of users using Dorian as IdP (registered with Dorian directly).

**IDP\_ASSERTER** table stores information related to assertions made by Dorian IdP.



*Figure 23 Dorian Identity Provider data model*

**IFS\_USERS** table stores information related to federated users accessing Dorian from a trusted identity provider.

**USER\_CERTIFICATES** table stores information related to X.509 certificates generated by Dorian.

**HOST\_CREDENTIALS**

In order to run secure services securely, the container hosting the services must run with a host credential. A host credential consists of a X.509 certificate and private key. Dorian provides a means for users with a grid user account to request a host credential for their services. HOST\_CREDENTIALS table stores information related to certificates issued to a host. Please see section [3.3.1.10](#_Host_Certificates) for more details.

**TRUST\_MANAGER** maintains list of trusted identity providers.

**TRUST\_MANAGER\_AUTH\_METHODS** maintains the list of acceptable authentication methods for each trusted identity provider.

**CERTIFICATE\_AUTHORITY**

Dorian maintains its own certificate authority, which it uses to issues and manage user credentials, host credentials, and SAML asserting credentials. Depending on deployment requirements, Dorian can either be configured to use an existing certificate authority or it can generate a new certificate authority. CERTIFICATE\_AUTHORITY table stores information related to certificate authorities registered with Dorian. This table is not used if Hardware Security Module (HSM) is used to store certificates. Please see section [3.3.1.9](#_Dorian_Certificate_Authority) for details.

**CERTIFICATE\_BLACKLIST** table stores information related to black listed certificates.



*Figure 24 IFS data model*

#### Identity Federation Service (IFS)

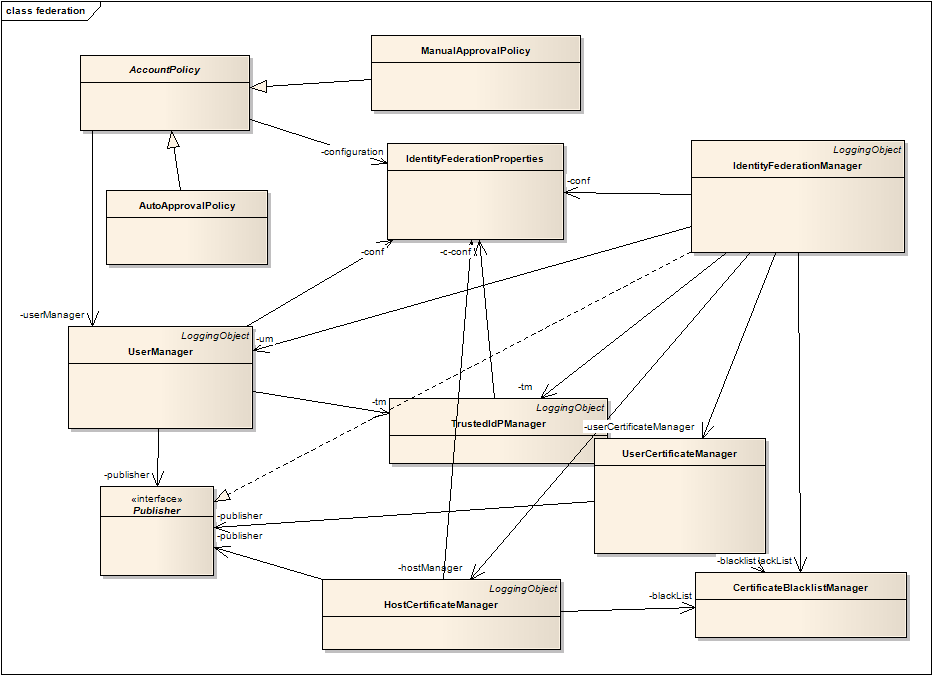
The Identity Federation Service (IFS) component of Dorian facilitates the federation of the local user accounts from multiple institutions to the caGrid. Architecturally (Figure 17) the IFS consists of five components: the Trusted IdP Manager, the Certificate Authority, the Grid User Manager, the Group Manager, and the Host Certificate Manager. The Trusted IdP Manager component manages the list of Institutional IdPs from which Dorian will accept SAML assertions as a mechanism of authentication. The Certificate Authority component manages and issues user and host/service credentials. The Grid User Manager component manages the account information for each grid user. This information includes the user's identity provider, institutional user id, email address, and the user's account status. The Host Certificate Manager manages host credentials requests and issued host credentials. The Group Manager component manages internal groups for Dorian, an example of one such group is the Dorian administrators group. The Dorian administrators group consists of all the users that may administer Dorian.

The IFS functionality can be divided into two sub interfaces, the IFS user interface, and the IFS administrative interface. The IFS user interface allows local users to create grid credentials. The IFS administrative interface provides operations that allow administrators to manage Trusted IdPs and grid user accounts. Note that invoking an administrative operation requires a grid proxy of an administrative user.

Following are the user attributes used by federation service to maintain user accounts.

* **Grid Identity** - globally unique identifier for the user, this identity is used by services and other parties for identifying the user.
* **Identity Provider** - The identity provider in the federation in which the user belongs to.
* ***Local User Id***- The users unique identity within the identity provider they belong to.
* ***First Name*** - The user’s first name
* **Last Name** - The user’s last name
* **Email Address** - The user’s email address
* **Account Status** – The status of a user's account. (Active, Pending, Suspended, Rejected)

Following class diagrams shows different classes, hierarchies and relationships in Identity Federation Service.



*Figure 25 IFS Class diagram*

Following diagrams shows different attributes and operations in TrustedIdPManager service. TrustedIdPManager is responsible maintaining trusted identity providers registered with Dorian. The IdP Id is a unique id assigned by Dorian to identify the IdP. Each Trusted IdP is associated with a set of configurable User Policies that are applied to each user when they authenticate. These policies designate how Dorian should handle users from a specified Trusted IdP. Automatic and manual approval policies are supported out of the box. The User Policy framework is extensible so that administrators can implement local policies.

Each Trusted IdP must also specify its own certificate. When Dorian receives a SAML assertion from a Trusted IdP it verifies that the assertion was signed with the private key that corresponds to the Trusted IdP's certificate. Finally, each Trusted IdP must be configured with a list of acceptable authentication methods. A SAML authentication assertion specifies the method in which the Trusted IdP authenticated the user. In order for the SAML assertion to be accepted by Dorian, the authentication method specified in the assertion must be specified as acceptable in the corresponding Trusted IdP.

*Figure 26 TrustedIdPManager class diagram*

Following class diagram shows attributes and operations of UserManager.

UserManager is responsible managing caGrid users registered with external identity providers. When a user authenticated and asserted by external authentication service try to get caGrid short term credentials, Dorian uses UserManager to verify, create, update user. UserManager also supports admin operation to manage individual grid user accounts.



*Figure 27 UserManager class diagram*

Following diagram shows attributes and operation of UserCertificateManager. Dorian uses UserCertificateManager to manage short term certificates issued to a user.



*Figure 28 UserCertificateManager class diagram*

HostCertificateManager is responsible for managing host certificates. Some of the operations allow user to request and manage host certificates. Other operations are restricted to administrators allowing them to approve, revoke, renew, and manage all host certificates issued by Dorian.



*Figure 29 HostCertificateManager class diagram*

#### Dorian Identity Provider

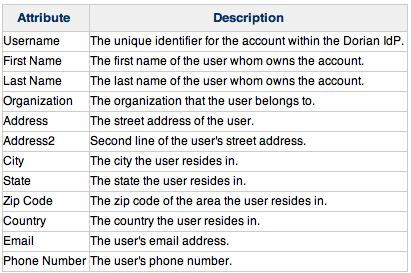
For organizations without an Identity Management system, Dorian provides its own Identity Provider for the management of user accounts. The DorianIdP provides a method for prospective users to register for an account and obtain grid credentials through Dorian. Architecturally, (Figure 17) the Dorian IdP consists of three components: the DorianIdP User Manager, the SAML Asserter, and a certificate authority. The Dorian IdP User Manager provisions and manages local Dorian IdP accounts. The SAML Asserter creates and signs SAML assertions for Dorian IdP users, which will be later consumed by the Dorian IFS in creating grid credentials. The Certificate Authority is used by the SAML Asserter for signing assertions.

To support authentication services to the users registered with Dorian Identity Provider, it implements AuthenticationService.

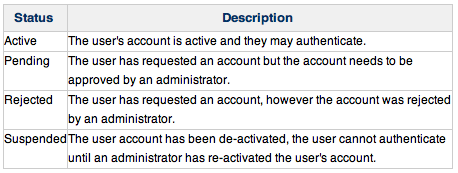
Dorian Identity Provider stores following information to maintain local user accounts.

* **User Information** – Attributes describing the identity of the user.
* **Account Information**– The Account Information tab contains user's Status and Role attributes.
* **Password**– The user’s password and information related to its security.
* **Auditing** – Auditing information for the account

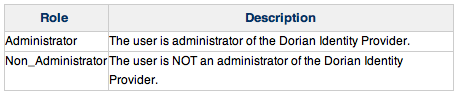
**User Information –** Attributes describing the identity of the user.

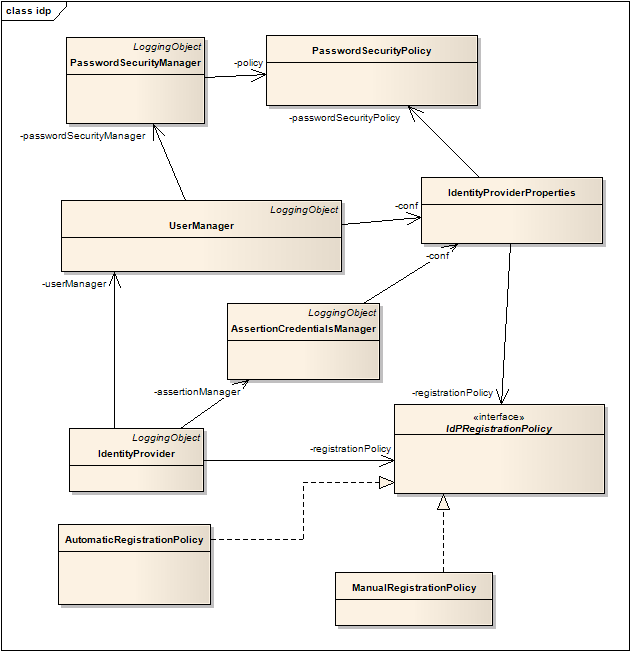


Account Information – Information describing the role and current status of a users account.



**User Role –** Whether or not the user can administer the Dorian Identity Provider



Following diagram shows Dorian Identity Provider classes and their dependencies.

*Figure 30 Dorian Identity Provider class diagram*

Following class diagram shows attributes and operations of IdentityProvider. Dorian uses IdentityProvider to manage users registered with Dorian directly.



*Figure 31 IdentityProvider class diagram*

Following class diagram shows attributes and operations of AssertionCredentialsManager component that creates and signs SAML assertions for Dorian IdP users.



*Figure 32 AssertionCredentialsManager class diagram*

PasswordSecurityManager is responsible for managing passwords of Dorian registered users based on enforced policies. When a user registers with Dorian IdP, password will be verified against defined security policies and will be encrypted with SHA-512 algorithm using cryptographically generated strong pseudo-random number. This information is stored in idp\_password\_security table.

Following are the password security policies defined:

**Password Strength Requirements**

* + - 10 – 20 Characters (Configurable)
    - At Least 1 Capital Letter
    - At Least 1 Number
    - At Least 1 non-alphanumeric symbol
    - No dictionary words (forwards or backwards)

**Password Security (Configurable)**

5 Invalid consecutive logins, results in a 4 hour suspension of account 500 total invalid logins results in permanent suspension of account until password is reset by and administrator

**Password Digest Algorithm**: SHA-512



UserManager is responsible for managing local user accounts registered with Dorian identity provider. Following class diagram shows all operation and attributes.



#### Dorian Certificate Authority

Dorian maintains its own certificate authority, which it uses to issues and manage user credentials, host credentials, and SAML asserting credentials in X.509 format. Depending on deployment requirements, Dorian can either be configured to use an existing certificate authority, or it can generate a new certificate authority. Existence of AutoCreateelement in Dorian configuration would let Dorian create a new certificate authority using configuration properties. Following is an example of Dorian CA configuration for a new certificate authority.

<DorianCAConfiguration>

<CertificateAuthorityType>DBCA</CertificateAuthorityType>

<CertificateAuthorityPassword>admin</CertificateAuthorityPassword>

<CertificatePolicyOID>5.5.5.5.5.5.5555.555.5.5</CertificatePolicyOID>

<UserKeySize>1024</UserKeySize>

<AutoCreate>

<CASubject>C=US,O=abc,OU=xyz,OU=caGrid,OU=Users,CN=caGrid Dorian CA</CASubject>

<CAKeySize>2048</CAKeySize>

<lifetime years="10" months="0" days="0" hours="0" minutes="0" seconds="0"/>

</AutoCreate>

<AutoRenewal years="10" months="0" days="0" hours="0" minutes="0" seconds="0"/>

</DorianCAConfiguration>

As mentioned above, Dorian can be configured to use existing certificate authority. At first time, existing certificate authority should imported into Dorian trust store. Following is an example configuration to use existing certificate authority.

<DorianCAConfiguration>

<CertificateAuthorityType>DBCA</CertificateAuthorityType>

<CertificateAuthorityPassword>admin</CertificateAuthorityPassword>

<CertificatePolicyOID>5.5.5.5.5.5.5555.555.5.5</CertificatePolicyOID>

<UserKeySize>1024</UserKeySize>

<AutoRenewal years="10" months="0" days="0" hours="0" minutes="0" seconds="0"/>

</DorianCAConfiguration>

<CertificateAuthorityPassword> element value is used to access CA private key. The CertificatePolicyOID is an optional element that specifies the OID of the CA's policy. The <UserKeySize> element specifies the key size (512, 1024, or 2048) that Dorian should use in creating user credentials. The <AutoRenewal> element instructs Dorian what to do when a certificate authority expires. If the <AutoRenewal> element is specified Dorian will renew the certificate authority when it expires for the lifetime specified. If the <AutoRenewal> element is not specified, Dorian will allow the certificate authority to expire, invalidating any grid user accounts that it manages.

Dorian provides multiple approaches for creating and storing private keys, these approaches include the following:

* **DBCA** - User and CA private keys are created in memory and stored along with their corresponding certificates in Dorian's MySQL database. Following is the structure of database table used to store the information.

*CREDENTIALS\_TABLE*{

ALIAS VARCHAR(255) NOT NULL PRIMARY KEY,

SERIAL\_NUMBER BIGINT NOT NULL,

CERTIFICATE TEXT NOT NULL,

PRIVATE\_KEY TEXT NOT NULL

}

* **Eracom** - The CA private key is generated and stored in a [SafeNet Protect Server Gold](http://www.safenet-inc.com/Products/Data_Protection/Hardware_Security_Modules/General_Purpose_HSMs,_Embedded/ProtectServer_HSMs.aspx) HSM. User private keys are generated in memory and wrapped by a 256 bit RSA key stored and maintained in a [SafeNet Protect Server Gold](http://www.safenet-inc.com/Products/Data_Protection/Hardware_Security_Modules/General_Purpose_HSMs,_Embedded/ProtectServer_HSMs.aspx) HSM. The wrapped user private keys are stored in a database.

#### 

*Figure 33 Dorian CA class diagram*

The Dorian Certificate Authority uses Bouncy Castle Crypto APIs to generate

Version 3 X.509 certificates. Using the HSM with Dorian requires that the Java Cryptography Extension (JCE) unlimited strength jurisdiction policy files be installed into your JVM. By default Dorian is configured to use the **DBCA** approach.

Following class diagrams shows attributes and operations of Dorian CertificateAuthority.

CertificateAuthority is responsible in creating new certificate authority, use imported certificate authority to create, manage user credentials, host credentials and SAML asserting credentials. It is extended to support DBCA and Eracom storage types. Access to CertificateAuthority operations are restricted to user with CA credentials.



*Figure 34 CertificateAuthority class diagram*

DBCertificateAuthority extends CertificateAuthority to support certificate storage in a given configured database. It uses “SHA1WithRSAEncryption” algorithm to sign generated certificates.



EracomCertificateAuthority uses Eracom provided keystore to store and retrieve user, CA private keys. EracomCertificateAuthority uses “SHA1WithRSA” algorithm to sign generated certificates.



#### Host Certificates

The Dorian grid service interface provides several operations for managing host certificates. Some of the operations allow users to request and manage host certificates. Other operations are restricted to administrators, allowing them to approve, revoke, renew, and manage all host certificates issued by Dorian. In order to run secure services securely, the container hosting the services must run with a host credential. A host credential consists of a X.509 certificate and private key. Dorian provides a means for users with a grid user account to request a host credential for their services. Dorian maintains a host credential record for each host credential requested and issued. Dorian assigns each host credential record one of the following statuses:

* **Pending -**Host credentials that have been requested but not yet issued because they require approval of an administrator.
* **Rejected -**Host credentials that have been requested but were not issued because the request was rejected by and administrator.
* **Active -**Host credentials that have been issued.
* **Suspended -**Host credentials that were issued but have been temporarily revoked.
* **Compromised -**Host credentials that were issued and are permanently revoked.

Host credentials issued by Dorian are bound to a grid user account managed by Dorian. This binding makes users responsible for any host credentials bound to their account. If a user account is suspended, any host credential bound to their account will be revoked and listed in the Dorian CA CRL. If a user account is removed, any host credential bound to that account will be marked as Compromised. Each host certificate record is an assigned an owner, or the user who the credential is bound to. If a user’s account is Suspended, all of the user’s host certificates are revoked. Dorian will issue one host credential per host name. Existing certificate must be revoked before another one is issued on host name. Dorian can be configured to automatically approve all host certificate requests or require sign of by administrators.

#### Dorian Client API

Dorian provides convenient client APIs to work with its web service. Following class diagram shows different classes and their relationships.



*Figure 35 Dorian Client API class diagram*

GridAdministrationClient class provides client APIs to maintain Trusted Identity Providers and their user policies. It provides APIs to query, add, remove and update grid user accounts and administrator accounts. It also provides APIs to maintain host certificates, user certificates, CA certificates and auditing.

GridUserClient class provides client APIs to request short term grid credential from Dorian and to request host certificates. It provides API to get all trusted identity providers registered with Dorian.

It also provides API to query for owned host certificates and Dorian CA certificate.

LocalUserClient class provides client APIs to register a local user with Dorian CA and to change registered user password. It provides API to get supported authentication profiles and authenticate with Dorian IdP.

LocalAdministrationClient class provides client APIs to manage locally registered users with Dorian IdP. It also provides API to perform audit.

### Dorian Administration

GAARDS presents rich set of user interface for Dorian security administration. The administrative features include Account Management, Managing Trusted Identity Providers, and the ability to grant administrative access to users. Only users that have been granted administrative access to Dorian will be able to access the administrative features of Dorian. When Dorian is started for the first time, the *Dorian* user or the default user will have administrative access. The *Dorian* user can be used to assign administrative privileges to other users. It is important to note that being granted administrative access does allow the administration of the Local Dorian Identity Provider. The local Dorian Identity Provider provides a separate mechanism for assigning administrative rights.

#### Trusted Identity Providers

The Institutional Identity Providers (IdPs) that Dorian is configured to trust are referred to as Trusted Identity Providers (Trusted IdPs). Dorian only creates credentials for users whose identity assertions come from a Trusted IdP. The set of Trusted IdPs can be managed by Dorian IFS administrators through its grid service interface. The Dorian grid service interface provides functionality for adding, modifying, and removing Trusted IdPs. The Trusted IdP information consists of the following: IdP Id, IdP Name, IdP Status, User Policy, Certificate, and acceptable authentication methods. The IdP Id is a unique id assigned by Dorian to identify the IdP. The IdP name is assigned by an administrator and provides human readable name to easily identify an IdP. The IdP Status specifies the current status of the IdP: Active or Suspended. Users associated with a "suspended" IdP will be refused access to Dorian. Each Trusted IdP is associated with a set of configurable User Policies that are applied to each user when they authenticate. These policies designate how Dorian should handle users from a specified Trusted IdP. As an example, a policy might dictate what to do when a new user tries to create grid credentials for the first time. An automatic approval policy would automatically register the user with Dorian and create a grid account for the user. A manual approval policy would automatically register the user but not enable the caGrid account until an administrator manually approves it. User policies can also be used to dictate what to do when a user's grid credentials expire. For example, an automatic renewal policy would enable automatic creation of a new set of credentials using the Dorian certificate authority, where as a manual renewal policy would require and administrator to do so. The User Policy framework is extensible; administrators can implement local policies.  
Each Trusted IdP must also specify its own certificate. When Dorian receives a SAML assertion from a Trusted IdP it verifies that the assertion was signed with the private key that corresponds to the Trusted IdP's certificate. Finally, each Trusted IdP must be configured with a list of acceptable authentication methods. A SAML authentication assertion specifies the method in which the Trusted IdP authenticated the user. In order for the SAML assertion to be accepted by Dorian, the authentication method specified in the assertion must be specified as acceptable in the corresponding Trusted IdP.

#### Grid User Management

Dorian provides a mechanism for its administrators to manage individual grid user accounts. When a user first attempts to create a proxy using Dorian, a grid user account is created for them. The account includes user information, user status,, and a set of grid credentials including the associated grid identity. The user information includes the user's local user id at their institution, the id of the Trusted IdP the user is associated with, their first name, their last name, and their email address. The user's status corresponds to the user's current status: Active, Suspended, Pending, or Expired. Only users with an "Active" status may access Dorian. A user's grid credentials consist of a certificate and private key that are used by Dorian to issue grid proxy certificates. A user's grid identity is compromised of the Certificate Authority's Subject DN (Distinguished Name), the IdP Id, and the user's id at his institution. When a user's grid account is created the initial status of the account is "Pending". As mentioned earlier, if the TrustedIdP has an Auto Approval User Policy in place, the status will automatically be changed to "Active", giving the user instant access to Dorian. Administrators can update a user's status and can renew a user credentials.

#### Dorian IdP Users

The DorianIdP provides administrators a mechanism of administrating IdP users. Invoking any of the Dorian IdP administrative operations requires a grid proxy of a Dorian IdP administrator Administering Dorian IdP user accounts includes changing of contact info, changing of passwords, and creation and revocation of accounts.

Detailed description on Dorian Administration feature can be referred at

<http://cagrid.org/display/dorian/Home>

### Dorian Configuration

Dorian leverages the Spring Frameworkfor configuration.  The use of the Spring Framework provides provides Dorian alot of flexibility in being able to replace components with alternative implementations.  The [Dorian configuration file](http://cagrid.org/display/dorian13/Dorian+Configuration+File) is contained in the file DORIAN\_HOME/etc/dorian-configuration.xml. This file contains the Spring Beans for the configurable components of Dorian.  In most cases users will not need to make changes to the Dorian Configuration file, we will highlight the more common changes in the section [Configuration Beans](http://cagrid.org/display/dorian13/Dorian+Configuration#DorianConfiguration-ConfigurationBeans). The second configuration file Dorian uses is the [Dorian properties file](http://cagrid.org/display/dorian13/Dorian+Properties+File), which is contained in the file DORIAN\_HOME/etc/dorian.properties.  The Dorian properties file contains commonly edited properties for configuring Dorian.

Following properties can be set in dorian.properties to customize the configuration.

* Database
* identityProviderProperties
* identityFederationProperties
* certificateAuthority

DorianConfiguration singleton loads and provides configuration data to the application components.



*Figure 36 Dorianconfiguration class diagram*

Dorian is configured through a single configuration file which is located at DORIAN\_LOCATION/etc/dorian-conf.xml. For simple deployments only the following configuration elements need to be modified.

1) Database Configuration

2) CA Subject Name

**Database Configuration**

Dorian uses a MySQL database to persist account information. Dorian must be modified such that it will interact with your MySQL database. To modify the database configuration to interact with your database, set the values of the host, port, username, and password elements.

**Certificate Authority Subject Name**

Dorian manages an internal certificate authority for signing user and host certificates. The certificate authority is created the first time the service is started. It is important that the subject of the CA certificate is unique and meaningful to your deployment. To set the subject of the certificate authority for your deployment, edit the CASubject element. The default value of C=US,O=abc,OU=xyz,OU=caGrid,CN=caGrid Dorian CA is provided as an example.

Please refer to following link for full Dorian configuration details.

Full details on Dorian configuration can be referred at

<http://cagrid.org/display/dorian13/Dorian+Configuration>

<https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/>[dorian-configuration.xml](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/etc/dorian-configuration.xml)

<https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/schema/Dorian/>[dorian.properties](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_final/cagrid-1-0/caGrid/projects/dorian/etc/dorian.properties)

### SAML Assertion

Dorian uses SAML 1.1 assertions as the enabling mechanism for federating users from local institutions to the caGrid. Under Dorian users authenticate with their local Institution’s IdP. Upon successful authentication, the local Institution IdP will issue and sign a SAML Assertion, which can then be used to authenticate with Dorian and obtain grid credentials.

When Dorian receives a SAML Assertion from a user it verifies that the SAML Assertion was signed by a trusted IdP. In order for the verification process to be successful, it will make sure that the signer of the assertion is a registered Trusted IdP. This is accomplished by ensuring that the SAML Assertion is assigned with the private key that corresponds to the certificate that Dorian has registered with the trusted IdP.

The Dorian SAML Assertion consists of three main parts, the conditions, an authentication statement and an attribute statement. The conditions specify the timeframe that the assertion is valid for. The conditions element specifies a *“NotBefore”* and *“NotOnOrAfter”* attributes. The *“NotBefore”* attribute specifies the date and time for the beginning of the time frame that the assertion for, the *“NotOnOrAfter”* attribute specifies the ending date and time. Dorian does not place restrictions on Identity Providers as to how long assertions should be valid for however Dorian will only except assertions that fall within the specified time frame. It is recommended that the time frame be minimized (with an acceptable buffer) to the amount of time needed for the user to receive the SAML assertions from the IdP and send it on to Dorian. The Authentication Statement specifies the identity of the user and how they locally authenticated. The Attribute Statement consists of several user attributes describing the user. The attributes are used as information by Dorian administrators in assisting them with user account management.

#### Authentication Statement

The Authentication Statement consists of two relevant parts the Authentication Method and the Subject.

*Figure 37 Authentication statement*

<AuthenticationStatement AuthenticationInstant="2006-08-14T13:06:00.401Z" AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">

<Subject>

<NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter">JohnDoe</NameIdentifier>

<SubjectConfirmation>

<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>

</SubjectConfirmation>

</Subject>

</AuthenticationStatement>

**Authentication Method**

The AuthenticationMethod, shown in bold text in below figure, specifies the method in which the user used to authenticate to the IdP. The following is a list of acceptable authentication methods as define by SAML (see SAML 1.1 specification for more details):

1. urn:oasis:names:tc:SAML:1.0:am:password
2. urn:ietf:rfc:1510
3. urn:ietf:rfc:2945
4. urn:oasis:names:tc:SAML:1.0:am:HardwareToken
5. urn:ietf:rfc:2246
6. urn:oasis:names:tc:SAML:1.0:am:X509-PKI
7. urn:oasis:names:tc:SAML:1.0:am:PGP
8. urn:oasis:names:tc:SAML:1.0:am:SPKI
9. urn:oasis:names:tc:SAML:1.0:am:XKMS
10. urn:ietf:rfc:3075
11. urn:oasis:names:tc:SAML:1.0:am:unspecified

<AuthenticationStatement AuthenticationInstant="2006-05-11T16:21:24.575Z"

**AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"**>

<Subject>

<NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter">JohnDoe</NameIdentifier>

<SubjectConfirmation>

<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>

</SubjectConfirmation>

</Subject>

</AuthenticationStatement>

*Figure 38 Authentication method*

**Subject**

Dorian requires the subject element in both the Authentication and Attribute statements. Within the subject element Dorian requires a NameIdentifier and SubjectConfirmation. Dorian does not use the NameIdentifier element, therefore it does not place any restrictions on the format or value of the NameIdentifier element. Dorian does require the NameIdentifier specified in the AuthenticationStatement to “strongly match” the NameIdentifier element specified in the AttributeStatement. The SubjectConfirmation should contain the ConfirmationMethod for a bearer assertion.

<AuthenticationStatement AuthenticationInstant="2006-05-11T16:21:24.575Z"

AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">

**<Subject>**

**<NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter">JohnDoe</NameIdentifier>**

**<SubjectConfirmation>**

**<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>**

**</SubjectConfirmation>**

**</Subject>**

</AuthenticationStatement>

*Figure 39 Subject*

#### Attribute Statement

The Attribute Statement consists of a set of attributes describing the user and a subject identifying the user. The subject is required to strongly match the subject in the Authentication Statement (Section 3.1.2). Dorian requires four attributes: local user id, first name, last name, and email address. In order to facilitate interoperability with many differing IdPs, Dorian is flexible in the attributes it can accept from IdPs. Dorian abstractly defines the need for the attribute localUserId, first name, last name, and email address. Dorian requires the localUserId attribute because it requires that IdP have a way of uniquely identifying it users. Dorian does not care of what an IdP calls its localUserId or the format of it. Dorian can be configured to use an individual IdPs representation for a given attribute. For example take the localUserId attribute, which specifies a User's Identity. When registering an IdP we can configure Dorian to look for the attribute that IdP uses for a User's Identity. For example a Shibboleth based IdP would use eduPersonPrincipalName, where as another IdP may use UID. Since Dorian is flexible with regards to attributes, this specification is also flexible and will abstractly define the attributes required. This specification will also specify format constraints on the attribute values where necessary.

**Local User Id**

This specification requires all Identity Providers to be able to uniquely identify each of its users that it asserts. The Local User Id attribute contains the value of this unique identity. Dorian uses the Local User Id attribute to uniquely identify a user within an IdP. The Local User Id Attribute is required to be specified every time an assertion is made to Dorian. The value of the Local User Id must always be the same each time an assertion is made may never change. The format of the local user id attribute is any string no more than 255 characters.

**First Name**

This specification requires all Identity Providers to be able to provide the first name of the user that the IdP is asserting. The format of the first name attribute’s value, is any string no more than 255 characters.

**Last Name**

This specification requires all Identity Providers to be able to provide the last name of the user that the IdP is asserting. The format of the last name attribute’s value, is any string no more than 255 characters.

**Email**

This specification requires all Identity Providers to be able to provide the email address of the user that the IdP is asserting. The format of the email attribute’s value should conform to RFC 2822.

The length of the value can be no more than 255 characters.

<AttributeStatement>

<Subject>

<NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter">JohnDoe</NameIdentifier>

<SubjectConfirmation>

<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>

</SubjectConfirmation>

</Subject>

**<Attribute AttributeName=”localUserId” AttributeNamespace="http://cabig.nci.nih.org/dorian">**

**<AttributeValue>jdoe</AttributeValue>**

**</Attribute>**

**<Attribute AttributeName="urn:mace:dir:attribute-def:givenName” AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">**

**<AttributeValue>John</AttributeValue>**

**</Attribute>**

**<Attribute AttributeName="urn:mace:dir:attribute-def:sn " AttributeNamespace=” urn:mace:shibboleth:1.0:attributeNamespace:uri ">**

**<AttributeValue>Doe</AttributeValue>**

**</Attribute>**

**<Attribute AttributeName="urn:mace:dir:attribute-def:mail" AttributeNamespace=" urn:mace:shibboleth:1.0:attributeNamespace:uri ">**

**<AttributeValue>jdoe@osu.edu</AttributeValue>**

**</Attribute>**

</AttributeStatement>

*Figure 40 Attribute statement*

#### **Sample Dorian SAML Assertion**

<?xml version="1.0" encoding="UTF-8"?>

<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" AssertionID="\_71fd1edd724149f11800150e0b2bc2e7" IssueInstant="2006-08-14T13:06:00.422Z" Issuer="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter" MajorVersion="1" MinorVersion="1">

<Conditions NotBefore="2006-08-14T13:06:00.201Z" NotOnOrAfter="2006-08-14T13:08:00.201Z" />

<AuthenticationStatement AuthenticationInstant="2006-08-14T13:06:00.401Z" AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">

<Subject>

<NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter">JohnDoe</NameIdentifier>

<SubjectConfirmation>

<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>

</SubjectConfirmation>

</Subject>

</AuthenticationStatement>

<AttributeStatement>

<Subject>

<NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="O=OSU,OU=BMI,OU=caGrid,OU=Dorian,OU=localhost,CN=Dorian IdP Authentication Asserter">dorian</NameIdentifier>

<SubjectConfirmation>

<ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>

</SubjectConfirmation>

</Subject>

<Attribute AttributeName="localUserId" AttributeNamespace="http://cabig.nci.nih.org/dorian">

<AttributeValue>dorian</AttributeValue>

</Attribute>

<Attribute AttributeName="urn:mace:dir:attribute-def:givenName" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">

<AttributeValue>Mr.</AttributeValue>

</Attribute>

<Attribute AttributeName="urn:mace:dir:attribute-def:sn" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">

<AttributeValue>Administrator</AttributeValue>

</Attribute>

<Attribute AttributeName="urn:mace:dir:attribute-def:mail" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">

<AttributeValue>dorian@dorian.org</AttributeValue>

</Attribute>

</AttributeStatement>

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

<ds:SignedInfo>

<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />

<ds:Reference URI="#\_71fd1edd724149f11800150e0b2bc2e7">

<ds:Transforms>

<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />

<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">

<ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="code ds kind rw saml samlp typens #default xsd xsi" />

</ds:Transform>

</ds:Transforms>

<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

<ds:DigestValue>GQ9icA9yZcSGGCBxA2pnTP77OJg=</ds:DigestValue>

</ds:Reference>

</ds:SignedInfo>

<ds:SignatureValue>PsBUYBxzGQTcJM019AuAhuYHxEdT21ixpwkuh22Ogh1vXzgkcFU4ZwhrnC0GNwqJ+RwDzrzcZMPl

k7wEtqkn38RnH2bHbqU/OII84CgIBpxe+XbzWg07luZgHJV06DYjuBcj1ZRemqNbFCeQXGInJzLn

FjH7WntDyfAGmewcCEQ=</ds:SignatureValue>

<ds:KeyInfo>

<ds:X509Data>

<ds:X509Certificate>MIICxzCCAjCgAwIBAgIGAQ0MxEo6MA0GCSqGSIb3DQEBBAUAMG0xDDAKBgNVBAoTA09TVTEMMAoG

A1UECxMDQk1JMQ8wDQYDVQQLEwZjYUdyaWQxDzANBgNVBAsTBkRvcmlhbjESMBAGA1UECxMJbG9j

YWxob3N0MRkwFwYDVQQDExBjYUdyaWQgRG9yaWFuIENBMB4XDTA2MDgxNDEyNTk1M1oXDTExMDgx

NDEyNTk0OVowfzEMMAoGA1UEChMDT1NVMQwwCgYDVQQLEwNCTUkxDzANBgNVBAsTBmNhR3JpZDEP

MA0GA1UECxMGRG9yaWFuMRIwEAYDVQQLEwlsb2NhbGhvc3QxKzApBgNVBAMTIkRvcmlhbiBJZFAg

QXV0aGVudGljYXRpb24gQXNzZXJ0ZXIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAJVJO9Y8

72zGcW2dd2GexQduSlaRc0/hxCQJIjp8K56eeOcjkJrcGB5CRf97BNJ7lPlBqR5jDYLyxubwcMwq

xCzHImliZu9QeSjpugZ+ZIYhdxwa85dpw1RS/MeM3HfQ2lwLfxa+YkhL2DVaGBZBJKkNra36HNE/

2cJKrKCqaX5dAgMBAAGjYDBeMAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BAQDAgWgMB0GA1UdDgQW

BBTCdSL1cZ6NI8sxg5r1K1IMoSSRGzAfBgNVHSMEGDAWgBSbn8ySrMmXOv2cj04a7MMwYK3QujAN

BgkqhkiG9w0BAQQFAAOBgQCbUmHxepezWeatUolwXOE3tk21epYZ1Rtiy/XUcsRN78e1RtG61KKi

4nxAd6jckj4hiSlA3FgR/vIWvI/gQeeTg3d2mpcCxD02aeect60gf/tiN79Rr3KanEgHvrjMgRQb

v2GgNurMLKfO73FPL6iFRQKhWdin5RZCIbiHipnjiQ==</ds:X509Certificate>

</ds:X509Data>

</ds:KeyInfo>

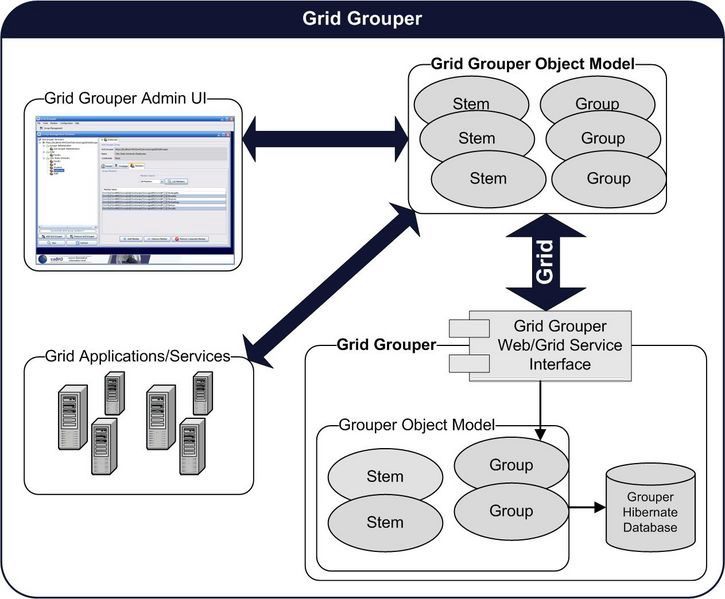
</ds:Signature>

</Assertion>

## Grid Grouper

Grid Grouper is a group/virtual organization management solution for the caGrid supporting group-based authorization. Grid services and applications enforce authorization policy based on membership to groups defined and managed at the caGrid level. Grid Grouper is built on top of Grouper, which is an Internet2 initiative focused on providing tools for group management. Grouper is a java object model which currently supports: basic group management by distributed authorities; subgroups; composite groups (whose membership is determined by the union, intersection, or relative complement of two other groups); custom group types and custom attributes; trace back of indirect membership; delegation. Applications interact with Grouper by embedding Grouper's Java object model inside the application. Grid Grouper is a Grid-enabled version of Grouper (in summary, a Grid service that provides Grouper functionality). It provides a service interface to the underlying Grouper object model. Groups are then available and manageable to applications and other services in the caGrid. Grid Grouper provides an almost identical object model to the Grouper object model on the caGrid client side. Applications and services can use the Grid Grouper object model much like they would use the Grouper object model to access and manage groups and enforce a group-membership authorization policy.

In Grouper/Grid Grouper, groups are organized into namespaces called stems. Each stem can have a set of child stems and set of child groups, with exception of the root stem which cannot have any child groups. For example, let's take a university comprised of many departments, each of which has Faculty, Staff, and Students. To organize the university in the Grid Grouper, a stem would be created for each department. Each department stem would contain three groups: Faculty, Staff, and Students. The Stem hierarchy in Grid Grouper is publicly visible to anyone accessing the service, however the ability to view a group within a stem publicly depends on the privileges for the group. A Stem can have two types of privileges associated with it, the *“Stem Privilege”* and the *“Create Privilege”*. Users with the *“Stem Privilege”* can create, modify, and remove child stems. Users with the *“Create Privilege”* can create, modify, and remove child groups.



*Figure 41 Grid Grouper*

In Grouper/Grid Grouper groups are compromised of a set of metadata describing the group, a set of members in the groups, and a set of privileges assigned to users for protecting access to the group. Grid Grouper provides three mechanisms for adding members to a group: 1) Directly adding a member 2) Adding a subgroup to a group 3) Making a group a composite of other groups. Directly adding a user as a member to a groups is straight forward, these members are referred to as *“Immediate Members”*. Adding a subgroup to a group makes all the members of the subgroup members of the group in which the subgroup was added. Members in a group whose membership is granted by membership in a sub group are referred to as *“Effective Members”*. A group can also be set to be a Composite group. A composite group consists of a set operation (Union, Intersection, Complement) on two other groups. For example a composite group consisting of the Intersection of Group X and Group Y would contain all the members that are both member of Group X and Group Y. Members whose membership is granted through a composite group are referred to as *“Composite Members”*.

To protect access to groups in Grid Grouper, users can be assigned the following privileges on a group: View, Read, Update, Admin, Optin, and Optout. Users with the View privilege can see that the group exists. Users with the Read privilege can read basic information about the group. Users with the Update Privilege can manage memberships to the group as well as administer View, Read, and Update privileges. Users with the Admin privilege can modify/administer anything on the group: metadata, privileges, and memberships. Users with the Optin privilege can add themselves as a member to a group, similarly users with the Opout privilege can remove themselves from a group. By default Grid Grouper grants Read and View privileges to all users on each group.

Initially grid grouper has a root stem with on child stem named *“Grouper Administration”* (grouperadministration). The Grouper Administrative stem contains one group named *“Grid Grouper Administrators”* (grouperadministration:gridgrouperadministrators). The *“Grid Grouper Administrators”* is the super user group for Grid Grouper, all members of this group will have admin privileges on all the stems and groups within Grid Grouper. This group is initially empty, but at least one administrative user must be added during Grid Grouper installation. This can be done using the GridGrouperBootstrapper command line tool.

### Use cases

Following use case diagram shows different use cases for a GridGrouper Administrator.



*Figure 42 GridGrouper use case digram*

Following use case diagram shows different use cases for a GridGrouper users maintaining groups and members. Users with Stem privileges can create, modify and update child stems. Users with “Create” privileges can create, update and remove child groups. Users with Optin and Optout privileges can add or remove themselves from a group.



*Figure 43 GridGrouper use case diagram*

Following use case diagram shows different use cases for a service accessing GridGrouper.



*Figure 44 Grid Grouper use case diagram*

### Interfaces

Following class diagram shows different classes, hierarchies and relations in Grid Grouper.

****The Grid Grouper object model consists of several objects: GridGrouper, Stem, Group, Member, Membership, NamingPrivilege, and AccessPrivilege. The Grid Grouper object corresponds to an instance of a Grid Grouper service, it provides high level operations such as finding stems and groups or determining is a user is a member of a group, etc. The Stem object represents an instance of a stem within Grid Grouper. The Stem object provides operations for managing the stem: viewing metadata, managing child stems, managing child groups, managing stem privileges, etc. The group object models a group instance within Grid Grouper, providing operations for managing metadata, managing privileges, and managing members. Membership object represents a member membership to a group.

*Figure 45 Grid Grouper Interfaces*

### Operations

Following are the operations exposed by GridGrouper web service to interface with its clients.

getStem()

**Description:** Find a stem

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.StemDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* None

**Post-condition:**

* Return stem matched with given identifier.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getChildStems()

**Description:** Find all child stems for a given stem

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.StemDescriptor[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* None

**Post-condition:**

* Return child stems for a parent stem matched with given identifier.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getParentStem()

**Description:** Find a parent stem for a given stem

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.StemDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* None

**Post-condition:**

* Return parent stem matched with given child stem identifier.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getSubjectsWithStemPrivilege()

**Description:** Find all members with stem privilages

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.StemPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemPrivilegeType

**Output:** java.lang.String[]

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* Valid stem identifier and privilege type (“Create” or “Stem”) as input.

**Post-condition:**

* Return all subjects with stem privileges if the request is about stem privileges.
* Return all subjects with create privileges if the request is about create privileges.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getStemPrivileges()

**Description:** Get stem privilages

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

java.lang.String subject

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.StemPrivilege[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemPrivilege

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* Valid stem identifier and subject as input.

**Post-condition:**

* Return all stem privileges for the given stem and subject.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

hasStemPrivilege()

**Description:** Find if a subject has stem privilages

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

java.lang.String subject,

gov.nih.nci.cagrid.gridgrouper.bean.StemPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemPrivilegeType

**Output:** Boolean

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* Valid stem identifier and privilege type (“Create” or “Stem”) and subject as input.

**Post-condition:**

* Return true if given subject has requested privilege.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

grantStemPrivilege()

**Description:** Grant stem privilages to a given subject

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

java.lang.String subject,

gov.nih.nci.cagrid.gridgrouper.bean.StemPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemPrivilegeType

**Output:** None

**Actors:** User with stem privilege

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault,

GrantPrivilegeFault,

InsufficientPrivilegeFault,

SchemaFault

**Pre-condition:**

* Caller has stem privilege

**Post-condition:**

* Assign stem privilege to requested subject on given stem.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

revokeStemPrivilege()

**Description:** Revoke stem privilages to a given subject

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

java.lang.String subject,

gov.nih.nci.cagrid.gridgrouper.bean.StemPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemPrivilegeType

**Output:**

**Actors:** User with stem privilege

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault,

GrantPrivilegeFault,

InsufficientPrivilegeFault,

SchemaFault

**Pre-condition:**

* Caller has stem privilege

**Post-condition:**

* Revoke stem privilege to requested subject on given stem.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

addChildStem()

**Description:** Add a child stem

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier stem,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

java.lang.String extension,

java.lang.String displayExtension

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.StemDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemDescriptor

**Actors:** User with stem privilege

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault,

InsufficientPrivilegeFault,

StemAddFault

**Pre-condition:**

* Caller has stem privilege

**Post-condition:**

* Child stem has been created for given stem.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

deleteStem()

**Description:** Delete a stem

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

**Output:**

**Actors:** User with stem privilege

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault,

InsufficientPrivilegeFault,

StemDeleteFault

**Pre-condition:**

* Caller has stem privilege

**Post-condition:**

* All create and stem privileges on the stem to everyone have been revoked.
* Requested stem has been deleted including its children.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getGroup()

**Description:** Find a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Return group matched with given group identifier.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getChildGroups()

**Description:** Get child groups

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

StemNotFoundFault

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Return child groups matched with given group identifier.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

addChildGroup()

**Description:** Add a child group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

java.lang.String extension,

java.lang.String displayExtension

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** User with create privilege

**Exceptions**: GridGrouperRuntimeFault,

GroupAddFault,

InsufficientPrivilegeFault

**Pre-condition:**

* Caller has create privilege on requested stem.

**Post-condition:**

* Child grouper to given stem has been added and returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

deleteGroup()

**Description:** Delete a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

**Output:**

**Actors:** User with create privilege

**Exceptions**: GridGrouperRuntimeFault,

GroupDeleteFault,

InsufficientPrivilegeFault,

GroupNotFoundFault

**Pre-condition:**

* Caller has create privilege on requested stem.

**Post-condition:**

* Revoke all group privileges from everyone.
* Requested group has been deleted including its children.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

updateStem()

**Description:** Update a stem

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.StemIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.StemUpdate

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemUpdate

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.StemDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** StemDescriptor

**Actors:** User with stem privilege

**Exceptions**: GridGrouperRuntimeFault,

StemModifyFault,

InsufficientPrivilegeFault

**Pre-condition:**

* Caller has stem privilege on requested stem.

**Post-condition:**

* Stem has been updated with given details
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

updateGroup()

**Description:** Update a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.GroupUpdate

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupUpdate

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** Admin

**Exceptions**: GridGrouperRuntimeFault,

GroupModifyFault,

GroupNotFoundFault,

InsufficientPrivilegeFault

**Pre-condition:**

* Caller has create privilege on requested stem.

**Post-condition:**

* Group is updated with given details and updated group is returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

addMember()

**Description:** Add a member to a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String subject

**Output:**

**Actors:** User with create or stem privilege

**Exceptions**: GridGrouperRuntimeFault,

MemberAddFault,

GroupNotFoundFault,

InsufficientPrivilegeFault

**Pre-condition:**

* Caller has create or stem privilege on requested group stem.
* Caller has Admin or Optin privilege on group
* Requested subject exists in Grid Grouper.

**Post-condition:**

* If Subject is a member, immediate member is added to the given group.
* If Subject is a group, sub group is added to the given group.
* Composite group is updated
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getMembers()

**Description:** Find all members in a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.MemberFilter

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MemberFilter

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.MemberDescriptor[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MemberDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Return all members in the given group matched the given filter.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

isMemberOf()

**Description:** Find if a given member is part of a given group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String member,

gov.nih.nci.cagrid.gridgrouper.bean.MemberFilter

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MemberFilter

**Output:** Boolean

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault

**Pre-condition:**

* None

**Post-condition:**

* Return true if given member is part of given group.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getMemberships()

**Description:** Find all members in a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.MemberFilter filter

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MemberFilter

**Output:**  gov.nih.nci.cagrid.gridgrouper.bean.MembershipDescriptor[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MembershipDescriptor

**Actors:** Admin, User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault

**Pre-condition:**

* None

**Post-condition:**

* Return all members in a given group and matched with given filter criteria.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

deleteMember()

**Description:** Delete a member from a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String member

**Output:**

**Actors:** Admin

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

MemberDeleteFault

**Pre-condition:**

* Caller has Admin / Optout privilege on requested group.

**Post-condition:**

* Given member is deleted from the group and from immediate, effective membership.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

addCompositeMember()

**Description:** Add a composite member to a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupCompositeType,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupCompositeType

gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** Admin

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

InsufficientPrivilegeFault,

MemberAddFault

**Pre-condition:**

* Caller has create or stem privilege on requested group stem.
* Caller has Admin or Optin privilege on group
* Requested subject exists in Grid Grouper.

**Post-condition:**

* Member is added to composite group.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

deleteCompositeMember()

**Description:** Delete a composite member from a group

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** Admin

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

InsufficientPrivilegeFault,

MemberDeleteFault

**Pre-condition:**

* Caller has Admin / Optout privilege on requested group.

**Post-condition:**

* Given composite member is deleted from the group.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

grantGroupPrivilege()

**Description:** Grant group privileges to a member

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String subject,

gov.nih.nci.cagrid.gridgrouper.bean.GroupPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupPrivilegeType

**Output:**

**Actors:** Admin

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

InsufficientPrivilegeFault,

**Pre-condition:**

* Caller has Admin privilege on the requested group.
* Caller has stem or create privileges on the requested group

**Post-condition:**

* Requested privileges are granted to given member.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

revokeGroupPrivilege()

**Description:** Revoke group privileges to a member

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String subject,

gov.nih.nci.cagrid.gridgrouper.bean.GroupPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupPrivilegeType

**Output:**

**Actors:** Admin

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

RevokePrivilegeFault,

InsufficientPrivilegeFault,

SchemaFault

**Pre-condition:**

* Caller has Admin privilege on the requested group.
* Caller has stem or create privileges on the requested group

**Post-condition:**

* Requested privileges are revoked to given member.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getSubjectsWithGroupPrivilege()

**Description:** Find all subjects with group privileges

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

gov.nih.nci.cagrid.gridgrouper.bean.GroupPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupPrivilegeType

**Output:** java.lang.String[]

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Matched subjects are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getGroupPrivileges()

**Description:** Get group privileges

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String subject

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupPrivilege[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupPrivilege

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Group privileges for the given group identifier and subject are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

hasGroupPrivilege()

**Description:** Find if a given subject has group privileges

**Input:** gov.nih.nci.cagrid.gridgrouper.bean.GroupIdentifier,

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupIdentifier

java.lang.String subject,

gov.nih.nci.cagrid.gridgrouper.bean.GroupPrivilegeType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupPrivilegeType

**Output:** boolean

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

GroupNotFoundFault,

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Return true if given member has requested privilege in the given group.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

isMember()

**Description:** Find if a given member is part of GridGrouper

**Input:** java.lang.String member,

gov.nih.nci.cagrid.gridgrouper.bean.MembershipExpression

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MembershipExpression

**Output:** boolean

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Return true if given member is part of given group.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getMember()

**Description:** Find a member

**Input:** java.lang.String member

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.MemberDescriptor

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MemberDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

InsufficientPrivilegeFault

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Matched member is returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

getMembersGroups()

**Description:** Find all groups for a given member

**Input:** java.lang.String member

gov.nih.nci.cagrid.gridgrouper.bean.MembershipType

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** MembershipType

**Output:** gov.nih.nci.cagrid.gridgrouper.bean.GroupDescriptor[]

**Namespace:** http://cagrid.nci.nih.gov/1/GridGrouper

**Schema type:** GroupDescriptor

**Actors:** User

**Exceptions**: GridGrouperRuntimeFault,

InsufficientPrivilegeFault

**Pre-condition:**

* Caller has View / Read / Admin / Update / Optin / Optout privilege on requested group.

**Post-condition:**

* Return matched member groups.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for Grid Grouper fault codes.

Following class diagrams shows different operations supported by GridGrouper service.



*Figure 46 GridGrouper class diagram*

### Data Types

Following links provides WSDL and schema definitions for GridGrouper service.

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/)[GridGrouper.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/GridGrouper.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/)[ServiceSecurity.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/ServiceSecurity.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/)[gridgrouper.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gridgrouper/schema/GridGrouper/gridgrouper.xsd)

### Error handling

Following are the fault codes returned by GridGrouper web service for an error processing a request.

GridGrouperRuntimeFault: This fault is returned if there is an unexpected error while processing a request.

*<element name="GridGrouperRuntimeFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

StemNotFoundFault: This fault is returned if requested stem is not found.

*<element name="StemNotFoundFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InsufficientPrivilegeFault: This fault is returned for insufficient privileges to access requested data or operation.

*<element name="InsufficientPrivilegeFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

StemModifyFault: This fault is retuned if there is an error while updating a stem.

<element name="StemModifyFault">

<complexType>

<complexContent>

<extension base="wsrbf:BaseFaultType" />

</complexContent>

</complexType>

</element>

GrantPrivilegeFault: This fault is returned if there is an error while granting a previlage.

<element name="GrantPrivilegeFault">

<complexType>

<complexContent>

<extension base="wsrbf:BaseFaultType" />

</complexContent>

</complexType>

</element>

SchemaFault: This fault is returned if there is an error while creating schema.

<element name="SchemaFault">

<complexType>

<complexContent>

<extension base="wsrbf:BaseFaultType" />

</complexContent>

</complexType>

</element>

RevokePrivilegeFault: This fault is returned if there is an error while revoking a privilege.

<*element name="RevokePrivilegeFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

StemAddFault: This fault is returned if there is an error while adding a stem.

*<element name="StemAddFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

StemDeleteFault: This fault is returned if there is an error while deleting a stem.

*<element name="StemDeleteFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

GroupNotFoundFault: This fault is returned if a group is not found.

*<element name="GroupNotFoundFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

GroupAddFault: This fault is returned if there is an error while adding a group.

*<element name="GroupAddFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

GroupDeleteFault: This fault is returned if there is an error while deleting a group.

*<element name="GroupDeleteFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

GroupModifyFault: This fault is returned if there is an error while updating a group.

*<element name="GroupModifyFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

MemberAddFault: This fault is returned if there is an error while adding a member.

*<element name="MemberAddFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

MemberDeleteFault: This fault is returned if there is an error while deleting a member.

*<element name="MemberDeleteFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

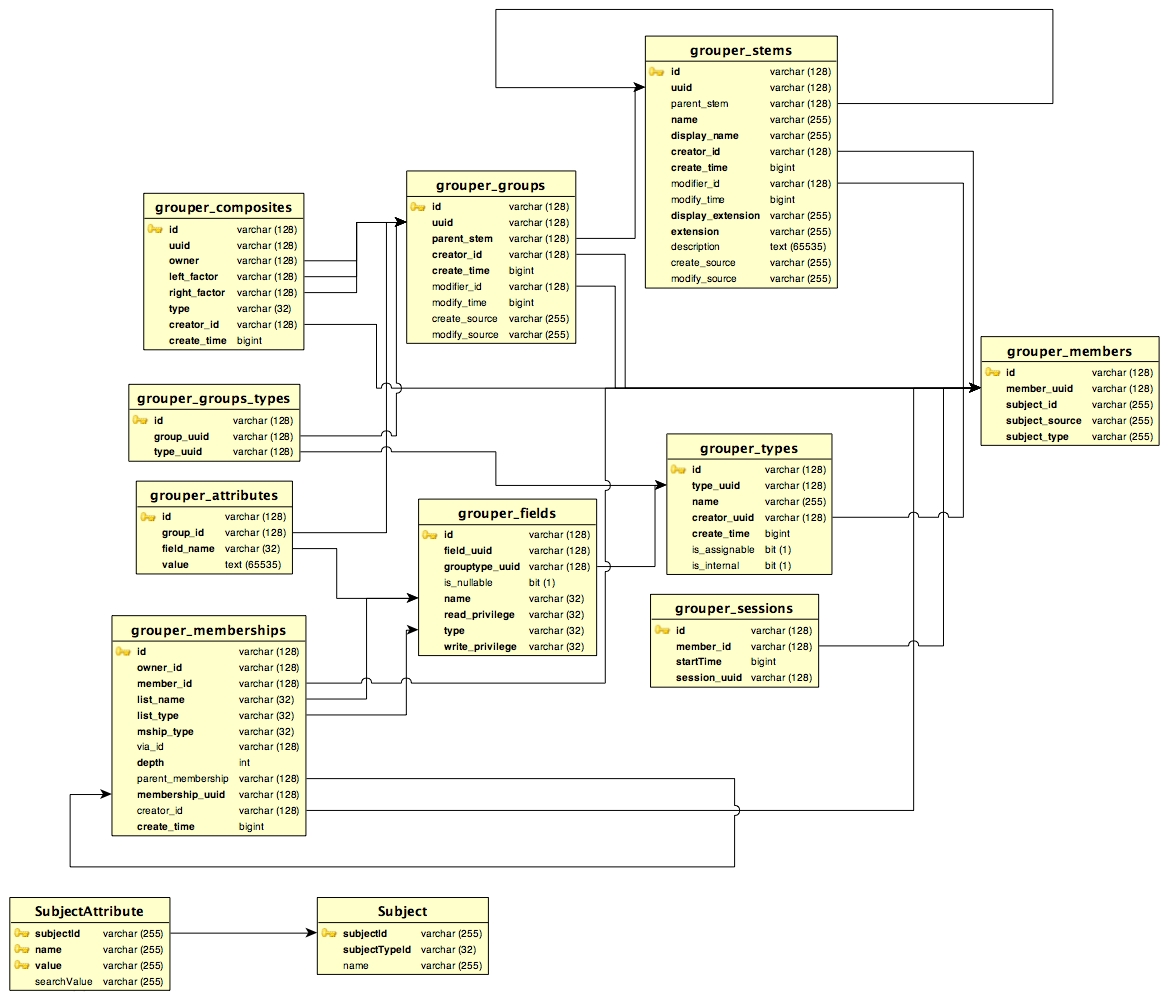
*</complexContent>*

*</complexType>*

*</element>*

### Data Model

Grid Grouper Object persistence is provided by Hibernate, which in turn uses JDBC to connect with a back-end RDBMS. The following diagram describes the relationships between all the Grouper tables.



*Figure 47 Grouper data model*

### Client API

GridGrouper client API would let a user easily interact with its Webservice interfaces. GridGrouperClient class has convenient methods to call all exposed Webservice interfaces.

Following class diagram shows client API hierarchy.



### Configuration

GridGrouper is configured through a configuration file called **grouper.hibernate.properties** located in **GRID\_GROUPER\_LOCATION/resources/conf/**. GridGrouper uses these properties to connect to MySQL database to maintain data.

hibernate.dialect =net.sf.hibernate.dialect.MySQLDialect  
hibernate.connection.driver\_class = com.mysql.jdbc.Driver  
hibernate.connection.url = jdbc:mysql://localhost:3306/grouper  
hibernate.connection.username = root  
hibernate.connection.password = YOUR\_PASSWORD

## Grid Trust Service

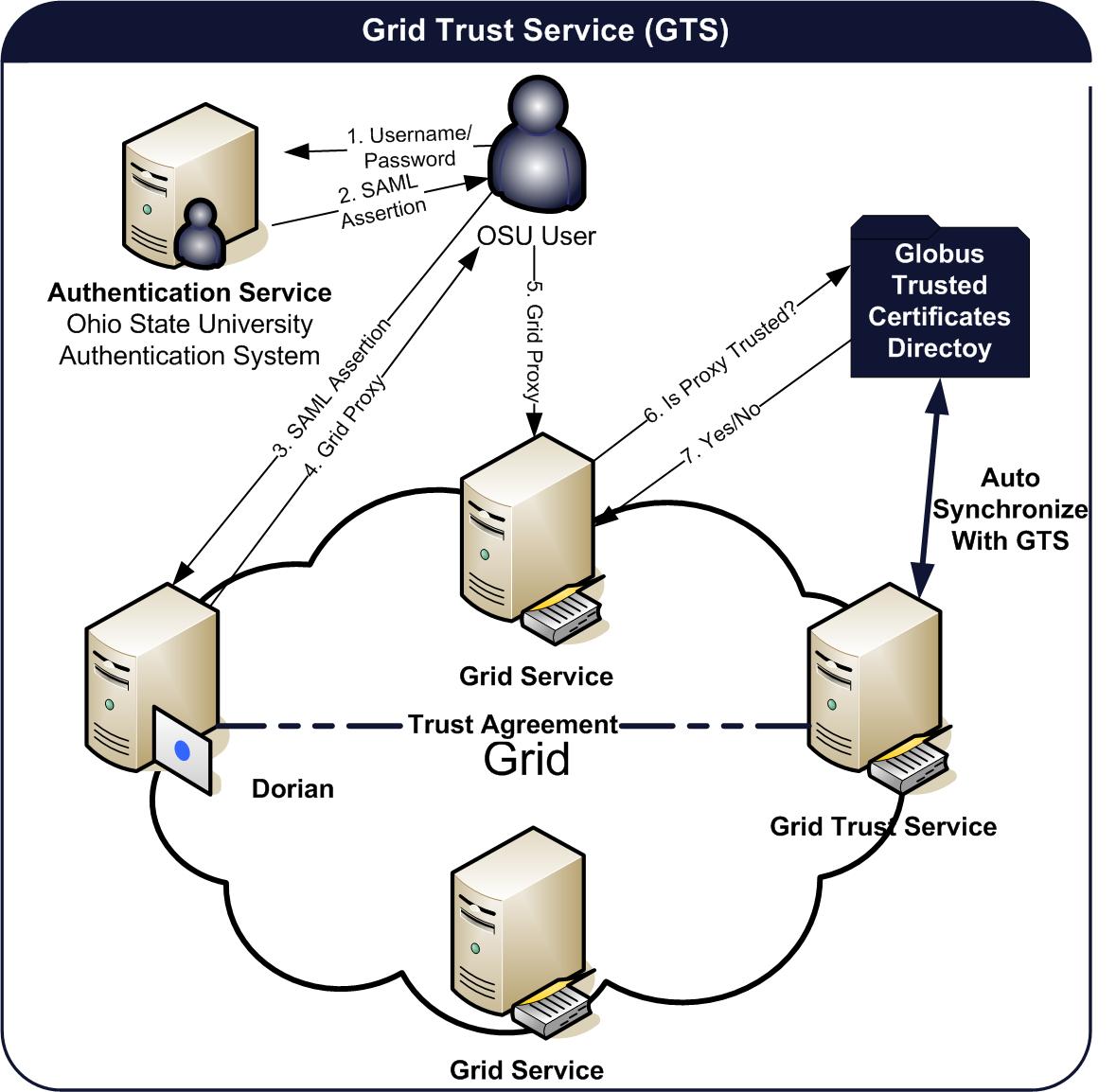
The Globus Toolkit implements support for security via its Grid Security Infrastructure (GSI). GSI utilizes X509 Identity Certificates for identifying and authenticating users. An X.509 certificate with its corresponding private key constitutes a unique credential or so-called "Grid credential" that is used to authenticate both users and services within the caGrid. Under Globus, the authentication process ensures that the X.509 Identity provided by the peer was issued by a trusted certificate authority. Multiple certificate authorities are used on a Grid. Thus, there is a need to manage which CAs are trusted and at what "level of assurance" (that is, how well they are trusted). In a Grid environment, the number of certificate authorities and the number of user identities can grow to be very large. Moreover, in a dynamic multi-institutional environment, the status of identities may be updated frequently. Identities and credentials can be revoked, suspended, reinstated, or new identities can be created. In addition, the list of trusted authorities may change. In such settings, certificate authorities will frequently publish Certificate Revocation Lists (CRL), which specify "blacklisted" certificates that the authority once issued but no longer accredits. For the security and integrity of the caGrid, it is critical to both authenticate and validate a given credential against an accurate list of trusted certificate authorities and their corresponding CRLs. The Grid Trust Service (GTS) is a federated infrastructure enabling the provisioning and management of a Grid trust fabric. The salient features of GTS are as follows:

* A complete Grid-enabled federated solution for registering and managing certificate authority certificates and CRLs, facilitating the enforcement of the most recent trust agreements.
* Definition and management of levels of assurance, such that certificate authorities may be grouped and discovered by the level of assurance that is acceptable to the consumer.
* Due to the federated nature of GTS and its ability to create and manage arbitrary arrangements of authorities by level of assurance, it facilitates the curation of numerous independent trust overlays across the same physical Grid.
* Client validation, allowing a client to submit a certificate and trust requirements in exchange for a validation decision, which allows for centralized certificate verification and validation.

Tiered approach of Globus Toolkit supports following configurations.

* *Locally Stored, Locally Validated* (LSLV) profile, specifies that CA certificates are stored and accessed locally; certificate validation is done against the locally stored CA certificates.
* *Remotely Retrieved, Locally Validated*(RRLV) profile, specifies that CA certificates are retrieved remotely from a Trust Service; certificate validation is done locally against the remotely retrieved CA certificates
* Remotely Stored, Remotely Validated (RSRV) profile, specifies that CA certificates are stored remotely; certificate validation is done remotely against the remotely stored CA certificates.

Out of the box, the Globus Toolkit supports the LSLV profile, and the GTS adds the support for RRLV profile. caGrid use cases for enabling trust between identity providers and consumers of identities and between attribute authorities and consumers of attributes call for the use of Remotely Stored Remotely Validated (RSRV) profile. It is also anticipated that future releases of Globus will support callouts to remote validation services. The GTS supports this profile by providing a validation operation through its service interface. In this manner, the GTS is a validation service allowing clients to submit validation criteria and an X509 certificate chain for validation.



*Figure 48 Grid Trust Service*

*Figure* *48* illustrates an example usage scenario of the RRLV profile. In this example, a GTS instance is used to manage and validate client certificates in caGrid. This example has a Dorian instance serving as a CA and proxy certificate generator. Dorian is a Grid service infrastructure for the management of Grid user accounts.  Dorian provides an integration point between external security domains and the Grid security domain, enabling users to obtain Grid credentials using their locally provided authentication mechanism. In *Figure* *48*, an Ohio State University (OSU) user authenticates to the OSU authentication system using her local user name and password. Upon successfully authenticating, the user is given a signed SAML assertion, which can be given to Dorian in exchange for a Grid proxy or Grid credentials. In the example, a trust agreement has been established between the GTS instance and the Dorian instance wherein the Dorian instance's CA is listed as a trusted authority in the GTS instance. Trust and the trust level between the GTS and Dorian instances can be established by the administrator of Dorian and the administrator of GTS through the exchange of information such as certificate policy and certification process statements. When the user presents the Grid proxy certificate to a Grid service, the certificate is sent to the GTS instance for validation (Step 6 in the figure). The GTS instance can respond back to the service with a "yes" or "no" answer. If the response is "no", the service prevents the user from accessing its resources. If the response is "yes", the service can take additional steps to authorize the user and control his access to the service's functionality based on the authenticated user's privileges.

### Level of Assurance

A Level of Assurance specifies the level of confidence in a given certificate authority.  The level of assurance concept is similar to obtaining an identification card, for example obtaining a passport requires extensive documentation and a thorough background check where as obtaining a library card requires much less documentation and background check. When comparing a passport to a library card for identity validation, most would have more confidence in the passport than in the library card.

In the Web/Grid service environments, certificate authorities issue credentials to users.  Each certificate authority may apply different policies in issuing credentials.  Because of this the level of confidence in the identity represented by the credential will vary between certificate authorities.  Level(s) of assurance can be used to group certificate authorities together that enforce similar policies for issuing credentials.  For example the [**federal e-authentication guidelines**](http://www.cio.gov/eauthentication/) specify four levels of assurance, if adopting these guidelines, certificate authorities can be associated with the levels of assurance that they comply to.  Clients and Services that are consuming credentials issued by certificate authorities can specify which level(s) of assurance they will accepts credentials from, allowing them to enforce the security policies required for the data they are sharing.

Each level of assurance registered to the GTS has a name and a description.  The name uniquely identifies the level of assurance among other levels of assurance.  The description provides information describing the level of assurance.

### Use cases

Following use case diagram shows different use cases for a GTS Administrator to administer Certificate authorities, their certificates and trust levels.



*Figure 49 GTS Administrator use cases*



*Figure 50 GTS CA Administrator use cases*



*Figure 51 GTS use case*

### Interfaces

GTS exposes its functionality through its webservice interfaces that interacts with its subcomponents. TrustLevelManager manages trust levels, PermissionManager manages permissions and TrustedAuthorityManager manages trusted authorities.



*Figure 52 GTS Interfaces*

### Operations

Many of the operations provided by the GTS provide a means of administrating the trust fabric and are therefore restricted to GTS administrators. The GTS allows for the assignment of two types of permissions; GTS Administrators, and Trusted CA Administrators. GTS Administrators are "super users" and can perform any operation on a GTS (i.e. manage certificate authorities, manage trust levels, manage permissions, etc). Trusted CA Administrator permission corresponds to a specific CA, giving a user the ability to update the CRL for the corresponding CA.

The GTS service interface provides several operations for registering and managing trusted certificate authorities. For each trusted certificate authority, the GTS maintains a Certificate Revocation List (CRL). The CRL contains a list of certificates that have been revoked by the CA.



*Figure 53 GTS class diagram*

addTrustedAuthority()

**Description:** Add a trusted authority. Adding of a certificate authority requires the specification of the CA's root certificate, a set of trust levels, a status, and an optional CRL. The CA's root certificate is required for validating certificates. The set of trust levels specifies the level of trust associated with the CA. The status specifies the current state of the certificate authority; the status can be set to "trusted" or "suspended". Setting the status of a certificate authority allows it to be temporarily added and removed from the trust fabric. For each trusted certificate authority, the GTS maintains a Certificate Revocation List (CRL). The CRL contains a list of certificates that have been revoked by the CA.

**Input:** gov.nih.nci.cagrid.gts.bean.TrustedAuthority

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustedAuthority

**Output:** gov.nih.nci.cagrid.gts.bean.TrustedAuthority

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustedAuthority

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalTrustedAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.
  + Trusted authority does not exist in GTS

**Post-condition:**

* Trusted certificate authority is added to GTS
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

findTrustedAuthorities()

**Description:** In order to support the RRLV profile, local validation processes will need a method of discovering and obtaining trusted CAs from the pre-configured GTS. The GTS provides findTrustedAuthorities() through its service interface. In discovering trusted certificate authorities the GTS allows the specification of search criteria. The search criteria include the name of the certificate authority, the trust level, the status, the lifetime, the source GTS, and the authority GTS. Using this operation, validators implementing the RRLV profile can discover a list of trusted certificate authorities based on the trust level. Additionally, trust fabric administrators may leverage this operation for discovering the trust fabric such that they may administer it.

**Input:** gov.nih.nci.cagrid.gts.bean.TrustedAuthorityFilter

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustedAuthorityFilter

**Output:** gov.nih.nci.cagrid.gts.bean.TrustedAuthority[]

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustedAuthority

**Actors:** Admin, User

**Exceptions**: GTSInternalFault,

**Pre-condition:**

* + None

**Post-condition:**

* Return all trusted certificate authorities with trust levels defined.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

removeTrustedAuthority()

**Description:** Remove a trusted authority. Trusted authority trust levels will be removed before removing a trusted authority.

**Input:** java.lang.String trustedAuthorityName

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

InvalidTrustedAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.

**Post-condition:**

* Trusted certificate authority is deleted from GTS
* All permissions associated with trusted certificate authority are revoked.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

addPermission()

**Description:** Add permission to access, manage a trusted authority. A Permission cannot be set on all trusted authorities at one time.

**Input:** gov.nih.nci.cagrid.gts.bean.Permission

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** Permission

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalPermissionFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.

**Post-condition:**

* Given permission is added to given trusted authority.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

findPermissions()

**Description:** Find all permissions assigned to a grid user for a trusted authority.

**Input:** gov.nih.nci.cagrid.gts.bean.PermissionFilter

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** PermissionFilter

**Output:** gov.nih.nci.cagrid.gts.bean.Permission[]

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** Permission

**Actors:** Admin

**Exceptions**: GTSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.

**Post-condition:**

* Return all matched permissions assigned to given grid user identity.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

revokePermission()

**Description:** Revoke a permission assigned to a grid user

**Input:** gov.nih.nci.cagrid.gts.bean.Permission

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** Permission

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

InvalidPermissionFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.

**Post-condition:**

* Given permission is revoked for given grid user.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

updateTrustedAuthority()

**Description:** Update a trusted authority.

**Input:** gov.nih.nci.cagrid.gts.bean.TrustedAuthority

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustedAuthority

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalTrustedAuthorityFault,

InvalidTrustedAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.
  + GTS is authority of given trust authority.
  + Certificate of trust authority is not changed.
  + Source trust service of trust authority is not changed.

**Post-condition:**

* Given trusted authority is updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

addTrustLevel()

**Description:** The trust level specifies the level of confidence with which a given certificate authority is trusted in the fabric in which it is deployed. Clients can specify the level of trust that they require when discovering trusted CAs or when requesting validation. Trust levels in the GTS each consist of a unique name (value) and description. The unique name is used to implicitly bind a certificate authority to a trust level. The description is used as a human readable method of understanding what a specific trust level represents.

**Input:** gov.nih.nci.cagrid.gts.bean.TrustLevel

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustLevel

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalTrustLevelFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.
  + GTS authority of given trust level is specified.
  + Source trust service is given.
  + Authority GTS URI is valid, if authority.

**Post-condition:**

* Given trusted level is added.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

updateTrustLevel()

**Description:** Update trust level.

**Input:** gov.nih.nci.cagrid.gts.bean.TrustLevel

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustLevel

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

InvalidTrustLevelFault,

IllegalTrustLevelFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.
  + GTS is authority of given trust level.
  + Source trust service of trust level is not changed.
  + Authority trust service of trust level is not changed.
  + Valid authority GTS is specified.

**Post-condition:**

* Given trust level is updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

getTrustLevels()

**Description:** Get all trust levels defined in GTS instance.

**Input:**

**Output:** gov.nih.nci.cagrid.gts.bean.TrustLevel[]

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustLevel

**Actors:** User

**Exceptions**: GTSInternalFault,

**Pre-condition:**

* + None

**Post-condition:**

* All defined trust levels are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

removeTrustLevel()

**Description:** Remove a trust level.

**Input:** java.lang.String trustLevelName

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

InvalidTrustLevelFault,

IllegalTrustLevelFault,

PermissionDeniedFault

**Pre-condition:**

* + Caller has Administrator privileges.

**Post-condition:**

* All authorities with given trust level are removed.
* Given trust level is removed.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

addAuthority()

**Description:** Adds a GTS to an authoritative GTS. In order to enable a federated trust fabric, each GTS can be administered to synchronize with a set of authoritative GTSs. GTSs can inherit both trust levels and trusted certificate authorities from its authority GTSs. Registering an authority GTS requires the specification of the following properties: service's uniform resource identifier (URI), priority, whether or not to synchronize the trust levels, time to live, whether or not to perform authorization, and the authority service's identity. The priority property is used for resolving conflicts between authority GTSs, for example if two authority GTSs have a listing for the same certificate authority, the authority GTS with the highest priority will be used for obtaining that certificate authority, and its corresponding information (e.g. its CRL). If contact to an authoritative GTS is lost for a significant amount of time, the trust fabric within the subordinate GTS may become significantly out of date; this could be a potential security risk. The time to live property specifies how long certificate authorities obtained from authoritative GTSs will be valid for in the subordinate GTS. The time to live on a given certificate authority record is reset after each synchronization with the authority GTS. If contact with an authority GTS is lost, the time to live will expire and the certificate authority will be removed from the subordinate's trust fabric.

**Input:** gov.nih.nci.cagrid.gts.bean.AuthorityGTS

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** AuthorityGTS

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has Administrator privileges.
* Service URI is specified.
* Authority does not exist.
* If authorization is required, service identifier is specified.
* Authority priority is unique and is between 1 and total number of authorities.

**Post-condition:**

* Given authority is added.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

updateAuthority()

**Description:** Update an authoritative GTS relation

**Input:** gov.nih.nci.cagrid.gts.bean.AuthorityGTS

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** AuthorityGTS

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalAuthorityFault,

InvalidAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has Administrator privileges.
* Service URI is specified.
* If authorization is required, service identifier is specified.
* Authority priority cannot be changed.
* Valid time to live is specified.

**Post-condition:**

* Given authority is updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

updateAuthorityPriorities()

**Description:** Each authority GTS is assigned a priority, the priority is used in resolving conflict between authorities.  When a conflict arises the information received from the GTS with the highest priority will be used.  This method provides a mechanism for updating the priority of the authority GTSs.

**Input:** gov.nih.nci.cagrid.gts.bean.AuthorityPriorityUpdate

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** AuthorityPriorityUpdate

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

IllegalAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has Administrator privileges.
* Authority priority is set between 1 and total number of existing authorities.
* Authority priority is unique.

**Post-condition:**

* Given authority priority is updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

getAuthorities()

**Description:** Get list of authoritative GTS instances.

**Input:**

**Output:** gov.nih.nci.cagrid.gts.bean.AuthorityGTS[]

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** AuthorityGTS

**Actors:** Admin

**Exceptions**: GTSInternalFault,

**Pre-condition:**

* None

**Post-condition:**

* Return all authoritative GTS instances
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

removeAuthority()

**Description:** Remove an authoritative GTS

**Input:** java.lang.String serviceURI

**Output:**

**Actors:** Admin

**Exceptions**: GTSInternalFault,

InvalidAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has Administrator privileges.

**Post-condition:**

* Given authority is removed.
* All trust levels defined for given authority are removed.
* Trusted authorities set with given authority as source are removed.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

updateCRL()

**Description:** Update certificate revocation list of a trusted certificate authority.

**Input:** java.lang.String trustedAuthorityName,

gov.nih.nci.cagrid.gts.bean.X509CRL crl

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** X509CRL

**Output:**

**Actors:** CA Admin

**Exceptions**: GTSInternalFault,

IllegalTrustedAuthorityFault,

InvalidTrustedAuthorityFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has Administrator privileges with given trusted authority.

**Post-condition:**

* Given trusted authority CRL is updated.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

validate()

**Description:** To support the Remotely Stored, Remotely Validated (RSRV) model, the GTS takes on the role of a validation service. The GTS validate() operation enables clients to submit, for validation, a certificate chain and validation criteria. The GTS uses the X509 validation specifications for validating the certificate chain. Additionally, it enforces the X509 Proxy extensions for validating the certificate chain, if an X509 proxy certificate exists in the chain. The GTS uses the validation criteria to identify a set of certificate authorities to validate the specified certificate chain against. The validation criteria are similar to discovery criteria, optionally allowing the set of certificate authorities to validate against to be limited based on the following: the name of the certificate authority, the trust level, the status, the lifetime, the source GTS, and the authority GTS. This operation validates a X509 certificate chain against the trust fabric.  Specification of a TrustedAuthorityFilter allows the caller to limit the set of trusted certificate authorities to perform the validation against.

**Input:** gov.nih.nci.cagrid.gts.bean.X509Certificate[]

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** X509Certificate

gov.nih.nci.cagrid.gts.bean.TrustedAuthorityFilter

**Namespace:** http://cagrid.nci.nih.gov/8/gts

**Schema type:** TrustedAuthorityFilter

**Output:** boolean

**Actors:** User

**Exceptions**: GTSInternalFault,

CertificateValidationFault

**Pre-condition:**

* None

**Post-condition:**

* Return true if given certificate is valid with given trusted authority.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for GTS fault codes.

### Data Model

Following is the data model diagram for GTS. Each of these entities is described below.



*Figure 54 GTS data model*

**TRUSTED\_AUTHORITIES** table stores information related all trusted certificate authorities in a GTS instance. Adding of a certificate authority requires the specification of the CA's root certificate, a set of trust levels, a status, and an optional CRL. The CA's root certificate is required for validating certificates. The set of trust levels specifies the level of trust associated with the CA. The status specifies the current state of the certificate authority; the status can be set to "trusted" or "suspended". Setting the status of a certificate authority allows it to be temporarily added and removed from the trust fabric. For each trusted certificate authority, the GTS maintains a Certificate Revocation List (CRL). The CRL contains a list of certificates that have been revoked by the CA. Name of the trusted authority is the primary key.

For each certificate authority the GTS maintains the following information:

* ***Certificate -*** The Certificate Authority's certificate which correspond to the private key that the certificate authority uses for signing certificates.  The certificate is needed for validating the signatures of the certificate authority.
* ***Certificate Revocation List (CRL) -*** A list of certificates issued by the certificate authority that have been revoked.  Traditionally distributing CRLS has been a challenging problem, especially in Web/Grid service environments.   The GTS solves this problem by maintaining and distributing CRLS for certificate authorities.
* ***Levels of Assurance -*** Each certificate authority can be assigned many Level(s) of Assurance.   A Level of Assurance specifies a level of confidence in a given certificate authority.  For example the [**federal e-authentication guidelineslinkext7**](http://www.cio.gov/eauthentication/) specify four levels of assurance, if adopting these guidelines, certificate authorities can be associated with the levels of assurance that they comply to.  When provisioning certificate authorities, the GTS will only provision certificate authorities that meet the level of assurance requirements for a given client or service.
* ***Status -*** Specifies the current status of the certificate authority.  A certificate authority can either have a status of ***Trusted*** or ***Suspended***.  The status allows GTS administrators to temporarily suspend a certificate authority without having to remove it.   This is useful in being able to isolate certificate authorities while investigating security breaches.
* ***Federation Metadata -*** For redundancy, scalability, and extensibility reasons GTS(s) can be grouped together to form a federated trust fabric.   The GTS maintains federation metadata for each certificate authority.  This information includes which GTS is the authority for the certificate authority and which GTS was the source of the certificate authority.

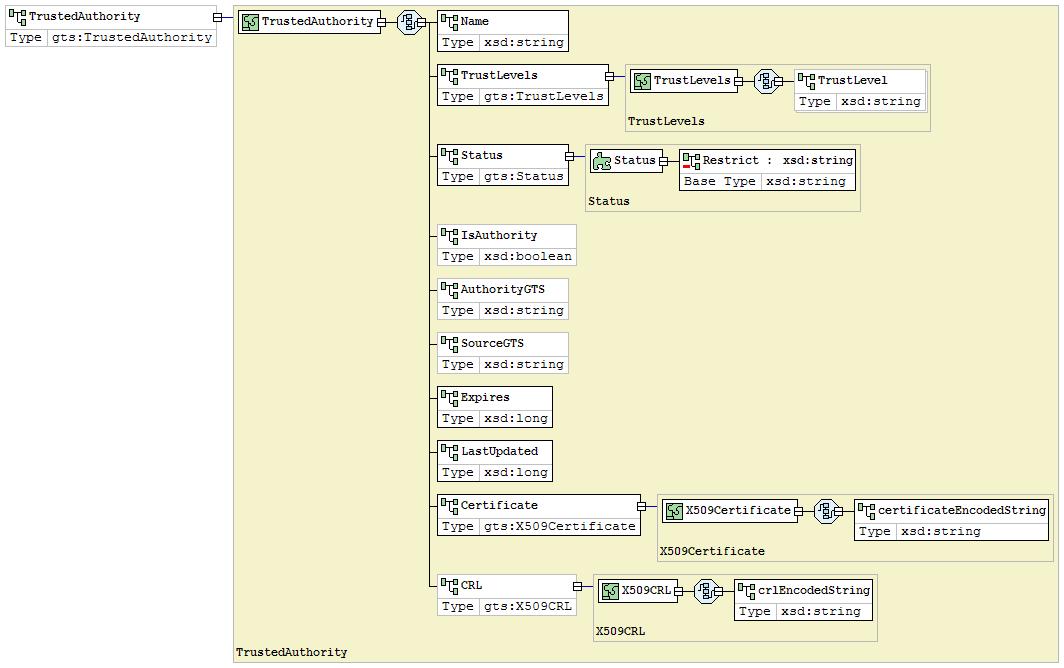
**TRUST\_LEVELS** table stores information related to all trust levels defined in a GTS instance. The trust level specifies the level of confidence with which a given certificate authority is trusted in the fabric in which it is deployed. In the caGrid, one can assume that certificate authorities will be trusted with different levels of confidence. There will be multiple types and instances of certificate authorities. Some authorities may be used to assert identities; other authorities may be used to assert digitally signed documents. Even certificate authorities asserting the same thing may have differing levels of trust associated with them, as they may employ different policies for issuing and validating identities. When certificate authorities are registered into the trust fabric they are assigned one or more trust levels. Clients can specify the level of trust that they require when discovering trusted CAs or when requesting validation. Trust levels in the GTS each consist of a unique name (value) and description. The unique name is used to implicitly bind a certificate authority to a trust level. The description is used as a human readable method of understanding what a specific trust level represents.

**TRUSTED\_AUTHORITY\_TRUST\_LEVELS** table links a trusted authority with one or more trust levels.

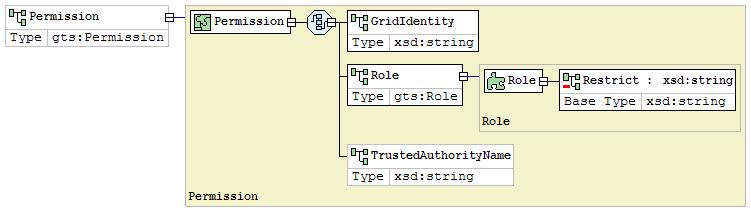
**PERMISSIONS** table stores information that supports administrative permission on a GTS instance. The GTS allows for the assignment of two types of permissions; GTS Administrators, and Trusted CA Administrators. GTS Administrators are "super users" and can perform any operation on a GTS (i.e. manage certificate authorities, manage trust levels, manage permissions, etc). A Trusted CA Administrator permission corresponds to a specific CA, giving a user the ability to update the CRL for the corresponding CA.

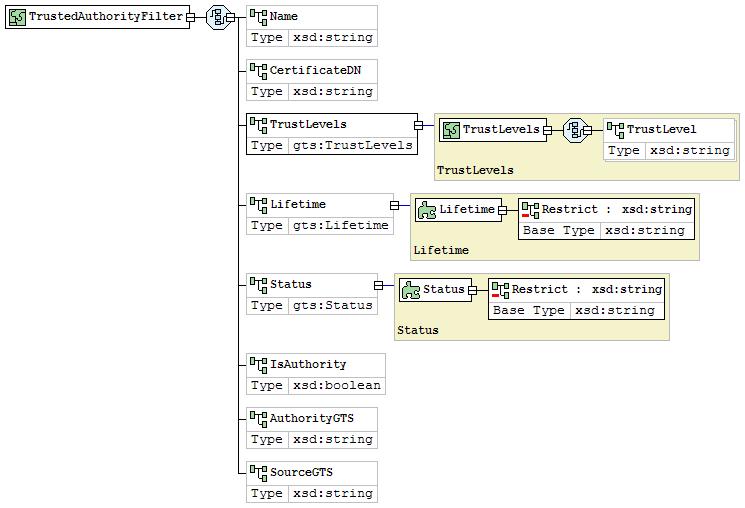
**GTS\_AUTHORITIES** table store information for authoritative GTS instances. Each GTS can be administered to synchronize with a set of authoritative GTSs. GTSs can inherit both trust levels and trusted certificate authorities from its authority GTSs. Registering an authority GTS requires the specification of the following properties: service's uniform resource identifier (URI), priority, whether or not to synchronize the trust levels, time to live, whether or not to perform authorization, and the authority service's identity. The priority property is used for resolving conflicts between authority GTSs. If contact to an authoritative GTS is lost for a significant amount of time, the trust fabric within the subordinate GTS may become significantly out of date; this could be a potential security risk. The time to live columns specifies how long certificate authorities obtained from authoritative GTSs will be valid for in the subordinate GTS. The time to live on a given certificate authority record is reset after each synchronization with the authority GTS. If contact with an authority GTS is lost, the time to live will expire and the certificate authority will be removed from the subordinate's trust fabric.

### Data Types

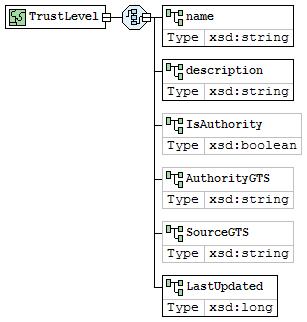
Following are the data types defined for GTS.

*Figure 55 GTS TrustedAuthority type*

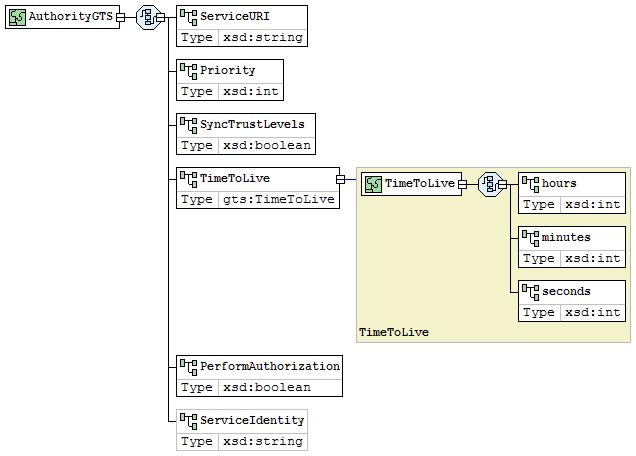
***Figure 56 GTS permission type*

**

*Figure 57 GTS TrustedAuthorityFilter type*



*Figure 58 GTS TrustLevel type*

***Figure 59 GTS AuthorityGTS type*

Full list of data types and WSDL can be referred at

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/gts/schema/GTS/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gts/schema/GTS/)[gts.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gts/schema/GTS/gts.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/gts/schema/GTS/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gts/schema/GTS/)[GTSTypes.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gts/schema/GTS/GTSTypes.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/gts/schema/GTS/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gts/schema/GTS/)[GTS.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/gts/schema/GTS/GTS.wsdl)

### Error Handling

GTS web service returns following fault code on any error servicing a client request.

GTSInternalFault: This fault is returned on any unexpected error serving a client request.

*<element name="GTSInternalFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

IllegalTrustedAuthorityFault: This fault is returned if validation failed during add, update or remove of a trusted authority.

*<element name="IllegalTrustedAuthorityFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

PermissionDeniedFault: This fault code is returned if permission is denied acessing certain operation or data.

*<element name="PermissionDeniedFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidTrustedAuthorityFault: This fault code is returned if givne trusted authority information is not valid.

*<element name="InvalidTrustedAuthorityFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

IllegalPermissionFault: This fault code is returned if given permission information is not valid.

*<element name="IllegalPermissionFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidPermissionFault: This fault code is returned if given permission is not valid.

*<element name="InvalidPermissionFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

IllegalTrustLevelFault: This fault code is returned if given trust level information is not valid.

*<element name="IllegalTrustLevelFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidTrustLevelFault: This fault code is returned if given trust level is not valid.

*<element name="InvalidTrustLevelFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

IllegalAuthorityFault: This fault code is returned if given authority information is not valid.

*<element name="IllegalAuthorityFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

InvalidAuthorityFault: This fault code is returned if given authority URL is not valid.

*<element name="InvalidAuthorityFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

CertificateValidationFault: This fault code is returned if a certificate chain could not be validated or no trusted roots found.

*<element name="CertificateValidationFault">*

*<complexType>*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

*</element>*

### Client API

Following class diagram shows GTS client API.

GTSPublicClient provides client APIs to find trusted authorities, trust levels and authority GTS instances. It also provides API to validate X.509 certificate with given trusted authorities.

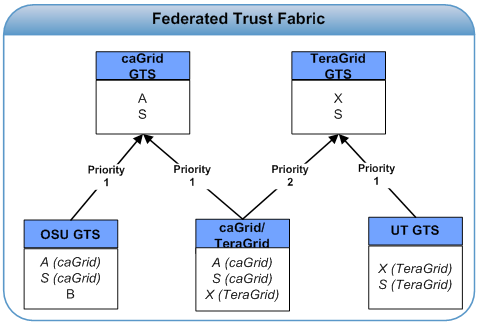
GTSAdminClient provides client API to maintain trusted authorities, permissions, trust levels and to update CRL.



*Figure 60 GTS client class diagram*

### Federation

Redundancy and scalability are critical properties of a federated trust fabric. Serious performance implications will occur if all entities in the federation are discovering and performing validation against a trust fabric maintained in a central GTS. In order to enable a federated trust fabric, each GTS can be administered to synchronize with a set of authoritative GTSs. GTSs can inherit both trust levels and trusted certificate authorities from its authority GTSs. Registering an authority GTS requires the specification of the following properties: service's uniform resource identifier (URI), priority, whether or not to synchronize the trust levels, time to live, whether or not to perform authorization, and the authority service's identity. The priority property is used for resolving conflicts between authority GTSs, for example if two authority GTSs have a listing for the same certificate authority, the authority GTS with the highest priority will be used for obtaining that certificate authority, and its corresponding information (e.g. its CRL). If contact to an authoritative GTS is lost for a significant amount of time, the trust fabric within the subordinate GTS may become significantly out of date; this could be a potential security risk. The time to live property specifies how long certificate authorities obtained from authoritative GTSs will be valid for in the subordinate GTS. The time to live on a given certificate authority record is reset after each synchronization with the authority GTS. If contact with an authority GTS is lost, the time to live will expire and the certificate authority will be removed from the subordinate's trust fabric.



*Figure 61 GTS Federation*

*Figure* 61 illustrates an example of how multiple GTSs can be deployed to create and manage a federated trust fabric. In the example there are five GTSs: caGrid GTS, TeraGrid GTS, OSU GTS, caGrid/TeraGrid GTS, and UT GTS. The caGrid GTS has no authority GTSs, it manages the certificate authorities A and S. The TeraGrid GTS has no authority GTSs, and it manages the certificate authorities X and S. The OSU GTS has one authority GTS, the caGrid GTS. The OSU GTS inherits the certificate authorities A and S from its authority the caGrid GTS. The OSU GTS manages an additional certificate authority B. The OSU GTS is an example of how the global trust fabric can be extended to include local trusted certificate authorities, in this case and the additional certificate authority CA B, which is trusted by OSU. The caGrid/TeraGrid GTS has two authority GTSs, the caGrid GTS and the TeraGrid GTS. The TeraGrid GTS inherits CA A from the caGrid GTS and CA X from the TeraGrid GTS, since the caGrid GTS has a higher priority than the TeraGrid GTS, it inherits CA S from the caGrid GTS. The caGrid/TeraGrid GTS is an example of how two existing trust fabrics from two different Grids can be joined together. Finally the UT GTS has one authority GTS, the TeraGrid GTS. The UT GTS inherits CA X and CA S from the TeraGrid GTS. The UT GTS is an example of standing up a GTS for better redundancy and scalability.

 Supporting a federated trust fabric across GTSs introduces additional metadata to be associated with trust levels and certificate authorities. This metadata includes the "Source GTS", "Authority GTS", and "Time to Live". The "Source GTS" specifies the service URI of the GTS in which the trust level or certificate authority was inherited from. The "Authority GTS" specifies the service URI of the GTS that is the authority of trust level or certificate authority. "Time to Live" specifies the date until which the certificate authority entry is valid.

### Configuration

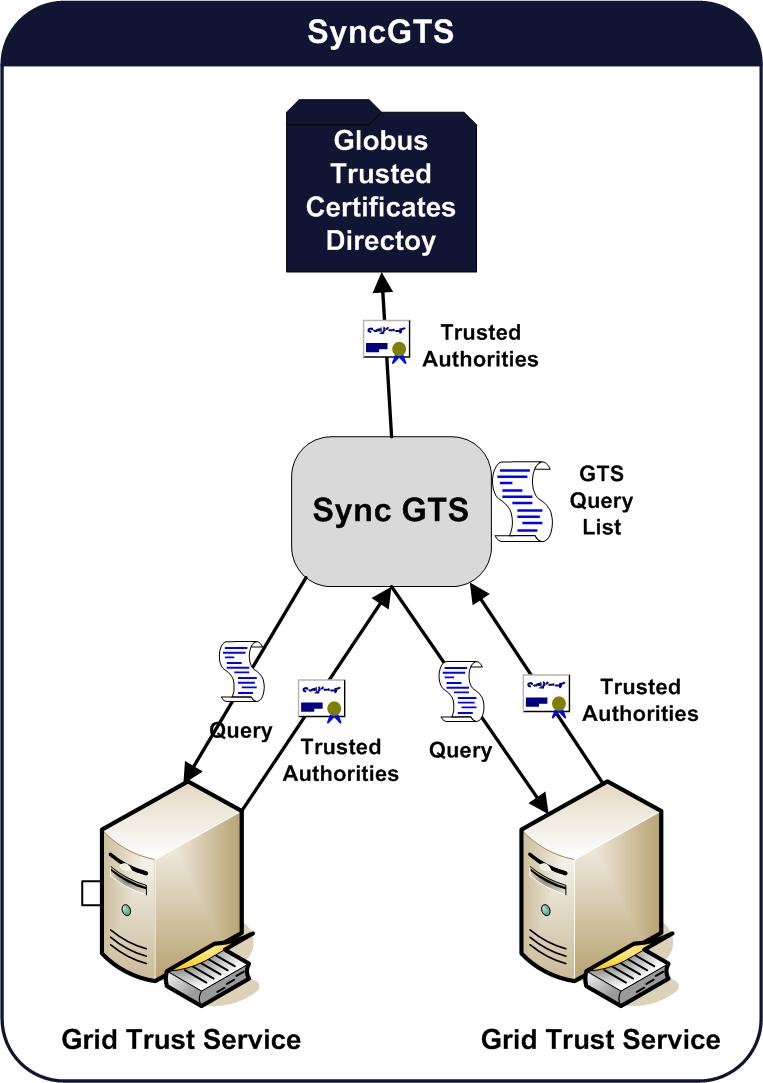
The GTS is configured through a configuration file which is located at GTS\_HOME/etc/gts-conf.xml. The GTS uses a Mysql Database as its backend data store; you must provide the GTS with the connection details for your Mysql database. The ***database*** element in the GTS configuration is used to specify the connection information for your Mysql database. When the GTS is first initialized it will create a database, named with the value of the ***gts-internal-id*** element. The GTS will also proceed to setup its database schema in the database it created. In order to do so the GTS needs to be configured with a database user that has the appropriate permissions. If you do not wish to provide the GTS with such a user you may create the database manually and provide the GTS with a user whom has the permission to modify the database schema. In this scenario the GTS will not create the database but will proceed to setup its database schema in the database that was manually created.  Finally the ***sync-authorities*** element specifies how often the GTS should sync with its authorities.

### SyncGTS

Many clients and services (including caGrid) leverage the Globus Toolkit as their underlying framework.  Globus requires the use of PKI credentials for authentication.   A PKI credential consists of a X.509 certificate and private key, the X.509 certificate is signed by a certificate authority.   In authenticating parties Globus ensure that the party authenticating is the holder of the private key that is bound to the X.509 certificate that they present and that the X.509 certificate presented is signed by a trusted certificate authority.  Globus maintains a list of certificate authority's that it trusts in a local trust store on the local file system.   Although this it is effective it is very limiting and difficult to manage because every time (1) a new certificate authority is trusted, (2) a existing certificate authority is no longer trusted, or (3) a certificate authority updates is CRL, the local trust store of all clients and services need to be updated.  Under the core Globus release this is a manual process which is not scalable in large distributed Grids.  The Grid Trust Service (GTS) is a grid service for managing certificate authorities (and CRLS) that are trust by a community.  The GTS provides a tool called SyncGTS which keeps all clients and services local trust stores in sync with the certificate authorities that are trusted by the GTS.  With SyncGTS, the local trust store for each client and service is updated each time a new certificate authority is added to the GTS or each time a certificate authority is removed from the GTS or each time the CRL for a certificate authority is updated.

SyncGTS provides several configuration options to clients and services.  These include syncing with multiple GTS(s) and specifying level of assurance requirements.  SyncGTS also provides several deployment options making it adaptable and easy to integrate with many types of systems.  These deployment options include:

* [***Globus Runtime***](http://cagrid.org/display/gts13/SyncGTS+Globus+Runtime) ***-*** SyncGTS is deployed directly into a container hosting Web/Grid services, keeping the entire container and services operating in it in sync with the trust fabric.
* [***Command Line***](http://cagrid.org/display/gts13/SyncGTS+Command+Line) ***-*** The command line approach is intended to be used to sync client environments with the trust fabric.
* [***Programmatically***](http://cagrid.org/display/gts13/SyncGTS+Client+API) ***-*** SyncGTS provides a client API, that allows developers to integrate SyncGTS into applications and other software projects.



*Figure 62 SyncGTS*

SyncGTS uses an XML file to describe what and how to synchronize the local environment with the trust fabric. This XML file is referred to as the sync description and is located at: SYNC\_GTS\_HOME/ext/resources/sync-description.xml.

SyncGTS reads following configuration information from Sync description:

GTS Service URI: This URI is used to connect to GTS service to synchronize trusted authorities.

Perform Authorization: Flag to force authorization while connecting to GTS.

Excluded CAs: CAs excluded from the list of trusted authorities

NextSync: Synchronization interval

Following activity diagram shows SyncGTS internal activities synchronizing a GTS with an authority GTS. As mentioned above, SyncGTS can be started manually, programmatically or through deployment of SyncGTSServlet in a web container. When SyncGTS starts its process, it connects to an authority GTS and gets all trusted authorities and synchronizes the list with Globus local trust store.



*Figure 63 SyncGTS activity diagram*

Following class diagram shows class hierarchy and relations of SyncGTS API.

HistoryManager provides API to access synchronization history report that can be filtered by date. SyncGTSConfiguration reads configuration information from SYNC\_GTS\_HOME/ext/resources/sync-description.xml and make it available to SyncGTSImpl.

SyncGTS is a singleton gets loaded into JVM memory on initial call and supports following methods to invoke synchronization.

**syncAndResync():** This synchronized method would start synchronization process and keep running the process at given intervals.

**syncAndResyncInBackground():** This asynchronous method would kick off a new thread and start synchronization process and keep running the process in the background at given intervals.

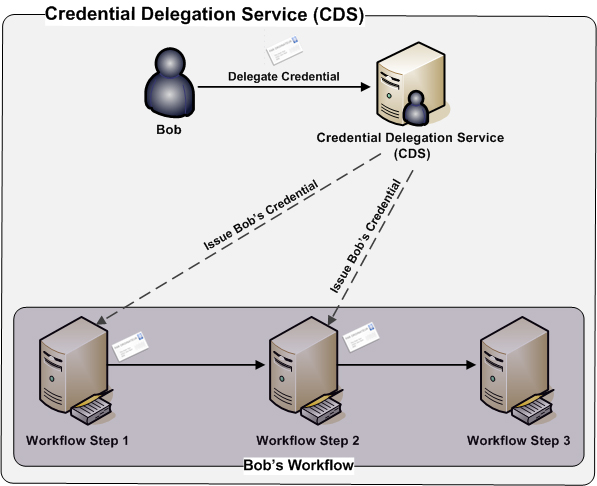
**syncOnce():** This method would run synchronization process one time.

To protect accuracy of trusted authorities list, SyncGTS implements locking mechanism to avoid racing condition with multiple invocations of its process. Only one synchronization process can run at a time.

*Figure 64 SyncGTS class diagram*

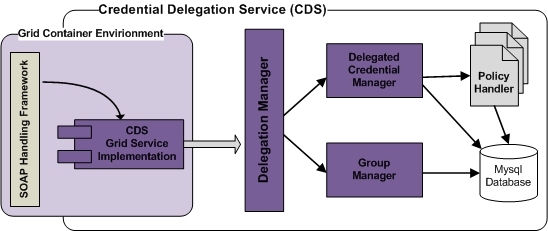
## Credential Delegation Service

The CDS is used by a caGrid user to securely and temporarily hand their credentials to a target service (or another user) to allow the service to perform work on their behalf. A primary use case for credential delegation is workflow. The brief explanation of how this works is as follows. A user logs on to the caGrid to retrieve their credentials. The user then wants to execute an analysis workflow which requires access to two data services and a chain of analytical services to process the data from the two data services. The user composes the workflow and submits it to the workflow management service of caGrid. In this scenario, the user needs the workflow management service to have his Grid credential to perform work on his behalf. For example, the workflow management service needs to submit a query to the two data services and invoke the first service in the processing chain with the results of the query, and then invoke the second service in the chain with the output from the first service, and so on. The workflow management service will need to be handed the user's credentials to access these services. The user delegates his credentials to the CDS, specifying that the workflow management service (identified by using the service's Grid identity) can retrieve the user's credentials for a limited period of time. Once the delegation is complete, the user hands a reference to the delegated credential to the workflow service. The workflow service uses the reference to actually retrieve the user's Grid credential and run the analysis.



*Figure 65 Credential Delegation service usage*

The CDS Grid Service Implementation component implements the CDS web service interfaceand is responsible for handling invocations received by the underlying web service container. For each invocation received the CDS Grid Service Implementation together with the underlying Grid container authenticates each requestor and desterilizes requests into associated Java objects. The request and associated Java objects are then passed to the Delegation Manager component.



*Figure 66 Credential Delegation service architecture*

The Delegation Manager component is responsible on enforcing access control on all request, once it is determined that a client is allowed to perform the request, the Delegation Manager passes the request onto the sub-component that specializes in the request. Requests associated with delegating credentials, obtaining delegated credentials, and monitoring delegated credentials are handled by the Credential Delegation Manager component. The Credential Delegation Manager component employs a set of pluggable Policy Handler(s) each of which manages and enforces the access control policy for delegated credentials. The ability to plug in Custom Policy Handler(s) to the CDS allows customized delegation policies to be used by clients when delegating their credentials to other clients. For example a client could delegated their credentials to (1) clients whom are a member of a specified Grid Grouper group or (2) or clients that are approved by a specified Common Security Module (CSM) policy or (3) clients that meet some other customized policy.

The Group Manager component is responsible for managing the administrators of the CDS. Request received by the DelegationManager for adding, removing, listing, and determining if a client is an administrator are handled by the Group Manager component.

### Use cases

Following diagram shows actors and use cases for Credential delegation.

#### 

*Figure 67 Credential delegation use case*

### Interfaces

Following class diagram shows different interfaces serving CredentialDelegationService. Each of these interface are detailed below.

**

*Figure 68 CDS Interfaces*

**DelegationManager**

DelegationManager is responsible for all operations facilitating credential delegation. It is also responsible for Admin operations. Following is the class diagram of DelegationManager showing attributes and operations.



**DBKeyManager**

DBKeyManager is responsible to encrypt private keys stored in the database with the password set in the configuration. Following is the class diagram of DBKeyManager showing attributes and operations.

### 

DelegatedCredentialManager is responsible for managing delegated credential. Following is the class diagram of DelegatedCredentialManager showing attributes and operations.



### Operations

Following class diagram shows different operations supported CredentialDelegationService. Each of these operations is described below.



*Figure 69 CredentialDelegationService class diagram*

initiateDelegation()

**Description:** This initial step of delegation will generate a key pair and store the information provided and the key pair in its database. The key pair generated will be used to make up a delegated credential for the client making the request. This delegated credential will be used to further delegate credentials to clients that are allowed by the delegation policy provided. The delegated credential will be made up of the generated private key and a certificate containing the generated public key. The certificate will be signed by the client making the request.

**Input:** org.cagrid.gaards.cds.common.DelegationRequest

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationRequest

**Output:** org.cagrid.gaards.cds.common.DelegationSigningRequest

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationRequest

**Actors:** Delegator

**Exceptions**: CDSInternalFault,

PermissionDeniedFault,

InvalidPolicyFault,

DelegationFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS using the credential to delegate.
* Delegation policy is provided.

**Post-condition:**

* Key pair is generated and store along with the request
* Signing request is returned with the public key generated.
* Delegation status is set to “pending”
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

approveDelegation()

**Description:** After successful initiateDelegation() execution, the client should sign the certificate returned by CDS with the private key of the credential that it is delegated. The client re-authenticates with the CDS using the credential being delegated and returns the signed certificate to the CDS. Upon receiving the signed certificate the CDS stores the certificate with the delegated credential record created during the Initiation Step. The signed certificate along with the earlier generated private key make up a credential which can be delegated to other clients based on the delegation policy specified. Finally the CDS creates a web service resource for the delegated credential and returns a reference to the resource (DelegatedCredentialReference) to the client. Client wishing to obtain the delegator's credential can use the DelegatedCredentialReference to request a credential from the CDS.

**Input:** org.cagrid.gaards.cds.common.DelegationSigningResponse

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationSigningResponse

**Output:** org.cagrid.gaards.cds.delegated.stubs.types.DelegatedCredentialReference

**Namespace:** http://cds.gaards.cagrid.org/CredentialDelegationService/DelegatedCredential/types

**Schema type:** DelegatedCredentialReference

**Actors:** Delegator

**Exceptions**: CDSInternalFault,

PermissionDeniedFault,

DelegationFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS using the credential to delegate.
* Certificate returned by CDS is signed by the client with its private key of the credential that it is delegated.
* Caller initiated the delegation.
* Delegation status is “pending”.

**Post-condition:**

* Credential has been generated with the client signed certificate along with CDS generated private key.
* Credential is stored in the database.
* Credential reference is returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

findDelegatedCredentials()

**Description:** Find delegated credentials for a given criteria.

**Input:** org.cagrid.gaards.cds.common.DelegationRecordFilter

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationRecordFilter

**Output:** org.cagrid.gaards.cds.common.DelegationRecord[]

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationRecord

**Actors:** Delegatee

**Exceptions**: CDSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.

**Post-condition:**

* If caller is Administrator, all delegated credentials matched with filter criteria are returned.
* If caller is not Administrator, caller issued delegation credentials matched with filter criteria are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

updateDelegatedCredentialStatus()

**Description:** Update status on a delegated credentials

**Input:** org.cagrid.gaards.cds.common.DelegationIdentifier,

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationIdentifier

org.cagrid.gaards.cds.common.DelegationStatus

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationStatus

**Output:**

**Actors:** Delegator, Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* If caller is administrator, requested delegated credential status is not in “Pending” status.
* If caller is not administrator, current delegated credential status is in “Approved” status and the request is to put in “Suspended” status.

**Post-condition:**

* Delegation record is updated with requested status.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

findCredentialsDelegatedToClient

**Description:** Find all credentials delegated to a client for a given criteria

**Input:** org.cagrid.gaards.cds.common.ClientDelegationFilter

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** ClientDelegationFilter

**Output:** org.cagrid.gaards.cds.common.DelegationDescriptor[]

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationDescriptor

**Actors:** Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* Caller is authorized to access delegation records.

**Post-condition:**

* All credential delegation records matched with the given filter and with “Approved” status, valid expiration are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

searchDelegatedCredentialAuditLog

**Description:** Search audit logs for a given criteria. CDS captures delegation initiation, delegation approval, delegation status update, credentials issued, access denied to credentials audit log information.

**Input:** org.cagrid.gaards.cds.common.DelegatedCredentialAuditFilter

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegatedCredentialAuditFilter

**Output:** org.cagrid.gaards.cds.common.DelegatedCredentialAuditRecord[]

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegatedCredentialAuditRecord

**Actors:** Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault,

DelegationFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* If caller is not admin, delegation record identifier is provided.

**Post-condition:**

* If caller is admin, all credential delegation audit records are returned.
* If caller is not admin, all credential delegation audit records matched with given identifier are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

deleteDelegatedCredential()

**Description:** Delete a delegated credential

**Input:** org.cagrid.gaards.cds.common.DelegationIdentifier

**Namespace:** http://gaards.cagrid.org/cds

**Schema type:** DelegationIdentifier

**Output:**

**Actors:** Delegator, Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault,

DelegationFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* Caller has CDS administrator privileges

**Post-condition:**

* Requested delegated credential record is deleted.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

addAdmin()

**Description:** Add administrator rights to a grid user

**Input:** java.lang.String gridIdentity

**Output:**

**Actors:** Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* Caller has CDS administrator privileges

**Post-condition:**

* Given grid user identity is granted with administrator privileges.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

removeAdmin()

**Description:** Remove CDS administrator rights to a grid user.

**Input:** java.lang.String gridIdentity

**Output:**

**Actors:** Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* Caller has CDS administrator privileges

**Post-condition:**

* Given grid user identity is revoked from administrator privileges.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

getAdmins()

**Description:** Get all grid users designated as CDS Administrators

**Input:**

**Output:** java.lang.String[]

**Actors:** Admin

**Exceptions**: CDSInternalFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* Caller has CDS administrator privileges

**Post-condition:**

* All grid users with CDS administrator privileges are returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

### DelegatedCredentialService

Apart from CredentialDelegationService that is responsible for credential delegation, CDS provides another webservice interface, DelegatedCredentialService to access delegated credentials.

Following is the only one operation supported by DelegatedCredentialService.



getDelegatedCredential()

**Description:** Get a delegated credential authorized to the user.

**Input:** org.cagrid.gaards.cds.common.PublicKey

**Namespace: http://gaards.cagrid.org/cds**

**Schema type:** PublicKey

**Output:** org.cagrid.gaards.cds.common.CertificateChain

**Namespace: http://gaards.cagrid.org/cds**

**Schema type:** CertificateChain

**Actors:** Admin, User

**Exceptions**: CDSInternalFault,

DelegationFault,

PermissionDeniedFault

**Pre-condition:**

* Caller has active grid user account and authenticated with CDS.
* Requested credential has been delegated by delegator with approved status and not expired.
* Caller is approved to access delegated credential through delegation policy set by delegator.
* A key pair is generated by the caller and its public key is provided.

**Post-condition:**

* Certificate provided by the caller is signed with private key of delegated credential.
* Requested delegated credential is returned.
* Fault code is returned if there is any error in processing the request. Please see “Error Handling” section for CDS fault codes.

### Delegating a Credential

The diagram below illustrates the process of a client (delegator) delegating a credential to the CDS. The first step or Initiation step, initiates the delegation process.

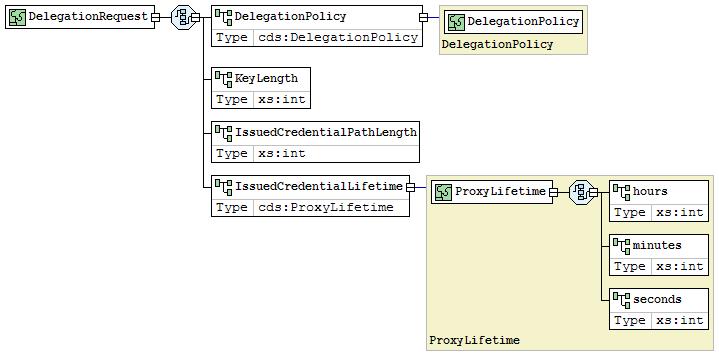


*Figure 70 CredentialDelegation sequence diagram*

To execute the ***Initiation*** step the client must have the credential they wish to delegate in hand. The client must also provide the following information to the CDS:

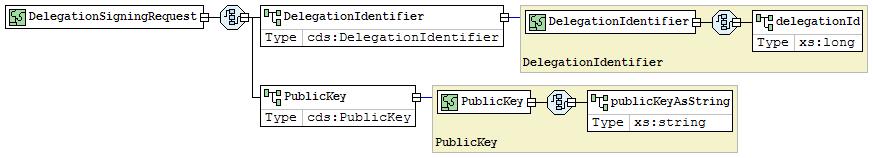
* **Delegated Credential Lifetime -** The amount of time that the CDS will be allowed to delegate the client's credential to allowed parties.
* **Delegated Credential Path Length -** Specifies how much the credential may be further delegated. A path length of 1 would allow the CDS to delegate the client's credential to other parties, however the other parties would not be allowed to further delegate. A path length of 2 would allow the CDS to delegated to second parties, the second parties would also be able to delegate the client's credentials to third parties, the third parties would not be allowed to further delegate.
* **Issued Credential Lifetime -** Specifies the amount of time that a client's credential issued to a allowed party by the CDS would be valid for.
* **Issued Credential Path Length -** Specifies whether of not a client's credential issued to an allowed party by the CDS could be further delegated. A path length of 0 would not allow the party to further delegate. A path length of 1, would allow the party to delegate to a second party, however the seconds party would not be able to further delegate.
* **Delegation Policy -** A policy that expresses which parties may request the delegators credential. The CDS provides a framework for plugging in and enforcing any type of Delegation policy.

Following is the type definition of DelegationRequest.



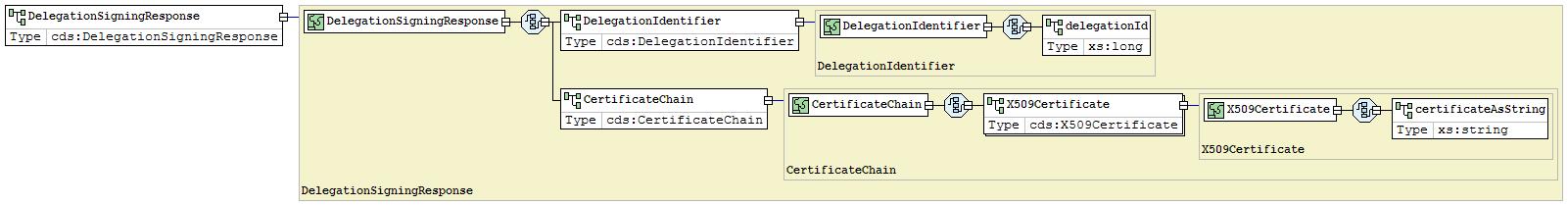
*Figure 71 DelegationRequest*

Upon successful execution of initiation step, CDS service would return DelegationSigningRequest to the delegator. Following diagram shows type definition of DelegationSigningRequest.



*Figure 72 DelegationSigningRequest*

When the delegator receives the signing request, it creates a certificate containing the public key provided by the CDS in the DelegationSigningRequest. The client signs the certificate with the private key of the credential that it is delegated. The client re-authenticates with the CDS using CredentialSigningResponse. Upon receiving the signed certificate through CredentialSigningResponse, the CDS stores the certificate with the delegated credential record created during the Initiation Step. The signed certificate along with the earlier generated private key make up a credential which can be delegated to other clients based on the delegation policy specified. Finally the CDS creates a web service resource for the delegated credential and returns a reference to the resource (DelegatedCredentialReference) to the client. Client's wishing to obtain the delegator's credential can use the DelegatedCredentialReference to request a credential from the CDS.

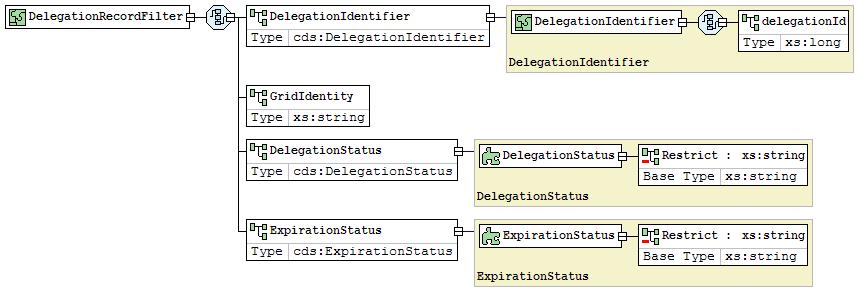
*Figure 73 DelegationSigningResponse*

### Obtaining a delegated credential

To obtain a credential, the client obtaining the credential must have in hand a DelegatedCredentialReference referring to the credential they wish to obtain. A DelegatedCredentialReference may be obtained directly from the delegator or directly from the CDS by asking the CDS which credentials have been delegated to them. Before making the request for the credential to the CDS, the client must generated a public/private key pair which will make up the credential. Once generated the client authenticates to the CDS using their credential and passes the CDS the DelegatedCredentialReference and the generated public key. Upon receiving the request from the client, the CDS check that the client against the delegation policy to validate that the client has been granted the ability to obtain the requested credential. If the client is authorized the CDS creates a certificate containing the public key supplied by the client and signs the certificate with the private key associated with the credential delegated the CDS by the delegator. The signed certificate is then returned to the client, the signed certificate along with the private key generated earlier by the client make up a credential that can be used to invoke secure services on the delegator's behalf.

***Figure 74 Obtaining delegated credential sequence diagram*

Following diagram shows type definition of DelegationRecordFilter used to obtain delegated credentials.



*Figure 75 DelegationRecordFilter type*

Complete schema definition for CredentialDelegationService can be referred at

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/)[credential-delegation-service.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/credential-delegation-service.xsd)

### Administration

The CDS maintains a group of administrative users. Administrative users have the ability to monitor all delegated credentials, update the status of delegated credentials, and manage the group of administrators. The CDS web service interface allows administrators to add administrators, remove administrators, and obtain a list of administrators from the CDS. CDS administrators are identified by their Grid identity and must authenticate with the CDS using their Grid credentials to invoke administrative operations.

The CDS allows both clients and administrators to monitor delegated credentials. Clients may monitor only the credentials they delegated, where as administrators may monitor any credential delegated to the CDS.

Following are the operation provided by CredentialDelegationService to administer credential delegation. Details on each of these methods can be referred at section 3.6.3.

addAdmin(String gridIdentity)

getAdmins()

removeAdmin(String gridIdentity)

updateDelegatedCredentialStatus(DelegationIdentifier, DelegationStatus)

deleteDelegatedCredential(DelegationIdentifier)

searchDeligatedCredentialAuditLog(DelegatedCredentialAuditFilter)

### Data types

Following links provides WSDL and schema definitions for CredentialDelegationService.

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/)[CredentialDelegationService.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/CredentialDelegationService.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/)[DelegatedCredential.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/DelegatedCredential.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/)[WS-ResourceLifetime.wsdl](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/WS-ResourceLifetime.wsdl)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/)[credential-delegation-service.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/credential-delegation-service.xsd)

[https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1\_3\_release\_final/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/)[WS-ResourceLifetime.xsd](https://ncisvn.nci.nih.gov/svn/cagrid/tags/caGrid-1_3_release_1_3_0_1/cagrid-1-0/caGrid/projects/cds/schema/CredentialDelegationService/WS-ResourceLifetime.xsd)

### Error handling

Following are the fault codes returned by CredentialDelegationService.

CDSInternalFault: This fault code is returned if there is an internal error while processing a request.

*<element name="CDSInternalFault" type="tns:CDSInternalFault" />*

*<complexType name="CDSInternalFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

InvalidPolicyFault: This fault code is returned if a given policy is not valid.

*<element name="InvalidPolicyFault" type="tns:InvalidPolicyFault" />*

*<complexType name="InvalidPolicyFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

PermissionDeniedFault: This fault code is returned if a requestor doesn’t have access to data or operation.

*<element name="PermissionDeniedFault" type="tns:PermissionDeniedFault" />*

*<complexType name="PermissionDeniedFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

DelegationFault: This fault code is returned if there is an error with delegation data.

*<element name="DelegationFault" type="tns:DelegationFault" />*

*<complexType name="DelegationFault">*

*<complexContent>*

*<extension base="wsrbf:BaseFaultType" />*

*</complexContent>*

*</complexType>*

### Configuration

The CDS can be configured through a properties file, **CDS\_LOCATION/etc/cds.properties**. Below are the properties with description:

1. **gaards.cds.name** - Unique name of the CDS, distinguishing one CDS from another CDS on the same host. This will be used as the database name.
2. **gaards.cds.max.delegation.path.length** - The maximum delegation path length of credentials issued by the CDS. A value of 0 specifies that credentials issued by the CDS cannot be delegated.
3. **gaards.cds.db.host** - The host name of the CDS database.
4. **gaards.cds.db.port** - The port to connect to the CDS database on.
5. **gaards.cds.db.user** - The user id to use to connect to the CDS database.
6. **gaards.cds.db.password** - The password to use to connect to the CDS database.
7. **gaards.cds.dbkeymanager.key.encyption.password** - This property specifies a password which is used to encrypt the keys of the delegated credentials in the database. This property in needed for DB Key Manager.

### Data Model

**GROUP\_POLICIES:** Delegating a credential with a Group Delegation Policy enables the delegator to specify a Grid Grouper group such that the members of the group may have access to their credential. GROUP\_POLICIES table stores information related to a policy based on Grid Grouper.

**IDENTITY\_POLICIES:** Delegating a credential with Identity Policy enables a delegator to specify a Grid Identity of Delegatee. IDENTITY\_POLICIES table stores information related to a policy based on grid identity.

**DELEGATED\_CREDENTIALS:** This table stores information related to all credentials delegated to a delegate.

**KEY\_MANAGER:** Upon receiving the Initiation request the CDS will generate a key pair and store the information provided and the key pair in KEY\_MANAGER table. The key pair generated will be used to make up a delegated credential for the client making the request.

**KEY\_MANAGER\_CERTIFICATES:** Upon receiving the signed certificate from a delegator, CDS stores the certificate in KEY\_MANAGER\_CERTIFICATES. The signed certificate along with the generated private key make up a credential which can be delegated to other clients based on the delegation policy specified.

**

*Figure 76 CDS data model*

### Client API

Following diagram shows client API hierarchy for Credential Delegation service.

DelegationUserClient is responsible providing APIs to a delegator to delegate a credential, find delegated credentials, suspend delegated credentials, and search audit log for delegated credentials.

DelegationUserClient is responsible providing APIs to an Admin to delegate a credential, find delegated credentials, suspend delegated credentials, and to search audit logs for all users.

DelegatedCredentialUserClient is responsible providing APIs to a delegatee to get a delegated credential.



*Figure 77 CDS client API*

## WebSSO

Single sign-on (SSO) is a property of access control of multiple, related, but independent software systems. With this property a user logs in once and gains access to all systems without being prompted to log in again at each of them.

WebSSO provides a comprehensive single sign-on capability across multiple web applications using GAARDS. This facilitates user to provide credentials once and be able to navigate across multiple web applications without prompting user to present credentials again. WebSSO also establishes a grid session for the user, allowing them access to other grid services seamlessly.

WebSSO framework supports following features enabling single sign-on.

1. **Grid Single Sign-On**Allows a user to access a Grid service without prompting for credentials again after establishing a single sign-on once. When the user navigates to another web application under the single sign on umbrella, then user grid identity should be effectively delegated to that other application for seamless navigation.
2. **Parameter Passing**Share user specific data elements through application URL. This would enable SSO with web applications using GET request.
3. **Single Sign-Out**  
   Signing from one application would signout from all other applications accessed through Single sign-on. This would also terminate grid session.

### JA-SIG's CAS

CAS is at the core of WebSSO framework. CAS is an Enterprise Java solution to web application authentication that also provides the benefit of Single Sign On (SSO). Technically, SSO can be achieved because the authentication can be removed from the web application and handled centrally. And, when this authentication is handled by a single service, access to many services can be granted once and "remembered" for the life of the web session.

The CAS authentication and service registry is a secure way to authenticate users and provide web application or web service access. Authentication credentials are managed by the CAS server and application servers never touch this information. User access to your application is quite involved. A multi-request approach adds a layer of security and truly logical mechanism where authentication is isolated from the application.

CAS server component facilitates authentication of user's credentials and establishment of the Single Sign-On Session and granting the SSO Session Tickets.

CAS client side web agent includes classes for ticket validation, proxy ticket acquisition, servlets and filters for implementing the client portion of the CAS protocol and Assertion Java object for representing the results of a validation attempt. This library can be used for implementing custom CAS functionality and to enable CAS SSO for existing applications.

Following diagram shows the workflow of CAS SSO process.



*Figure 78 CAS workflow*

Accessing a login protected web application resource on a web container would be redirected to CAS server URL including application URL.

Example: https://cas\_server/cas/login?service=https://other\_server/application1

When the CAS server receives the request, CAS programmatically forms a new URL request (redirection) and calls application1 and adds a unique one-time-only random ticket (String) as a request parameter.

Example: https://other\_server/application1?ticket=ST-8670-123buTvFFjo980

The other server receives this request through a CAS client servlet filter that's been configured with application1. It parses the ticket and starts a new HTTPS connection with the CAS server. This new request is formed programmatically by the CAS filter Java code and it's commonly called the "service validation" step. It receives an SSL certificate after it sends a request to the CAS server. By protocol and Java standard, the certificate is compared by Java to its collection of trusted certificates. If the certificate is "trusted", the HTTPS communication will occur.

CAS receives this secure request and prompts the user for his password. This is where the CAS server authenticates the user that's requesting access to application1. Remember that all this is occurring and the user has only made a single request to access application1. CAS will also verify that application1 has been loaded into its persistent store of CAS registered services. It is, so CAS presents an HTML login screen to the user. The user enters his credentials and CAS will do the verification.

Once the user has been authenticated, the CAS server fulfills the application server's HTTPS CAS client request and returns an XML message of "success" along with the authenticated username. The original request exits the CAS filter and now allows application1 to serve content to this newly authenticated user.

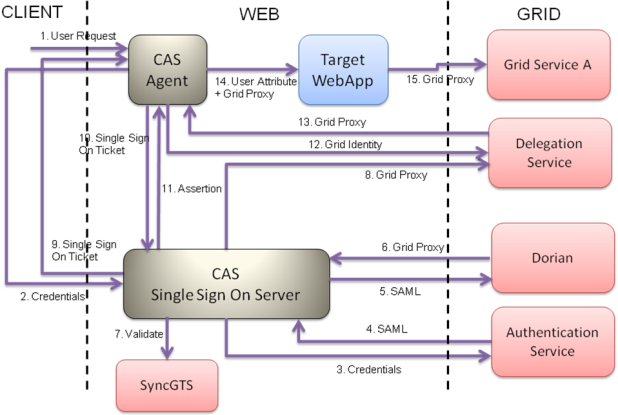
### WebSSO Workflow

The goal of WebSSO framework is to provide a seamless navigation across multiple WebSSO enabled web applications deployed on multiple web containers without forcing user to login multiple times for each web application. Following diagram shows high level architecture of WebSSO usage scenario. WebSSO enabled web applications running on different web containers would connect to single CAS server to verify user sign-on status and authentication. Detailed explanation of single sign-on workflow can be found in the following section.



*Figure 79 WebSSO architecture*

To suit caGrid needs with Single sign-on, WebSSO framework presents a solution that integrates CAS capabilities with GAARDS components. Following diagram shows different GAARDS components working together with CAS to provide a comprehensive Single sign-on solution to web applications running on multiple domains.



*Figure 80 WebSSO workflow*

Following is the workflow of interactions between different components to establish a single sign on session and allow a user to access a protected web application.

1. User initiates a call to the target application by typing the URL to the target application in the browser which is protected by the SSO Framework. This user request is intercepted by the CAS Agent which searches for an established session. Since it can't find it, it redirects the user's request to the CAS Server providing the target application's URL as return point.
2. The CAS Server displays a login page to the user. The user provides his local credentials in form username/ password pair on this login page for authentication.
3. The CAS server provides these credentials to the Authentication Service for the purpose of authentication.
4. If the credentials are valid, the Authentication Service returns a signed SAML Assertion back to the CAS Server.
5. The CAS server now passes this signed SAML assertion to Dorian.
6. Dorian makes sure that the SAML is signed by a registered Authentication Service and return a short term Grid Credentials (Grid Proxy) for the user.
7. They CAS server validates the Grid Proxy obtained from Dorian against the GTS to make sure that it is still valid and the CA has not be revoked.
8. Once validated, the CAS server obtains the list of Host Identities from the configuration files and formulates a delegation policy from it. It then publishes this delegation policy passing the user's Grid Credentials to the Delegation Service.
9. Now it formulates a service ticket and attaches it as part of the URL as a GET parameter. It redirects the user back to the application using the return URL provided
10. The CAS Agent sitting on the application again intercepts the request and retrieves the Service Ticket It now validates this Ticket against the CAS Server to make sure that the User is authenticated recently.
11. CAS Server validates the ticket and returns an assertion object back to the client which contains the user's details. The CAS Agent retrieves this information and attaches it to User Request/Session.
12. Now the CAS Agent connects to the Delegation Service using the Host Credentials of the Target Web Application to retrieve the User's Grid Credentials (Grid Proxy) by passing the user's Grid Identity.
13. The Delegation Server retrieves the Host Identity from the call and checks it against the policy published for that user. If the application has been given the delegation rights, it returns the Grid Proxy for that user back to the CAS Agent.
14. The CAS Agent attaches the Grid Proxy as an attribute to the Session and forwards the request to the target application
15. The Target Application can retrieve all user attributes from the Request/Session including the user's Grid Credentials. It then can use this Grid Credential to access a grid service on user's behalf.

WebSSO framework uses filter mechanism to intercept user requests and enforce single sign-on behavior. Following are the filters used by the framework.

**CAS Authentication Filter** - This is the entry for the CAS Authentication Filter. This filter checks if the User's Single Sign On Session has been established or not. This entry should be copied as it is in the client web application's web.xml file

**CAS Validation Filter** - This is the entry for the CAS Validation Filter. This filter validates the User's Authentication Session and then retrieves User's Attributes from the Central CAS Single Sign On Server. This entry should be copied as it is in the client web application's web.xml file

**WebSSO Attribute Loader Filter** - This is the entry for the WebSSO Attribute Loader Filter. This filter reads the attributes which are retrieved from the Central CAS Single Sign-On Server and loads each of them into the HTTP Session. This entry should be copied as it is in the client web application's web.xml file

**WebSSO Delegation Lookup Filter** - This is the entry for the WebSSO Delegation Lookup Filter. This filter connects to the Credential Delegation Service ([**CDS**](http://cagrid.org/display/cds/Home)) using the Client Application's Host Credentials and retrieves the User's Grid Credentials. It then stores the retrieved User's Grid Credentials into the session as a session attribute. This filter needs two parameters pointing to the certificate and the key file for the Client Application's Host Credential.

**certificate-file-path** - points to the certificate file for the Client Application's Host Credentials.

**key-file-path** - points to the key file for the Client Application's Host Credentials.

**WebSSO Logout Filter** - This is the entry for the WebSSO Logout Filter. This filter connects to the WebSSO Server and issues a logout call. On receiving this logout the CAS Server terminates the SSO Session and issues a call to the Credential Delegation Service ([**CDS**](http://cagrid.org/display/cds/Home)) to destroy user's delegation policy.

### Configuration

**WebSSO Client**

The WebSSO client is configured through a properties file: WEBSSO\_CLIENT\_LOCATION/build/cas-client.properties. Following are the properties defined.

* **cas.server.gateway** - This is a property indicating whether the login screen should be displayed to the user or not. This option should be always set to 'false'.
* **cas.server.renew** - This is a property if set to true would require the user to login again irrespective of whether the Single Sign On session has been established or not. For general use, this should be set to 'false'. **Note** If you want to log into another application within the SSO realm without providing your credentials again, this property should be set to 'true'
* **cas.server.url** - This is the URL to the WebSSO's CAS Server. **Note** The CAS Server is generally installed in a secured container as a result of which the URL should have 'https'.
* **cas.client.service** - This is the URL to the WebSSO client server.

**WebSSO Server**

The WebSSO server is configured through a properties file called websso-properties.xml. Following are the configuration properties.

**websso-server-information** - This section contains information about the WebSSO Server.

* **start-auto-syncgts** - This is a configuration parameter indicating whether the WebSSO Server should start SyncGTS automatically or not. "yes" indicates WebSSO Server to start the SyncGTS daemon.
* **host-credential-certificate-file-path** - This is the path to the WebSSO Server's Host Certificate File obtained in Step 3 above.
* **host-credential-key-file-path** - This is the path to the WebSSO Server's Host Key File obtained in Step 3 above.

**credential-delegation-service-information** - This section is used to configure the Credential Delegation Service which will be used to publish the delegation policy for User's Grid Credentials

* **service-url** - This is the URL to the Credential Delegation Service. This information is known based on the prior installation of CDS.
* **service-identity** - This is the service Identity for Credential Delegation Service.
* **delegation-lifetime-hours** - This is the hours for which the delegation policy remains alive.
* **delegation-lifetime-minutes** - This is the minutes for which the delegation policy remains alive.
* **delegation-lifetime-seconds** - This is the seconds for which the delegation policy remains alive.
* **issued-credential-path-length** - A path length specifies the length of a credential chain. For example a credential with a length of 2 means that the credential can be delegated to a second party and the second party could in turn delegate the credential to a third party at which point the third party can no longer delegate the credential. The Issued Credential Path Length specifies the path length of the credentials issued to third parties. An Issued Credential Path Length of 0 indicates that the third party may not further delegate the user's credential.

**dorian-services-information** -

**dorian-service-descriptor** - This section is used to configure the Dorian Server which will be used to retrieve User's Grid Credentials.

* **display-name** - This is the display name for the Dorian Service which is displayed on the login screen.
* **service-url** - This is the URL to the Dorian Service.
* **service-identity** - This is the service Identity for the Dorian Service.
* **proxy-lifetime-hours** - This is the hours for which the proxy remains alive.
* **proxy-lifetime-minutes** - This is the minutes for which the proxy remains alive.
* **proxy-lifetime-seconds** - This is the seconds for which the proxy remains alive.

**delegated-applications-group** - These are the group of applications to which user's credentials are to be delegated. In future these will be provided as a choice to the user. As of now they are just static list.

* **group-name** - This is the name given to the group of the applications to which a user's credentials is delegated.
* **delegated-application-list** - These are the list of the applications to which user's credentials are to be delegated.

**delegated-application** - This is the entry for an application to which the user's credentials are delegated..

* **application-name** - This is the name of the application to which the user's credentials are delegated.
* **host-identity** - This is the host identity (obtained from the Host Credentials that are obtained from the Dorian) of the application to which the user's credentials are delegated.

Following is the list of session attributes which hold values of the corresponding User Attributes:

|  |
| --- |
|  |
| |  |  | | --- | --- | | **Session Attribute Name** | **Information** | | CAGRID\_SSO\_GRID\_IDENTITY | User's Grid Identity | | CAGRID\_SSO\_FIRST\_NAME | User's First Name | | CAGRID\_SSO\_LAST\_NAME | User's Last Name | | CAGRID\_SSO\_EMAIL\_ID | User's Email Id | | CAGRID\_SSO\_DELEGATION\_SERVICE\_EPR | Delegation Service's End Point Reference | | CAGRID\_SSO\_GRID\_CREDENTIAL | User's Grid Credential | |

### Secure Communication

Communication between different components of WebSSO solution is secured using PKI infrastructure. A user access WebSSO enabled application through HTTPS. Web application hosting WebSSO client connects with Delegation Service securely with host credentials. A host credential consists of an X.509 certificate and private key. Dorian provides the ability to issue and manage host credentials. There are many methods of retrieving host credentials, including:

1. Requesting a credential from a known/trusted certificate authority (caGrid Certificate Authority).
2. Standing up a Dorian service.
3. Standing up a simple certificate authority.

Once a host credentials are obtained and stored on the server, it can be used to communicate with Delegation Service securely.

WebSSO Server running in a web container is enabled with SSL for secure communication. In order for the client to trust the WebSSO Server, its Public CA Cert should be installed into WebSSO client container trust store.

## Auditing

caGrid Event management API provides auditing capability to GAARDS infrastructure. Event Management is one of caGrid tools providing APIs to record events. Events generated by the security framework are logged into a database table with following columns.

EVENT\_ID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

TARGET\_ID VARCHAR(255) NOT NULL,

REPORTING\_PARTY\_ID VARCHAR(255) NOT NULL,

EVENT\_TYPE VARCHAR(50) NOT NULL

OCCURRED\_AT BIGINT NOT NULL,

MESSAGE TEXT NOT NULL

### Dorain

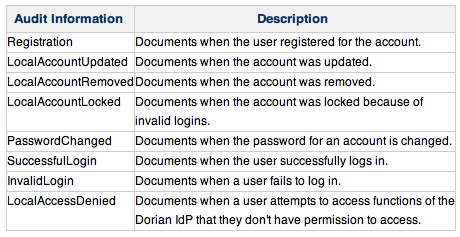
For security purposes and to give administrators insight on all aspects of Dorian, Dorian maintains a list of auditing information for grid user account, user certificates, host certificates, trusted identity providers, and general system information. Dorian allows administrators to perform and audit across all federation auditing information maintained by the system. The following is a list of auditing information maintained for each user account:

|  |  |
| --- | --- |
| **Audit Information** | **Description** |
| AccessDenied | Documents when a user was denied access to Dorian. |
| AccountCreated | Documents when a grid user account was first created. |
| AccountRemoved | Documents when the caGrid user account was removed. |
| AccountUpdated | Documents when the caGrid user account was updated. |
| AdminAdded | Documents when a user was granted administrative access to Dorian. |
| AdminRemoved | Documents when a user was revoked administrative access to Dorian. |
| CRLPublished | Documents when Dorian publishes it CRL and who it publishes it to. |
| HostCertificateApproved | Documents when and by whom a host certificate was approved. |
| HostCertificateRequested | Documents when and by whom a host certificate was requested. |
| HostCertificateRenewed | Documents when and by whom a host certificate was renewed. |
| HostCertificateUpdated | Documents when, what, and by whom the host certificate was updated. |
| IdPAdded | Documents when an identity provider was registered to Dorian as a trusted identity provider. |
| IdPUpdated | Documents when an identity provider was updated. |
| IdPRemoved | Documents when an identity provider was removed from Dorian as a trusted identity provider. |
| InternalError | Documents when an unexpected system error occurs in Dorian. |
| InvalidUserCertificateRequest | Documents when a user FAILED to obtain PKI user credentials. |
| SuccessfulUserCertificateRequest | Documents when a user was able to successfully obtain PKI user credentials. |
| SystemStartup | Documents each time the Dorian Service is started up. |
| UserCertificateUpdate | Documents when an individual user certificate is updated. |
| UserCertificateRemoved | Documents when an individual user certificate is removed. |

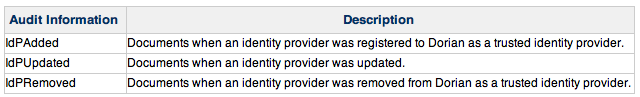
The GAARDS UI allows Dorian administrators to search the auditing information collected by Dorian using the following search criteria.

|  |  |
| --- | --- |
| **Criteria** | **Description** |
| Target | The identity of the subject that the audit information describes. |
| Reporting Party | The identity of the party that performed or reported the action. |
| Audit Type | The type of auditing information, please consult the table above for different types. |
| Start Date | The start of a date/time range of when the even occurred. |
| End Date | The end of a date/time range of when the even occurred. |
| Message | Search the content of the Audit Message. |

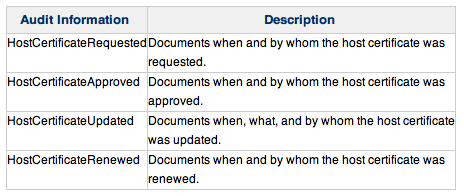
Dorian maintains auditing information for each Dorian Identity Provider users accounts. Dorian Identity Provider administrator may search/view the auditing information for each account.Dorian enables Dorian Identity Provider administrators to perform and audit across all the Dorian IdP auditing information.



Dorian maintains the following auditing information for trusted identity providers.



Dorian maintains the following auditing information for host certificates



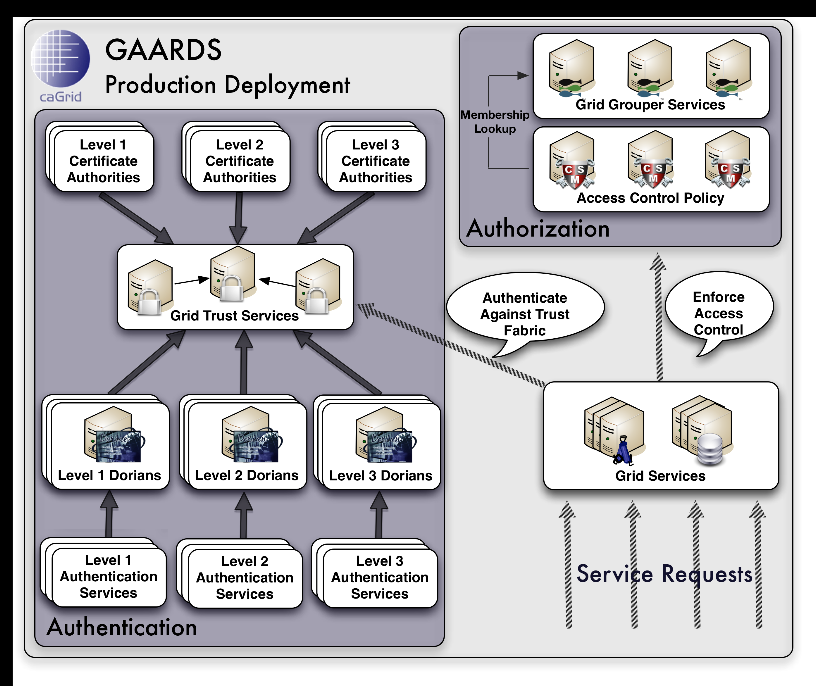
### CDS

Credential Delegation Service uses Event Management API to store audit information about credential being delegated. It stores delegation identifier, grid identity of the user, delegation event and message. Following are the event types generated by CDS.

|  |
| --- |
| **Event** |
| DelegationInitiated |
| DelegationApproved |
| DelegationStatusUpdated |
| DelegatedCredentialIssued |
| DelegatedCredentialAccessDenied |

# Deployment

Following is the high-level GAARDS Production Deployment methodology diagram. As it is shown in the diagram, GAARDS deployment is flexible to support different levels of assurance defined by NIST. A level of assurance or trust level specifies the level of confidence with which a given certificate authority is trusted in the grid in which it is deployed. Multiple Dorian instances and Authentication Services supporting different level of assurance can be deployed with each level of assurance registered to GTS.



*Figure 81 GAARD Production Deployment*

Decoupled nature of GAARDS framework let a user to deploy and use its components independently. It is not necessary to deploy all GAARDS components to use any one of its components. The dependency between GAARDS components is as given below.



*Figure 82 GAARDS dependencies*

As shown in the diagram above, WebSSO has dependency on AuthenticationService, Dorian and Credential Delegation service. As it is explained in [section 3.7](#_WebSSO), WebSSO component enables single sign-on between different Web applications. It uses AuthenticationService to authenticate a user and get a SAML assertion. WebSSO uses Dorian to get grid proxy certificate using SAML assertion got from AuthenticationService. WebSSO uses Credential Delegation service to delegate proxy credential to enable single sign-on behavior.

Credential Delegation service has dependency on GridGrouper to authorize group based delegation policies. Dependency can be ignored if there is no need for group based delegation policy. Credential Delegation service can omitted if delegation is not required, or in-container Globus delegation suffices, or not using security.

AuthenticationService, as it is explained in [section 3.2](#_Authentication_Service), used to integrate an existing authentication mechanism with Dorian. This component can be omitted if Only using Dorian’s internal user management, or manually managing End Entity Certificates, or not using security.

As it is explained in [section 3.3](#_Dorian), Dorian provides federated identity management for caGrid. Dorian provides a secure mechanism to issue and provision credentials for Web/Grid services. Dorian can be omitted if the setup is manually managing End Entity Certificates or not using security.

Grid Trust service, the trust fabric between caGrid participants, is a federated solution for registering and managing certificate authority certificates and CRLs, facilitating the enforcement of the most recent trust agreements, manage LOA, certificate verification and validation. As it is explained in [section 3.5](#_Grid_Trust_Service), GTS communicates with other GTS instances configured per trust federation. GTS can be omitted if the setup is manually managing trusted CAs and CRLs or not using security.

GridGrouper is a group/virtual organization management solution to support group-based authorization. It can be omitted if the setup is not using group/role/attribute based authorization, or have no need to manage groups.

Please refer to following link for a grid service deployment procedure.

<http://wiki.cagrid.org/display/knowledgebase/Grid+Service+Deployment+Guide>

# References

<http://cagrid.org/display/knowledgebase/caGrid+1.3+Technical+Overview>

<http://cagrid.org/display/gaards/Home>

<http://cagrid.org/display/documentation/Home>

<http://cagrid.org/display/dorian12/Design+Guide>

<http://cagrid.org/display/authenticationservice13/Design>

<http://cagrid.org/display/cds13/Design>

<http://cagrid.org/display/gridgrouper12/Design+Guide>

<http://cagrid.org/display/gts12/Design+Guide>

<http://cagrid.org/display/websso12/Design>

<https://spaces.internet2.edu/pages/viewpage.action?pageId=10059842>

<http://wiki.oasis-open.org/security>

<http://www.jasig.org/cas>

<http://www.globus.org/toolkit/>

<http://www.globus.org/security/overview.html>