

IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

Technical Specification for Authorization Service

Author(s):	Marcin Adamski, Michal Chmielewski, Sergiusz Fonrobert, Jarek Nabrzyski, Tomasz Nowocień, Tomasz Ostwald
Document Filename:	
Work package:	WP6 Security
Partner(s):	PSNC, MPG, MU, ISUFI
Lead Partner:	Poznan Supercomputing and Networking Center
Config ID:	GridLab-6-D6.2-0004-0.9
Document classification:	INTERNAL

Abstract: In this document, the description of initial architecture for the Authorization Service developed as a part of WP6 of the GridLab project is presented. The document contains the general overview of the concept, initial design details for selected components of the system and examples of its application (*Scenarios*). The architecture is presented using UML-based notations. This deliverable will be further evaluated and the final version or annexes will be delivered by the 1st EC review.



Version: 1.0

Updated: August 12th 2002



Contents

1	Introduction	3
1.1	Purpose of Document	3
1.2	Structure of Document	3
1.3	Status of Document	3
2	Requirements Specification	3
3	Authorization Service Architecture	4
3.1	Architecture Overview	4
3.2	Use case Diagrams	6
3.2.1	High-Level Use Case Diagram	6
3.2.2	System Administrator Use Case Diagram	7
3.2.3	Security Policy Management Use Case Diagram	7
3.2.4	Using System Use Case Diagram	8
4	AS Architecture Details	9
4.1	Packages Diagrams	9
4.1.1	Core Component Diagram	9
4.1.2	Authorization and Security Policy Engine	11
4.1.3	User/Application Interface	13
4.2	Collaboration Diagrams	13
4.2.1	Administrator Collaboration Diagram	14
4.2.2	Security Policy Manager Collaboration Diagram	15
4.2.3	User Collaboration Diagram	16
4.3	AS Client Class Diagram	16
4.4	Security Policy Database Diagram	17
5	Authorization Scenarios	17
5.1	Sample Scenarios	19
5.1.1	The CAS Scenario	20
5.1.2	Simple Super—Scheduler Scenario	22
5.1.3	More Complex Super—Scheduler Scenario	26
5.1.4	User to User Asynchronous Collaboration Scenario	27



5.1.5	User to User Synchronous Collaboration Scenario	29
5.1.6	Scenarios Summary and proposition of scenario file format	32
6	Summary	33

1 Introduction

1.1 Purpose of Document

This document contains the initial version of the technical specification for the Authorization Service, which is developed in the Security Workpackage (no. 6) of the GridLab project. As this document refers to the research activities, this specification will evolve along with the progress of the project. The final specification for the Authorization Service will be provided as a separate document.

1.2 Structure of Document

The document is divided into 6 general sections. In section 1, the general document structure and its goals are presented. Section 2 is dedicated to some of the requirements for the Authorization Service.

In section 3, the general Authorization Service Architecture is presented with its overview, Use Case Diagrams.

In section 4, the details of AS Architecture are presented with appropriate Packages, Sequence and sample Class Diagrams.

In section 5, for Authorization Service are defined with some examples.

The summary and final notes are presented in section 6.

1.3 Status of Document

This document is a working draft and refers to the current state of the project. As a working draft, this specification may be updated, replaced, or made obsolete at any time. It is distributed for discussion purposes only, and should not be used as a reference.

2 Requirements Specification

The general requirements for Security Workpackage 6 in the GridLab project have been presented in the first deliverable, *WP6 Security: Initial Requirements* [1]. In section 3.1 some specific requirements assumed for the development of Authorization Service are described.

One general requirement that should be specified at this point is the general dependency of Authorization Service on general security architecture. For example, the process of authorization is performed after appropriate authentication of all parties involved, and all data are transferred using secured communication channels.

Throughout this document it is assumed that the authorization process is performed on the basis of sufficient security architecture.

3 Authorization Service Architecture

In the [1], the need for effective *Authorization Service* (referred as AS throughout this document) has been specified as one of the most significant requirements for security in the grid environments. The goal of the Security Workpackage is to develop such a service and introduce it to the GridLab testbed [7].

3.1 Architecture Overview

As specified (for example in [5]), the problem of security in grid environments is both significant and difficult. One of the main current problems is the lack of the appropriate technical solutions that would be able to fulfill high security requirements in grid environments. It results mainly from the general complexity of securing open network environments. However, it is also meaningful that grid technologies are relatively new and solutions to many core problems have not been found yet [4].

Therefore, as the general field of grid security lacks sufficient standards or often even specifications, any security service, in order to be applicable in practice, must be able to cooperate with various solutions. Additionally there should exist a possibility of using this service with solutions and standards that are not available today. This was probably the most significant motivation (as well as requirement) for the architecture of the Authorization Service. The general view at this architecture is presented on figure 1.

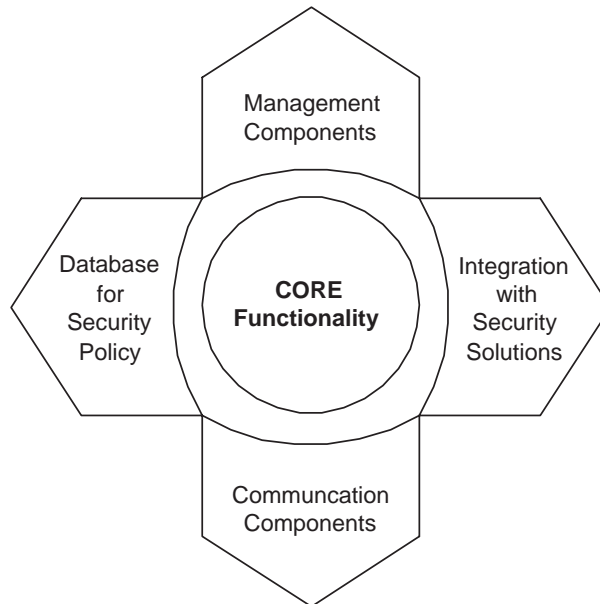


Figure 1: The general overview of authorization service architecture

While designing the architecture the following assumptions have been undertaken:

- A generic core functionality component should be distinguished, covering all issues connected with defining the security policy.

- The general architecture should be designed with modularity in mind; the core component should be able to use various components enabling the cooperation with various solutions.
- The following interfaces for such modules have been specified:
 - *Management Interface* - for actual managing the security policy stored in AS.
 - *Database Interface* - for storing data used in the security policy, for example resources, users or users groups definitions.
 - *User/Application Interface* - for users and applications to contact AS server with requests for authorization assertions. The final version of AS should be able to support various protocols for authorization, such as for example SAML [8], [10].
 - *CA Interface* - for communication with various Certification Authorities, used in the authentication process.

The more detailed view at this architecture is shown on the System Deployment Diagram (figure 2).

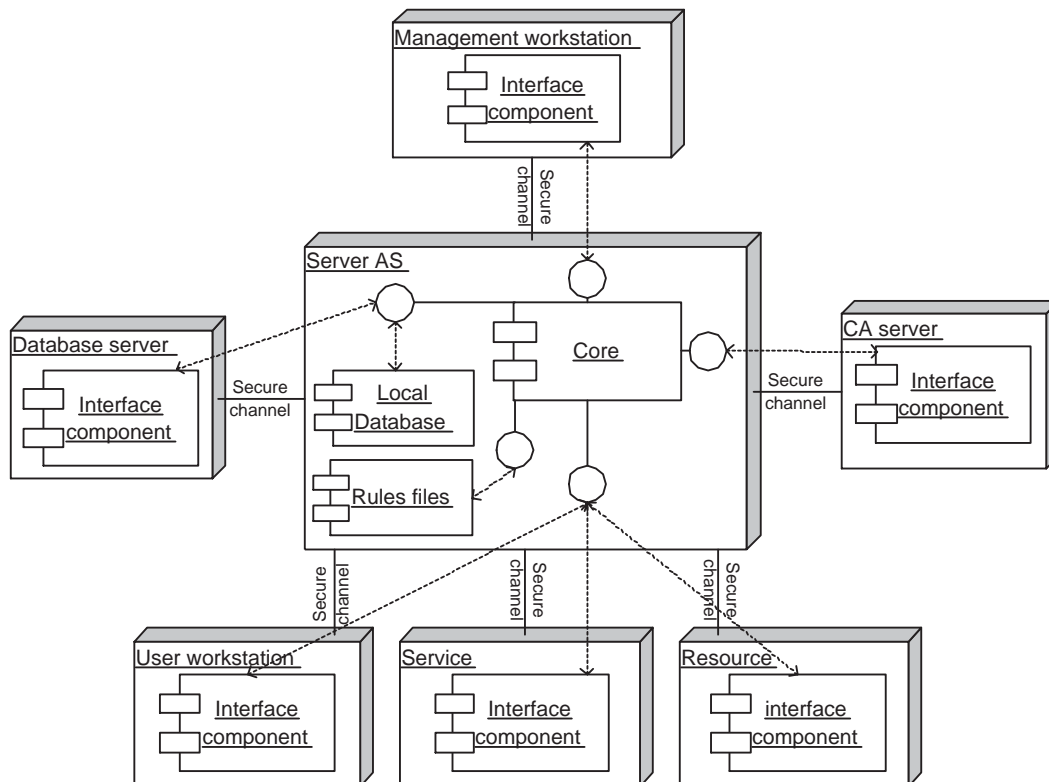


Figure 2: The general system deployment diagram

On figure 2 relations between AS Server and all external nodes are presented. The detailed internal architecture for the AS Server are discussed in further sections of this document.

The *Management Workstation*, used for AS Server administration, can either be installed on the same machine as the server or on an external one. The same refers to the *Database Server*,

which is used for storing information used in security policy such as objects, attributes, etc. (see 4.4).

As the AS Server is designed to be used in grid environments with various authorization protocols, the AS Server will be able to use different communication components (4.1.3) for users, resources services and other security components (like CAs). In order to enable using AS in various authorization models (based on *push* or *pull* approaches), the concept of the *Authorization Scenarios* is introduced (5). These scenarios are stored in the *Scenario Rules* files, which are located on the AS Server.

All components of the AS Server will be described in this document.

3.2 Use case Diagrams

Below, several Use Case Diagrams ([2], [6], [11]) describing the general functionality of AS and predefined users types for this system are depicted. The diagrams form a tree with the *High-Level Use Case Diagram* (figure 3) as a root. All the remaining diagrams, i.e. *System Management Use Case Diagram* (figure 4) and *Security Policy Management Use Case Diagram* (figure 5) are children.

3.2.1 High-Level Use Case Diagram

Three main actors can be distinguished in the AS System: *Administrator*, *Security Policy Manager* and *User*. The Administrator is generally responsible for the actual maintenance of the AS Server itself (3.2.2). The Security Policy Manager defines security policy, which is stored in the specific AS system (3.2.3). The last case refers to using the system by the user who wants to obtain authorization assertion (3.2.4).

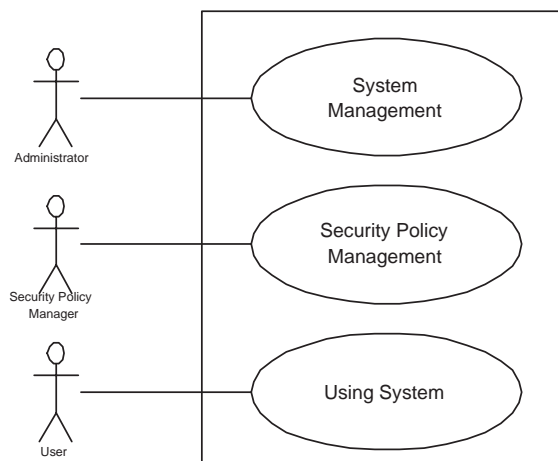


Figure 3: High-level use case diagram with distinguished main actors

3.2.2 System Administrator Use Case Diagram

The role of the Administrator is to maintain the AS system. He is therefore responsible for starting, stopping and configuring the AS server. He is also responsible for the security level of the base operating system, which should be as high as possible.

The configuration of the AS Server by the Administrator covers the following tasks:

- Appropriate configuration of AS software in the network environment.
- Gathering local requirements for using the authorization service and available security solutions.
- Implementing AS using predefined *Scenario* (5) or define its own, fulfilling the specific requirements.

The role of the Administrator does not cover creating or defining the security policy. This is a responsibility of the Security Policy Manager (3.2.3). In proposed architecture, roles of these two actors are disjointed. The general use case diagram for the Administrator can be referred to on figure 4.

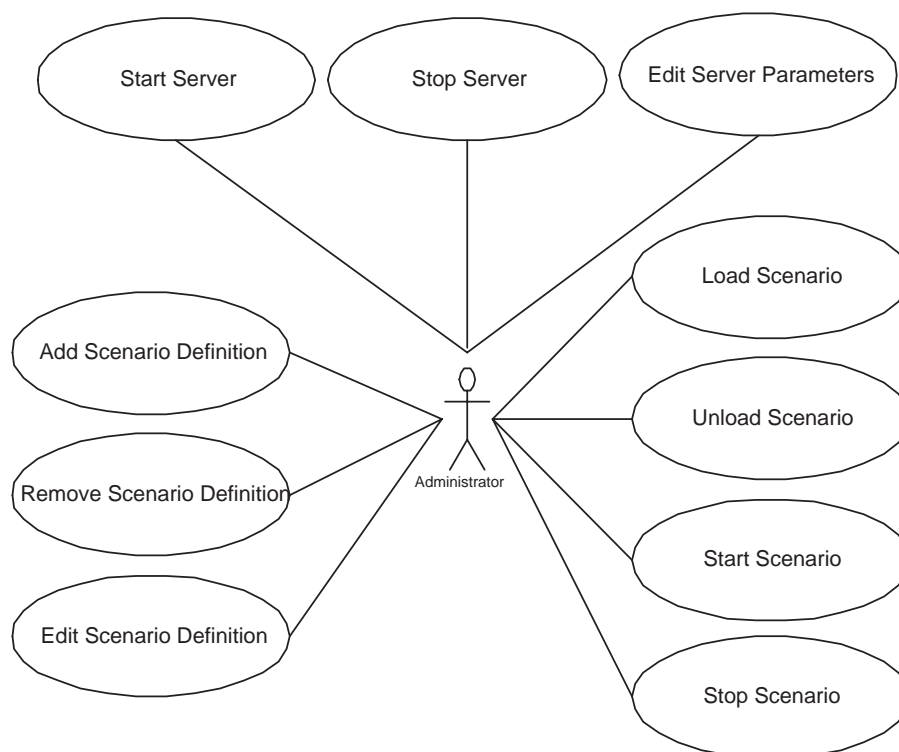


Figure 4: Use case diagram for AS system administrator

3.2.3 Security Policy Management Use Case Diagram

The role of the Security Policy Manager focuses on defining the actual security policy, i.e. assigning specific access rights for user or group of users in reference to the specific object, for

example resource or piece of data. The tasks of this role cover:

- Adding objects (and subjects) such as users, resources, pieces of data, and defining groups of objects.
- Defining security policy (granting subjects access rights to objects).
- Granting rights to defining some parts of security policy, for example users may be granted privilege to modify access rights for their data or resources.

As it was stated in the previous section, this role is strictly separated from the AS administrator. The Security Policy Manager defines the specific security policy, which is stored in AS managed by the Administrator.

The general use case diagram for Security Policy Manager is presented on figure 5.

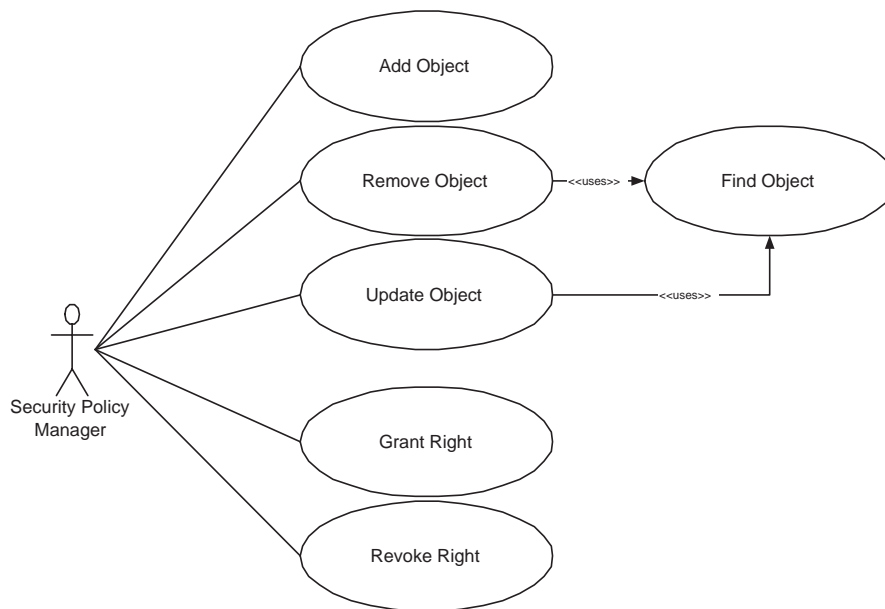


Figure 5: Use case diagram for Security Policy Manager

3.2.4 Using System Use Case Diagram

The role of the User refers to the most common case of using AS, i.e. requesting authorization assertion or granting access rights to some data or resources (if authorized). The role of the User can be performed by any authenticated entity, a person, a service or a resource, depending upon the applied security model.

The general use case diagram for the User is shown on figure 6. The action of requesting assertion was separated from defining policy in order to keep the possibility of placing additional restriction for this second type of operations.

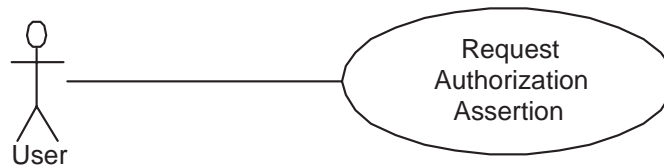


Figure 6: Use case diagram for User (typical using of AS)

4 AS Architecture Details

The details of the Authorization Service architecture are introduced in the sections below.

4.1 Packages Diagrams

On figure 7, the package diagram ([2], [6], [11]) of the Authorization Service architecture is presented. This figure shows direct relationships between the AS Server and other external components.

There are five types of interfaces defined:

- Management Interface,
- Databas Interface,
- CA Interface,
- Communication Interface for user, services and resources (with support of various modules),
- I/O Scenarios Files - additional interface, comparing to the figure 1.

4.1.1 Core Component Diagram

One of the main motivations for developing AS is to create flexible and efficient service, which could be used in various environments. The *AS Core* package is the main component responsible for internal representation and management of the security policy. This component is to be platform and system independent, while other one (for example the *Server Daemon* will depend on environment.

Additionally, as different security (and authorization) models are applied in different environments, AS should not support only one fixed scenario, but allow choosing the appropriate one. This is the reason why the AS Core component uses dynamically loadable scenarios. Its general design is shown on figure 8.

There are three module in the AS Core:

- Communication Component (CC),
- AS Scenario Engine (ASSE),

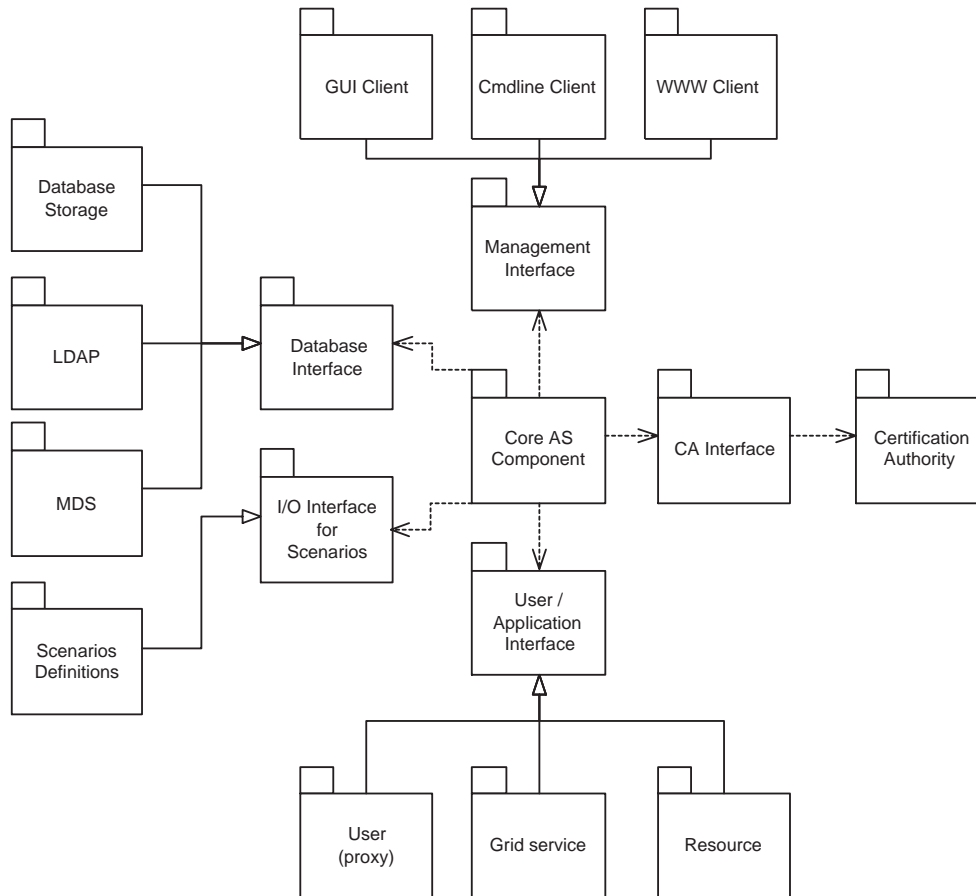


Figure 7: The package diagram of Authorization Service architecture

- Authorization and Security Policy Engine (ASPE).

The Communication Component is responsible for internal as well as external communication of the AS Server. Its main functionality can be divided into three sections:

- communication with external entities,
- communication between ASPE and ASSE,
- executing triggers (such like object *user* destroy).

The AS Scenario Engine (ASSE) is a component responsible for working with defined Authorization Scenarios. This component parse the scenario rules file describing the specific authorization model during AS initialization. Every use request is analyzed by the ASSE in reference to the specific scenario, i.e. to the authorization model used in the environment (and loaded in the AS Server). Then, the request in the formal structure of subject (or subjects) performing action (or actions) on object (or objects) is send to the ASPE, which makes actual authorization decision.

The operating of the ASSE may be divided into the following functions:

- Loading scenario file by using I/O rules file interface,

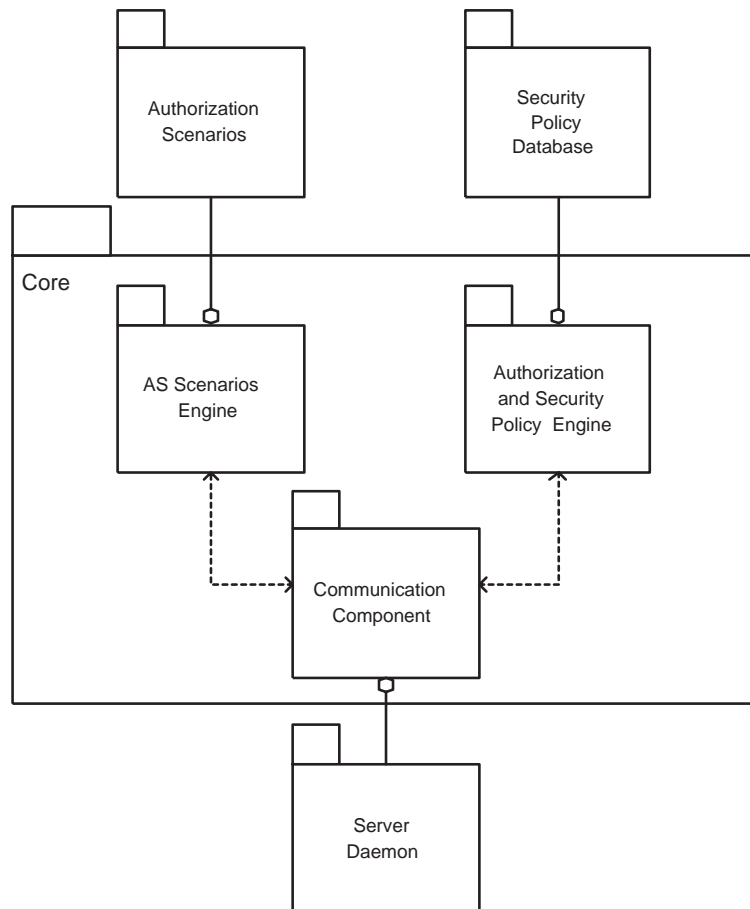


Figure 8: The AS Core component packages diagram

- Unloading scenario file by using I/O rules file interface,
- Start and Stop scenario,
- Verifying integrity of the scenario,
- Analyzing of user's request in reference to applied scenario.

The application of the Authorization Scenarios is described in section 5.

4.1.2 Authorization and Security Policy Engine

The general design of the Authorization and Security Policy Engine (ASPE) is depicted on figure 9. This module is responsible for all operations on the Security Policy Database. From functionality point of view, this module is divided into two components: the *Authorization Module* and the *Security Policy Manager*. However, from the design point of view, the four following packages of the ASPE can be distinguished:

- ASPE Interface,

- Authorization Module,
- Security Policy Manager,
- Security Policy Database Component.

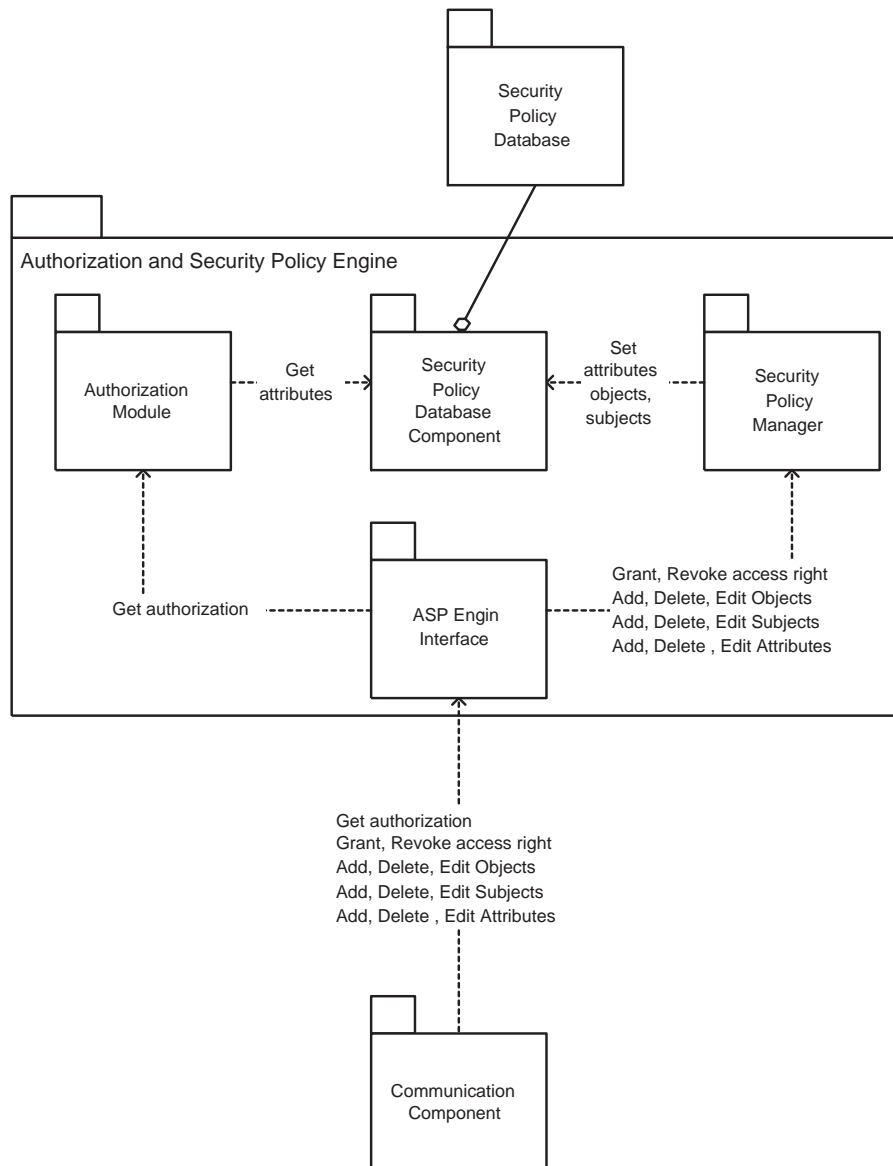


Figure 9: The Authorization and Security Policy Engine package diagram

As the ASPE is one of the Core packages, it has to communicate with other core packages using the Communication Component. Every message (like user's request in the context of specific scenario) arrives at the ASPE through *ASPE Interface*, which decides where to forward the message. If the request implies read-only operations, like a typical authorization request, the message is sent to Authorization Module. If the request implies write operations (like editing security policy database), the message is sent to Security Policy Manager.

The main function of Authorization Module is to make authorization decision upon user request, defined security policy and applied authorization scenario. However, the functionality of AM may be extended in the future. In the initial version, the AM will accept only structure with *Subject*, *Object* and *Action* as an input and return binary value with some possible message.

The Security Policy Manager package is responsible for operations related with editing security policy database. This module translates edit database actions for set of binary changes into database. Such changes can modify subjects, objects and attributes.

All messages that comes to ASPE (going both through AM and SPM) initiate appropriate operations in the *Security Policy Database Component*. This component is responsible for handling all database operations. The security policy database contains security policy definitions with elements such as subjects, objects, and their attributes. Attributes of subjects and objects are used to store information about access rights, descriptions, validations times etc. The security policy definition consist of qualifications access right for subjects and objects.

4.1.3 User/Application Interface

On figure 10, the packaged diagram for User/Application Interface is presented. This component is responsible for handling all external requests from users, services and resources. This component also has to be technology independent and be able to accept requests in various protocols, for example SAML. Every received request will be translated from supported language into internal AS representation.

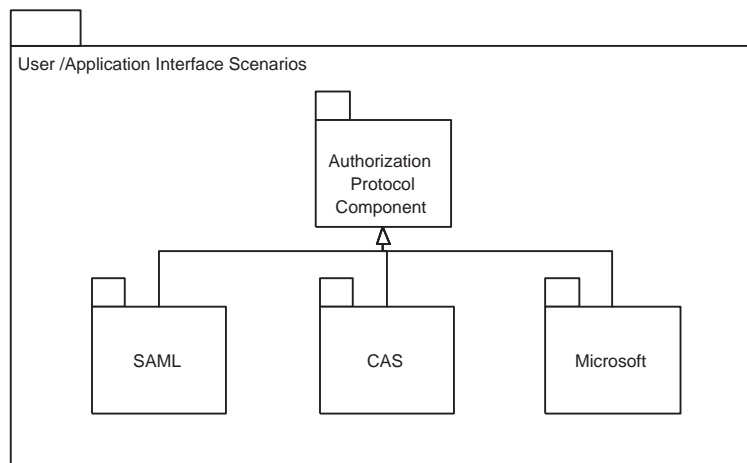


Figure 10: The user/application interface package diagram

4.2 Collaboration Diagrams

The collaboration diagrams ([2], [6], [11]) for interactions between actors introduced on figure 3 and the AS Server are introduced and commented in this section.

4.2.1 Administrator Collaboration Diagram

On figure 11, the sequence diagram of actions performed by the Administrator (3.2.2) is shown. The administrator communicates with the AS Server using the Management Interface and command line client. As the AS Server administration (without security policy!) is expected to be performed from the selected workstations only (server console is recommended), no additional UI clients are provided. Basic operations like *Edit Server Parameter File* or *Edit Scenario Rules File* are performed using the standard system editor. The integrity of the *Scenario Rules File* is verified when a file is loaded into the AS Server.

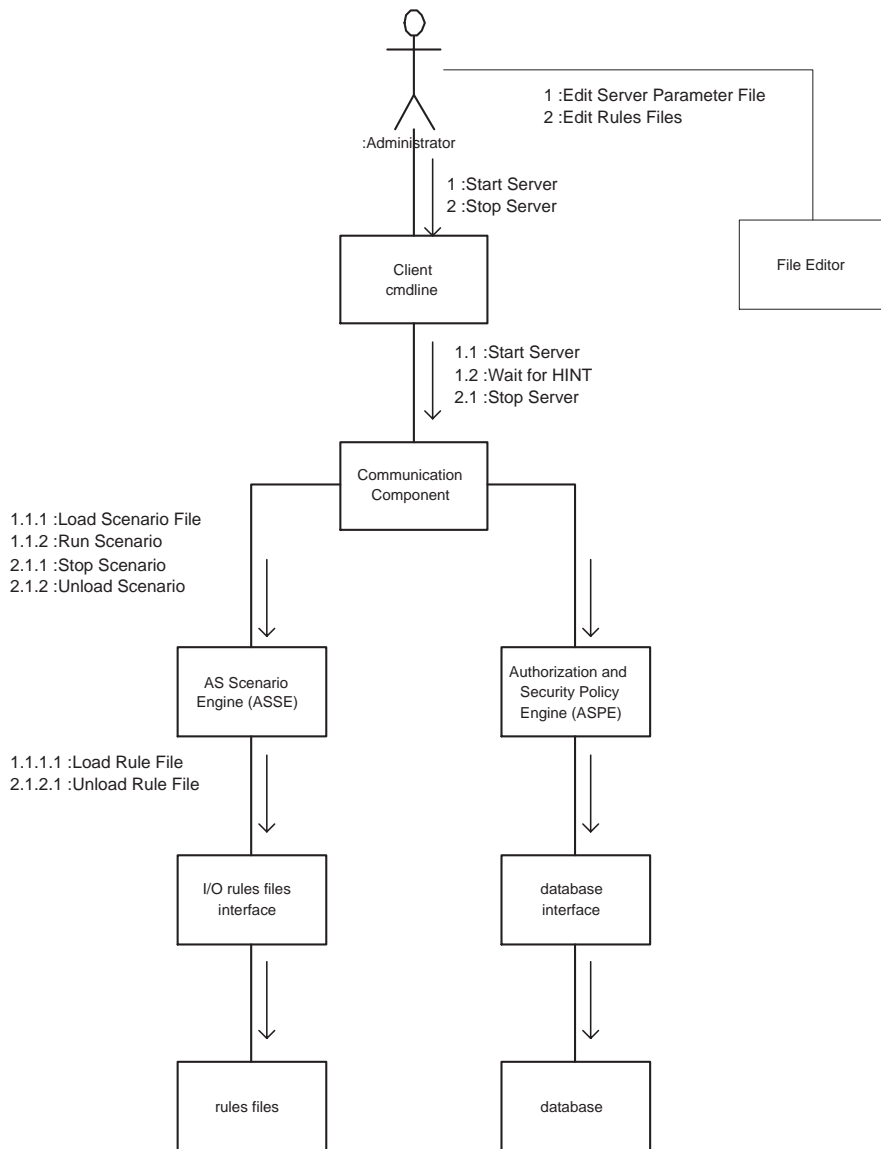


Figure 11: The collaboration diagram of actions performed by Administrator

4.2.2 Security Policy Manager Collaboration Diagram

On figure 12, the collaboration diagram of actions performed by the Security Policy Manager (3.2.3) can be referred to. This actor is responsible for editing the security policy database and all operations are performed through the ASPE component. Please note that the role of the Security Policy Manager is often performed by a user, who defines the security policy for his own data. This actor also plays a significant role in defining and maintaining collaborative groups.

The Security Policy Manager communicates with the AS Server using the Management Interface and various types of UI clients, depending on developed modules. Apart from the standard command line client, also GUI as well as WWW clients could be provided.

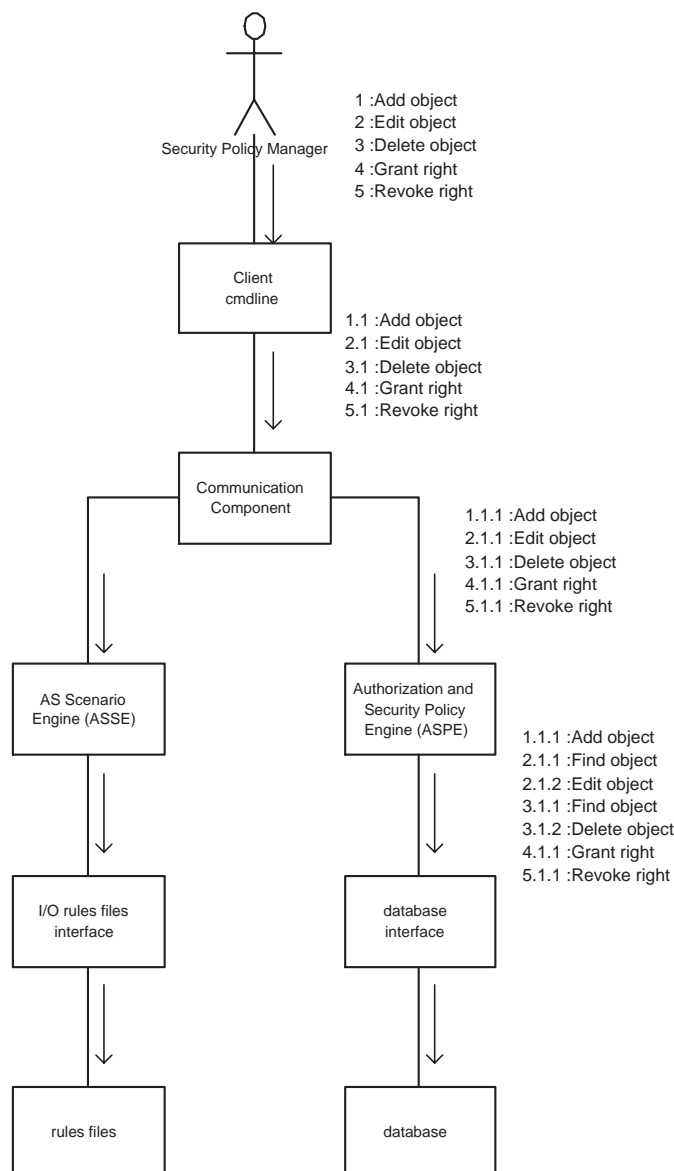


Figure 12: The collaboration diagram of actions performed by the Security Policy Manager

4.2.3 User Collaboration Diagram

On figure 13, the collaboration diagram of actions performed by User is shown. In this document, a User is defined as any person, service or resource, which request authorization assertion from the AS.

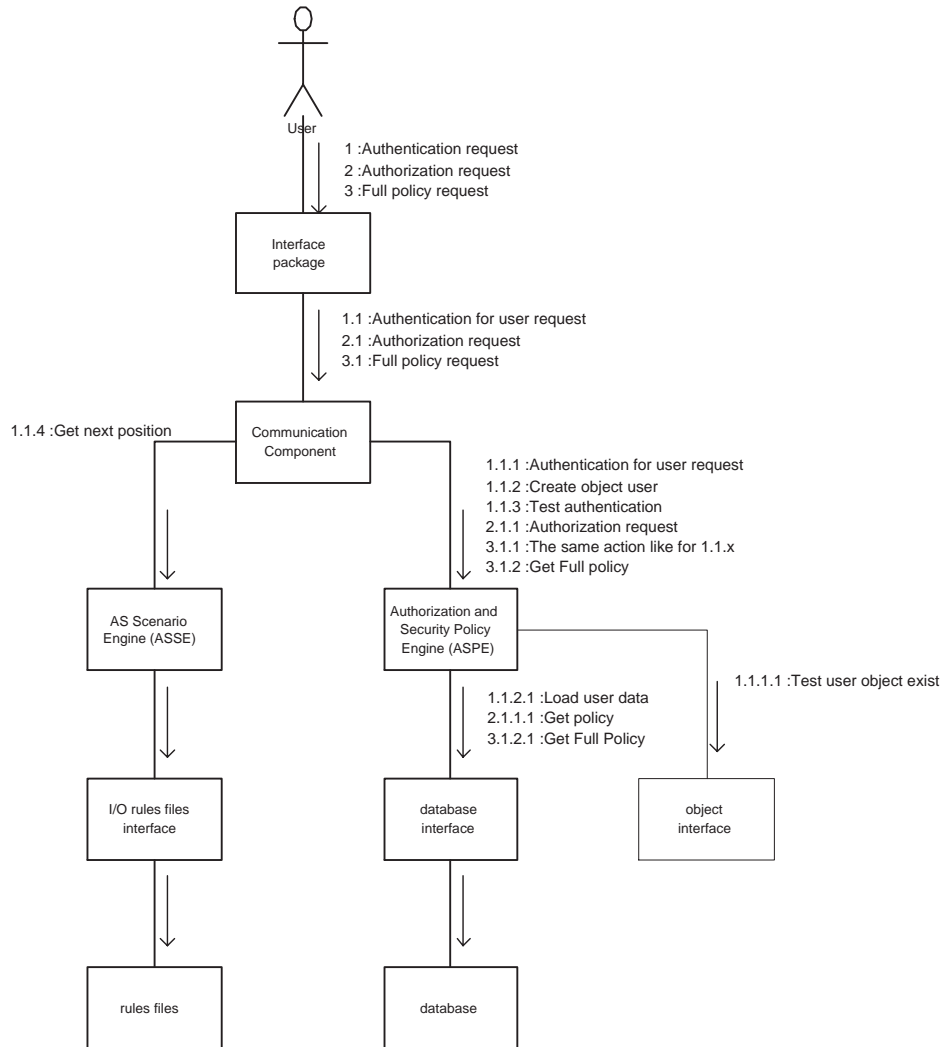


Figure 13: The collaboration diagram of actions performed by User

4.3 AS Client Class Diagram

On figure 14, the class diagram of standard users of AS is depicted. Please note that this diagram is only an expression of initial ideas for classes look in the AS. This diagram also contains only the selected objects - the full version of this diagram, as well as other class diagrams will be provided in future.

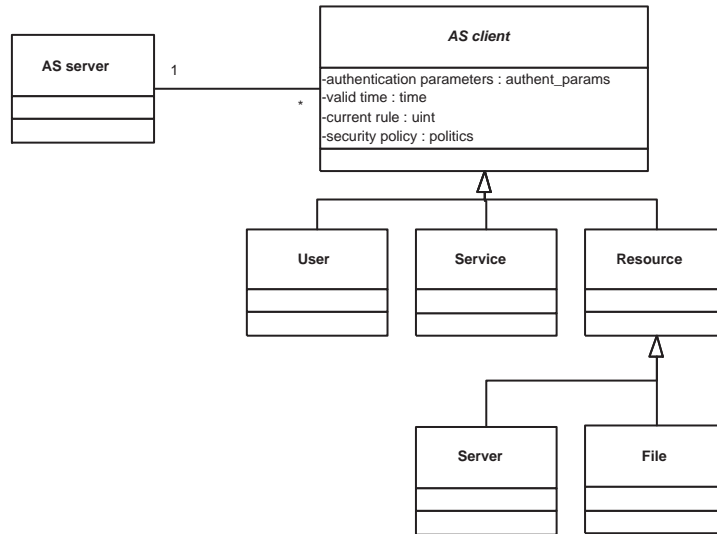


Figure 14: The class diagram of standard user of AS system

4.4 Security Policy Database Diagram

In the AS system there will be a possibility to store the security policy database locally on the AS Server or on a remote database server. The type of database is strictly specified, but will be implemented as a module to enable storing the security policy for example in MDS or LDAP. In the initial version of AS, the security policy database will be stored on the AS Server.

The structures of the database are not designed at the moment and will be prepared after further research. On figure 15 the UML structure of the Security Policy Database is depicted. Object associations between subjects with attributes and objects with attributes can be determined. These associations are n to n , which means that one subject or object can be connected with many attributes (it contains list of attributes) and one attribute can be connected to more than one subject or object.

A possible transformation of such structure to database tables is presented on figure 16. This figure presents five tables for Subjects, Objects, Attributes and two tables joining Subjects with Attributes and Objects with Attributes. Please assume that this structure may significantly change in the future.

Please note that this structure has to be extended with capabilities of grouping subjects, objects and attributes. Probably a hierarchical tree structure of subjects and objects will also be required.

5 Authorization Scenarios

As the architecture of the GridLab project is not fully specified, it is rather impossible to determine, which security model or models will be applied for the final version of the project. Surely, there will be no single scenario, which would be able to cover all the GridLab security requirements. However, it is very important to find a solution that will be as satisfactory as

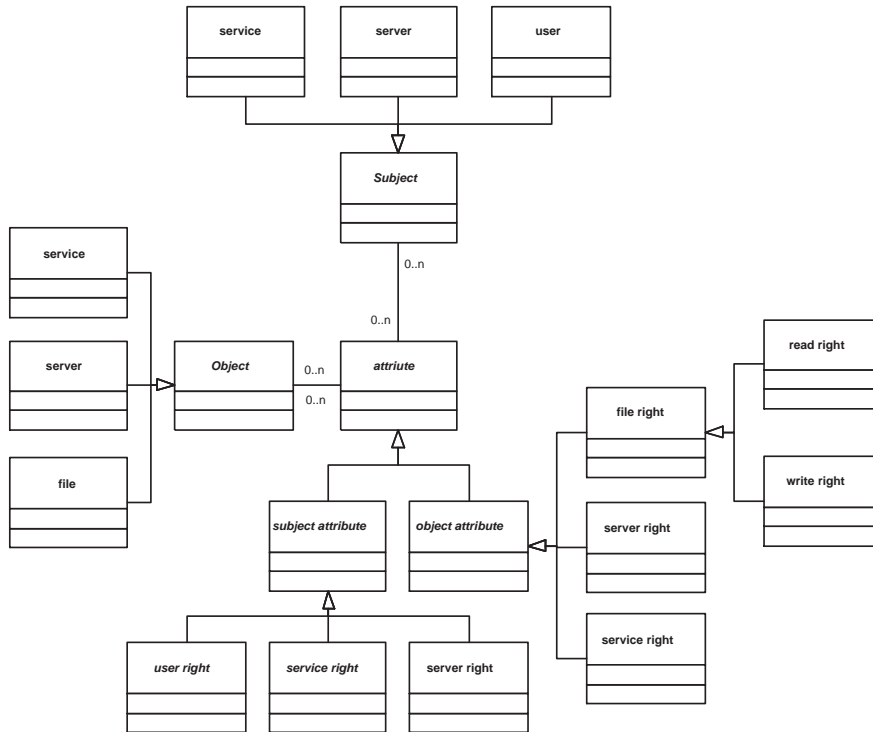


Figure 15: The general database diagram

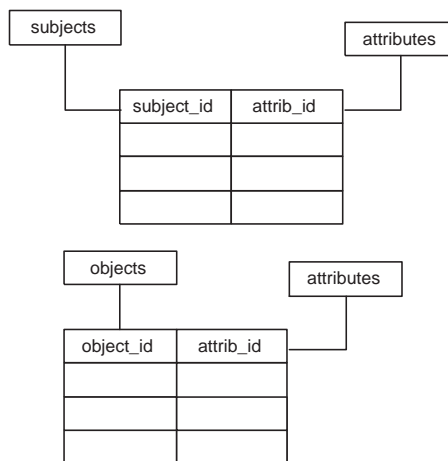


Figure 16: The general database tables (simplified view)

possible.

During the earlier phases, there was an attempt to use the existing solutions like CAS but a lot of limitations occurred and the future of CAS in its current form is not known yet. It was impossible to find an existing solution which would fully satisfy the requirements of the GridLab project. Thus, it is necessary to design a new service that would be able to fulfill all of the Gridlab requirements, and remain useful for other projects at the same time. The new service should be independent of any specific security model. It was the main reason to separate a part of the AS responsible for security modelling as an independent module.

The Authorization Scenario contains all information about the security model in such form which can be easy to understand and execute. The concept of the Authorization Scenario is introduced to make the AS universal and capable of implementing various security models.

The Authorization Scenario will be developed as a graphs of states. A subject (a user, a service, etc.) will change its state to the other one upon a predefined condition. The current state of the subject will be stored on the AS server side. The scenarios will be defined in text files, whose integrity will be verified while loading to the AS server. The language for defining scenarios has not been specified yet.

5.1 Sample Scenarios

In this section samples scenarios for the AS are presented (based on [3]). The following scenarios will be briefly described:

- CAS Scenario (5.1.1),
- Simple Super-Scheduler (5.1.2),
- More complex Super-Scheduler (5.1.3),
- User To User Asynchronous Collaboration (5.1.4),
- User To User Synchronous Collaboration (5.1.5).

Each subchapter contains a description of a sample scenario. For each scenario two diagrams are provided. The sequence diagram ([2], [6], [11]) shows interaction between AS and subject, object and service. The state diagram ([2], [6], [11]) describes internal AS states machine structure, states and connections between states.

The main reason for defining these scenarios is to obtain a set of predefined states which can be used in the AS prototype. The last subchapter (5.1.6) contains a summary of predefined states and parameters that will probably be used. In that section, a possible structure of a scenario file is also presented.

Specific terminology is used to describe scenarios in this section (based on [3]):

- Credential — an electronic document or record that conveys information to be used for authentication and authorization,
- Identity Credential — a Credential, through which a trusted party certifies an entity's identity,

- Attribute Credential — a Credential, which associates a set of attributes with a specific identity,
- Capability Credential — a Credential that holds a capability, which enables the presenter to perform a specific function,
- Subject — a principal in the context of a security domain [9]; user and service can be a subject in this meaning,
- Object — trusted computer system modeling usage: A system element that contains or receives information [13]; resources like server, service provided by a system, file or other data contained in an information system, an item of system equipment can be a object in this meaning,
- Subject Credential — Identity Credential referring to the subject, where the subject is a user,
- Service Credential — Identity Credential referring to the subject, where the subject is a service,
- Session Credential — Identity Credential referring to any subject; this credential is generated by AS and used to identify subject in multiple requests to AS,
- Object Credentials — Attribute Credentials and Capability Credentials for accessing object.

Please note that at this stage of the project, most of details in these sections are provided for informational purpose only and may change in the future.

5.1.1 The CAS Scenario

This scenario is created for compatibility with current version of the Globus CAS [12]. In the first step subject (user's) credential is checked for compatibility with the AS requirements. In the second step, object (resource) credentials for the specific subject are generated. The object credentials contain information, which can enable the subject (for example user) to make action or actions on the object (usually resource or service). In case of this scenario, the object credentials are assumed to contain the complete security policy for the subject. When the subject credential is not accepted by the AS server, a set of generated credentials is empty.

On figure 17, two situations are presented. In the first one, the subject credentials are not accepted by the AS server. In the second one they are accepted and the AS generates set of credential for the subject. These generated credentials can be used to access objects that trust the specific AS server. No further communication with the AS server is required in order to obtain access to the object.

The advantages of the CAS Scenario:

- Communication with AS is minimized,
- Simple scenario with only several states (easy to implement).

The disadvantages of the CAS Scenario:

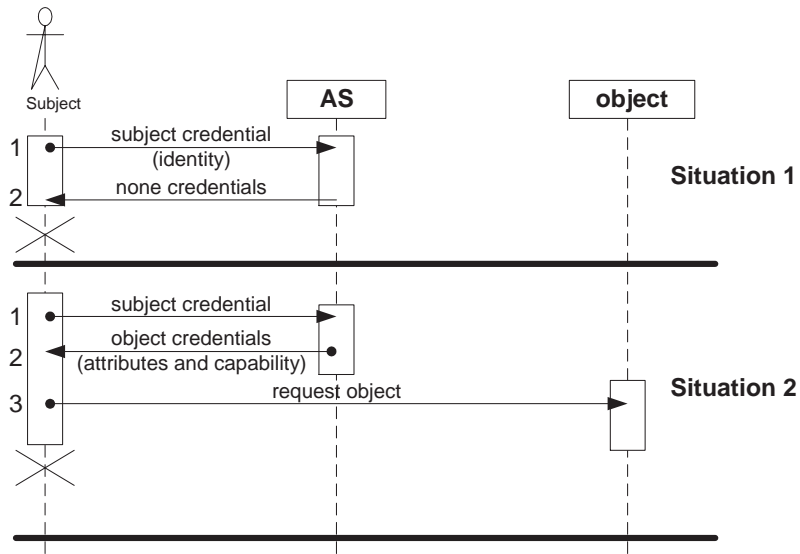


Figure 17: CAS Scenario Sequence Diagram

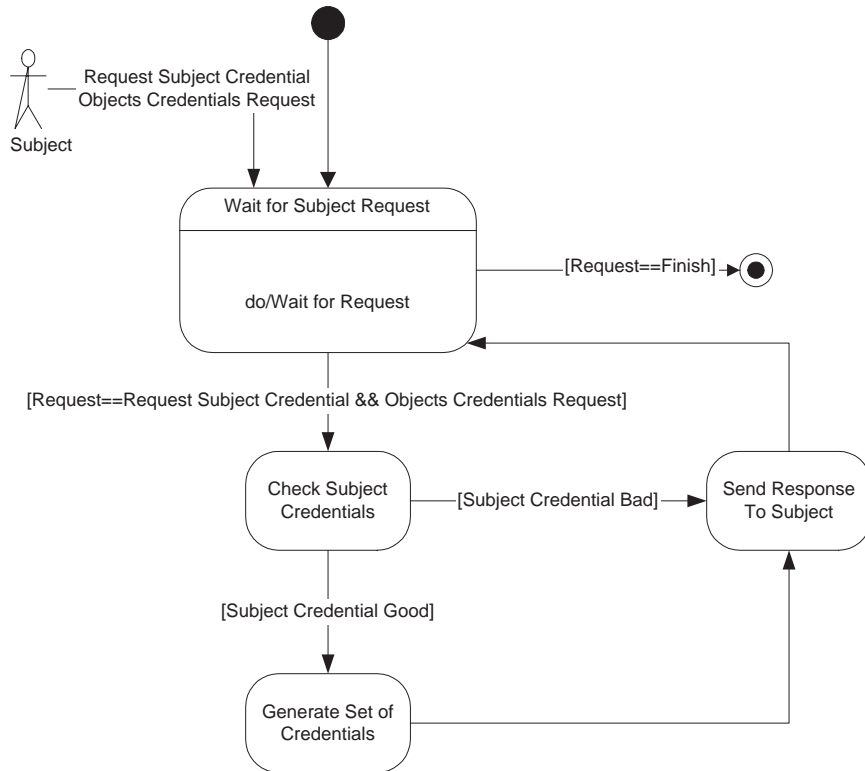


Figure 18: CAS Scenario States Diagram

- There is no possibility to revoke credentials; all modifications of security policy can be enforced with next request for credentials,
- The size of complete policy for subject may be large.

The states diagram (figure 18), presents states and connections between states for the CAS Scenario. In **WAIT** state, the AS server waits for subject's request or internal AS request like "Finish" (which ends scenario). In this scenario, only two types of requests are assumed: "Subject Credential" request and "Get Object Credentials" request. These requests are combined to minimize network communication.

After receiving a request the state is changed to **CHECK**, and subject credential is checked. If it is accepted by the AS server, the state is changed to **GENERATE**. In this state, the actual set of credentials is generated and the state is changed to **SEND**. If the subject credential is not accepted by the AS server (in the **CHECK** state), the state is changed directly to **SEND**. In the **SEND** state, the set of credential (possibly empty) is sent to the subject. This state can be only changed to **WAIT**.

Please note that this description will be also used in the case of other scenarios. In further sections, only extensions and differences to the CAS States diagram will be described.

The following states can be distinguished in this scenario:

- **CHECK_SUBJECT_CREDENTIAL**
input: subject credential,
output: YES/NO.
- **GENERATE_SET_OF_CREDENTIALS**
input: information about subject; list of objects for credentials (NULL for all objects),
output: object credentials covering full policy for the subject.
- **SEND_RESPONSE_TO_SUBJECT**
input: information to send,
output: sent YES/NO.
- **WAIT_FOR_SUBJECT_REQUEST**
input: request,
output: void.
- **START, STOP** — standard states.

5.1.2 Simple Super—Scheduler Scenario

A Super—scheduler is an entity that receives a description of a task from the user and then acts on the user's behalf to complete the task by coordinating all the resources necessary [3]. To cooperate with the AS server, the super-scheduler has to have its own credentials as well user's ones in order to obtain permission for accessing specific object. Comparing to the CAS scenario,

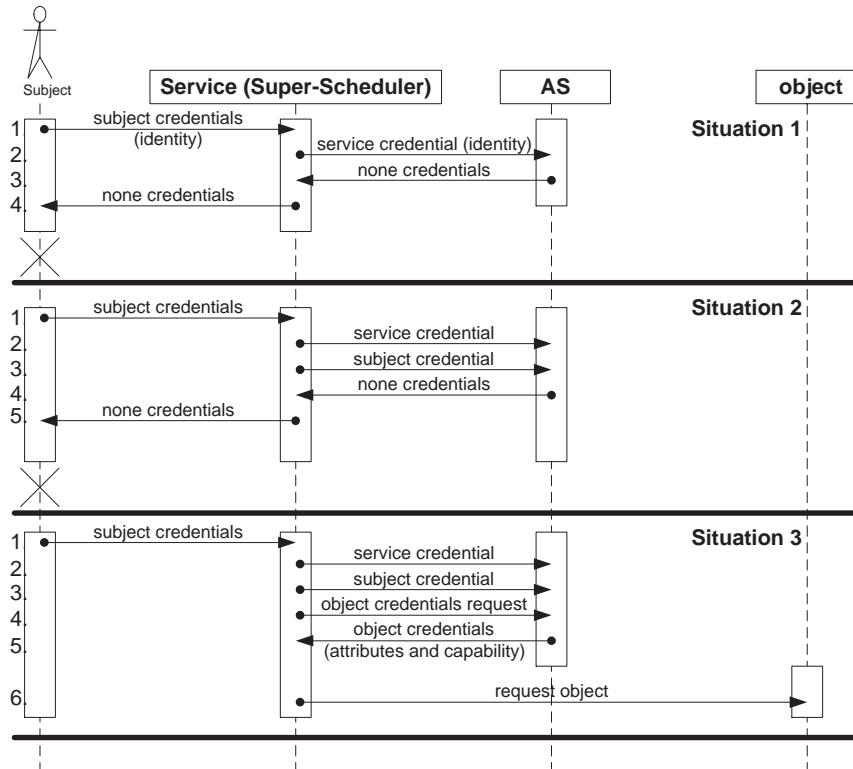


Figure 19: Simple Super—Scheduler Scenario Sequence Diagram

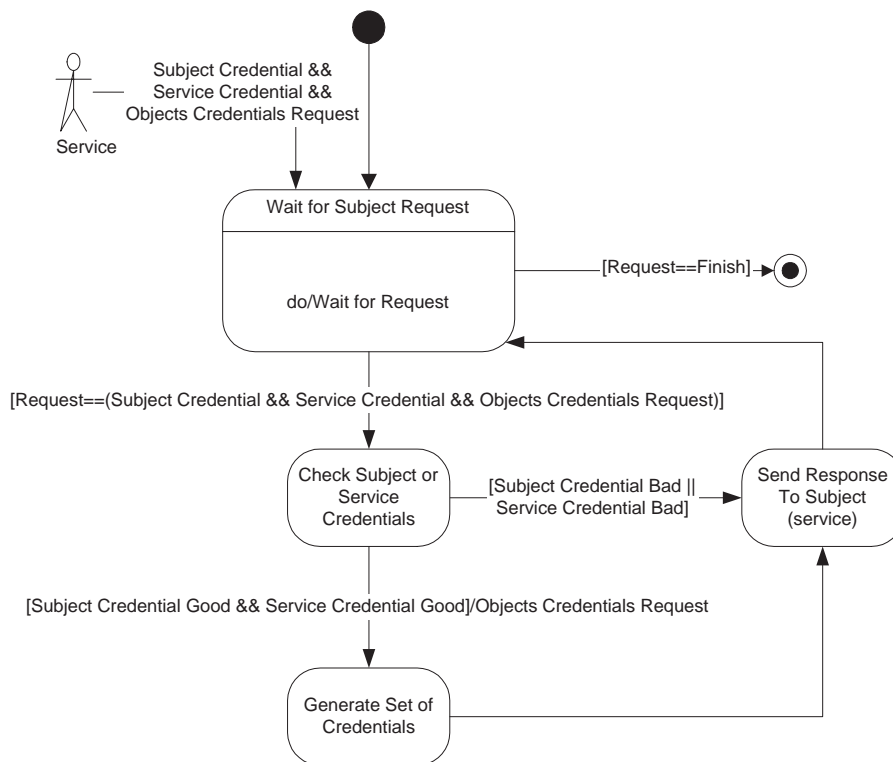


Figure 20: Simple Super—Scheduler Scenario States Diagram

the AS server does not generate a complete policy for the Super—Scheduler, but only required part.

On figure 19, three possible situations are presented. The first and the second one occur, when the subject (1st) or service (2nd) credentials are rejected by the AS server and the empty set of object credentials is generated. The last situation is fully succesful and new set of credentials for accessing the specific object is generated.

The advantages of Simple Super—Scheduler Scenario:

- Communication with AS is minimized,
- Size of required credentials depends on Super-Scheduler.

The disadvantages of Simple Super—Scheduler Scenario:

- There is no possibility to revoke credentials; all modifications of security policy can be enforced with next request for credentials,
- Poorly designed Super—Scheduler can use large set of credentials.

The states diagram for Simple Super—Scheduler Scenario is presented on figure 20.

The following states can be distinguished in this scenario:

- CHECK_SUBJECT_OR_SERVICE_CREDENTIAL
input: subject or service credentials,
output: YES/NO.
- GENERATE_SET_OF_CREDENTIAL
input: information about subject; list of objects for credentials (NULL for all objects),
output: object credentials covered Object Credentials Request.
- SEND_RESPONSE_TO_SUBJECT
input: information to send,
output: sent YES/NO.
- WAIT_FOR_SUBJECT_REQUEST
input: request,
output: void.
- START, STOP — standard states.

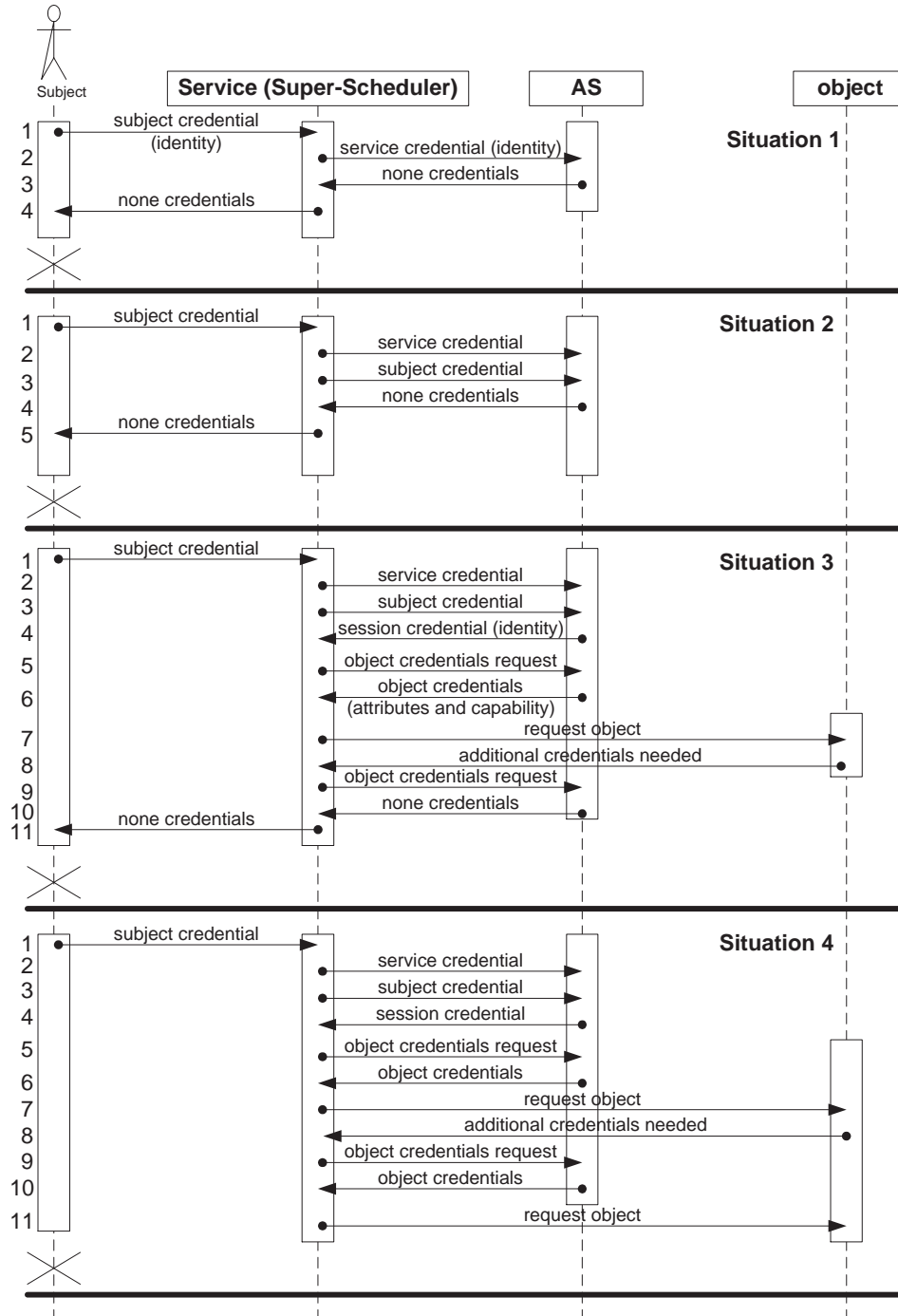


Figure 21: More Complex Super—Scheduler Scenario Sequence Diagram

5.1.3 More Complex Super—Scheduler Scenario

This scenario is similar to the previous one. However in this case, the Super—Scheduler is in continuous interaction with the AS server. After checking subject and service credentials, the AS generates a session credential for the Super—Scheduler. These session credentials are used by the service to communicate with the AS server.

On figure 21, four possible situations are presented. The first two cases end with failure, the third one with partial success and the last one with complete success. In the first case, the service is not authorized to use the AS server. In the second case, the user does not have sufficient privileges. In the third case, the object credentials are generated by the AS server, but the object (resource) requests additional credentials that are refused by the AS server. The last case presents fully succesful situation.

This scenario is more complicated then the previous ones, as the additional session credential is generated in order to enable mutiple interactions. The states diagram for this scenario is presented on figure 22.

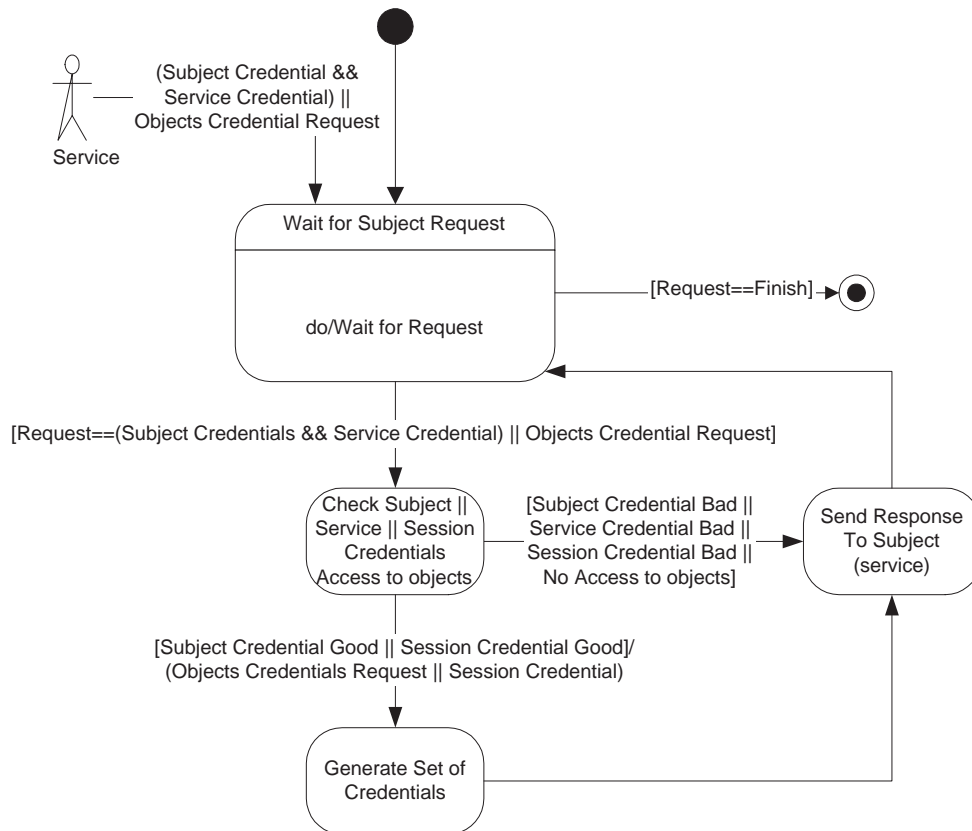


Figure 22: More Complex Super—Scheduler Scenario States Diagram

The advantages of More Complex Super—Scheduler Scenario:

- Size of required credentials depends on Super-Scheduler,
- It is possible to get object credentials for accessing specific objects only,

- It is possible to dynamically interact with the AS and get or update object credentials using session credential repeatedly.

The disadvantages of a More Complex Super—Scheduler Scenario:

- Poorly written Super—Scheduler can use a large set of credentials,
- The AS server has to be available online,
- Significant network communications between the AS server and the Super—Scheduler.

The following states can be distinguished in this scenario:

- CHECK_SUBJECT_OR_SERVICE_OR_SESSION_CREDENTIAL
input: subject or service or session credential,
output: YES/NO.
- GENERATE_SET_OF_CREDENTIAL
input: information about subject and service; list of objects for credentials (NULL for all objects),
output: object credentials covered Object Credentials Request or Session Credential.
- SEND_RESPONSE_TO_SERVICE
input: information to send,
output: sent YES/NO.
- WAIT_FOR_SUBJECT_REQUEST
input: request,
output: void.
- START, STOP — standard states.

5.1.4 User to User Asynchronous Collaboration Scenario

This is a special type of scenario, enabling direct cooperation of two subjects. The `subject_1` passes the set of credential to be used by `subject_2`. These credentials are stored on the AS server and no direct interactions between subjects are possible.

On figure 23, three possible situations are presented. The first and the second one fail, while the last one ends with success. In the first situation, the AS server refuses `subject_1` to store credentials for `subject_2`. In the second case, `subject_2` is not allowed to use stored credentials. In the last case, the credential is successfully used to access the object.

The advantages of User to User Asynchronous Collaboration Scenario:

- Network communication is minimized,

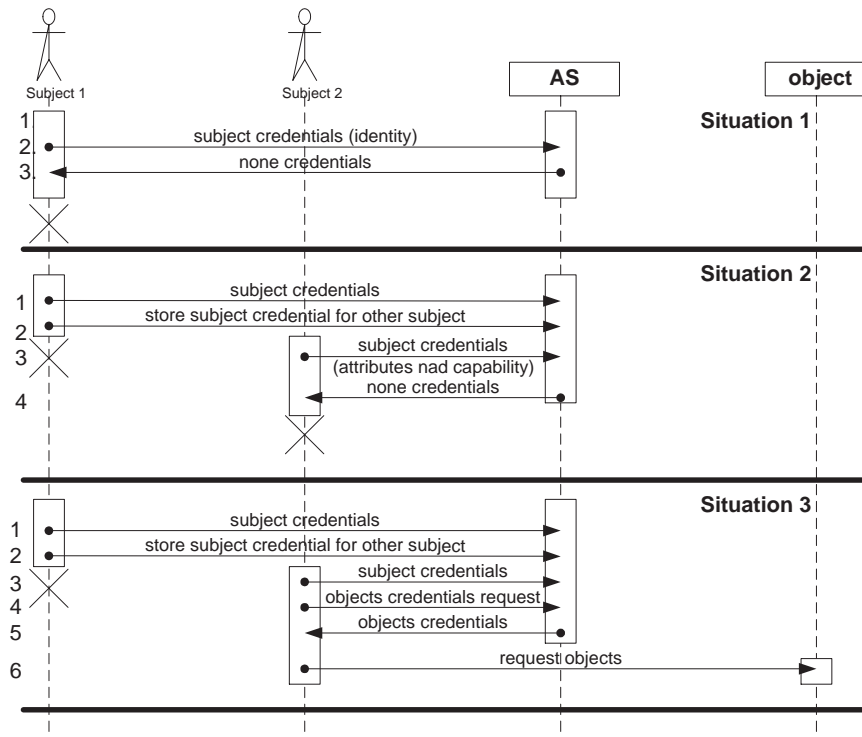


Figure 23: User to User Asynchronous Collaboration Scenario Sequence Diagram

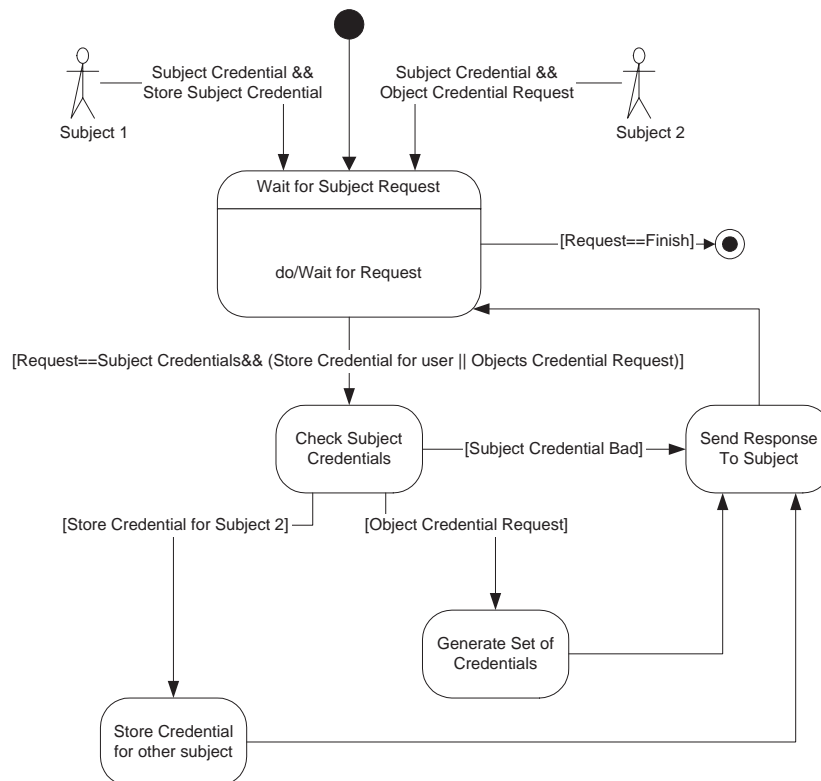


Figure 24: User to User Asynchronous Collaboration Scenario States Diagram

- The AS server does not have to accessible full time.

The disadvantages of User to User Asynchronous Collaboration Scenario:

- There is no possibility to get additional credentials form `subject_1` after first interaction,
- `Subject_1` has no control on sharing credentials.

On figure 24 the states diagram for this scenario is presented. Comparing to the previous state diagrams, an additional state STORE was introduce to store subject credential.

The following states can be distinguished in this scenario:

- CHECK_SUBJECT_OR_SERVICE_OR_SESSION_CREDENTIAL
input: subject or service credential,
output: YES/NO.
- GENERATE_SET_OF_CREDENTIAL
input: information about subject and service; list of objects for credentials (NULL for all objects),
output: object credentials covered Object Credentials Request or Session Credential.
- SEND_RESPONSE_TO_SERVICE
input: information to send,
output: sent YES/NO.
- WAIT_FOR_SUBJECT_REQUEST
input: request,
output: void.
- STORE_CREDENTIAL_FOR_OTHER_SUBJECT
input: credentials to stored,
output YES/NO (stored?).
- START, STOP — standard states.

5.1.5 User to User Synchronous Collaboration Scenario

This scenario is similar to the previous one. The biggest difference is that subjects and the AS server can continuously interact with each other and `subject_1` has to be *on-line* while the scenario is running. Therefore, the appropriate session credentials have to be generated. On figure 25, the states diagram for this scenario is presented.

On figure 26, three possible situations are presented. The first and the second one fail and the last one ends with success. In the first case `subject_1` is not able to store credentials for `subject_2`. In the second one, `subject_2` is not able to use these credentials.

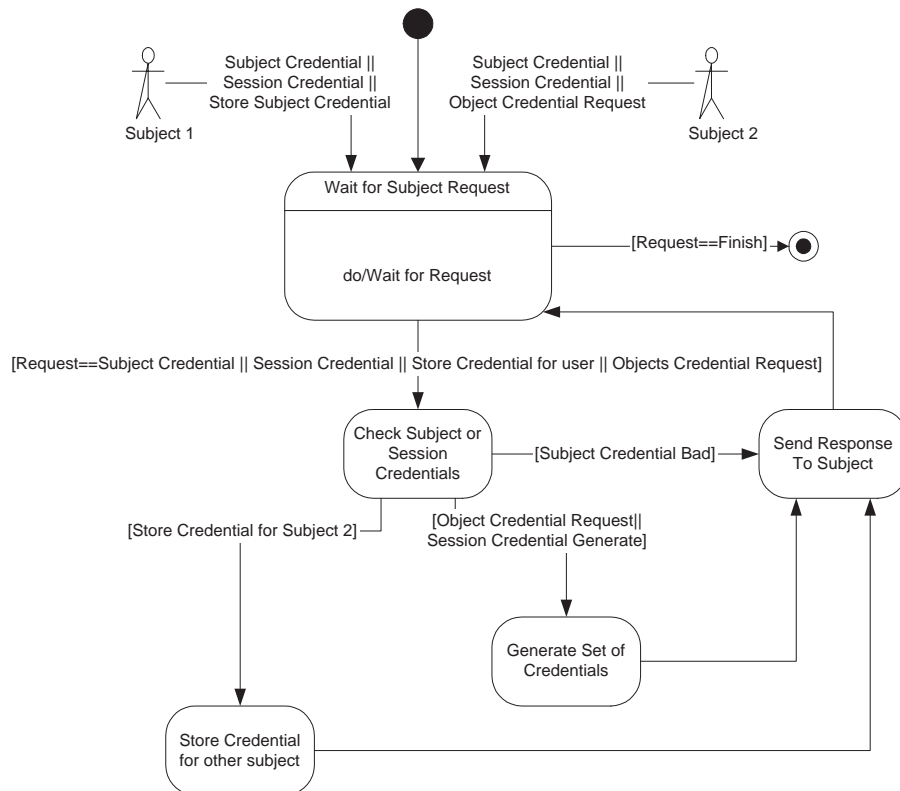


Figure 25: User to User Synchronous Collaboration Scenario States Diagram

In the last situation, the `subject_2` is able to use the credentials from the AS server, but the object has additional requests. Therefore, `subject_2` requests updated credentials from the AS server. `Subject_1` updates these credentials, and `subject_2` makes the request for the second time. With updated credentials, it is possible to access the object.

The advantages of User to User Synchronous Collaboration Scenario:

- It is possible to dynamically interact with AS, subjects and object,
- `Subject_1` has control for shared credentials,
- `Subject_2` cannot have all credential to access objects.

The disadvantages of User to User Synchronous Collaboration Scenario:

- Communication between AS and subjects can be significant,
- `Subject_1` has to be logged in for all scenario time.

On figure 25, the states diagram for the scenario is presented.

The following states can be distinguished in this scenario:

- `CHECK_SUBJECT_OR_SERVICE_OR_SESSION_CREDENTIAL`

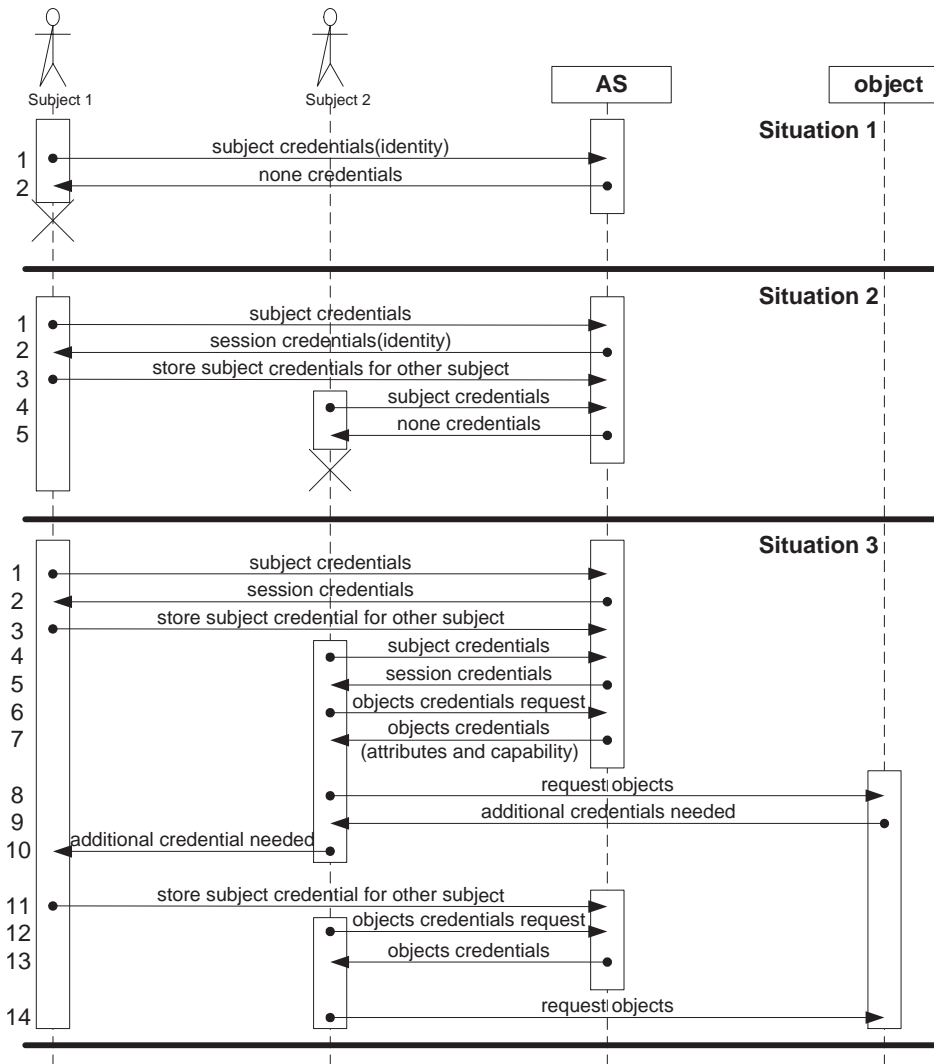


Figure 26: User to User Synchronous Collaboration Scenario Sequence Diagram

input: subject or service or session credential,

output: YES/NO.

- GENERATE_SET_OF_CREDENTIAL

input: information about subject and service; list of objects for credentials (NULL for all objects),

output: object credentials covered Object Credentials Request or Session Credential.

- SEND_RESPONSE_TO_SERVICE

input: information to send,

output: sent YES/NO.

- WAIT_FOR_SUBJECT_REQUEST

input: request,

output: void.

- STORE_CREDENTIAL_FOR_OTHER_SUBJECT

input: credentials to stored,

output YES/NO (stored?).

- START, STOP — standard states.

5.1.6 Scenarios Summary and proposition of scenario file format

Based on scenarios presented above, it is possible to determine five main predefined states:

- CHECK

input: set of credentials,

output: YES/NO.

- GENERATE

input: subject credential, information about subject, set of object,

output: set of credentials.

- SEND

input: information to send,

output: YES/NO (sent?).

- WAIT

input: void,

output: subject request.

- STORE

input: subject credential, information about subject, set of object, information about
subject_2,

output: YES/NO (stored?).

The CHECK state is used for getting authorization assertion decision. In the GENERATE state the AS server generates credentials based on input information. The generated set of credentials may be empty. The results are sent to the subject or the service in the SEND state. In the WAIT state, the AS server waits for subject's request. There is also the predefined state STORE, used for storing additional data (like credentials) for other subject.

[States]

State_1=Name_of_state, Type_of_state

State_2=Name_of_state, Type_of_state

(...)

State_n=Name_of_state, Type_of_state

[Connections]

Connection_1=State_1->State_2, Movement_conditions

(...)

A simplified proposal of a scenario file format is presented above. Two sections of the scenario file can be distinguished:

- The States section — contains information about states like name and type of state (pre-define state),
- The Connections section— contains information about connections; states that are connected and conditions, which have to be met to change from one state to another.

Please note that this proposal is initial and may be significantly changed and extended in the future.

6 Summary

The Authorization Service is assumed to be as universal as possible. It is designed with modular architecture, which should support many different authorization scenarios, created according to POP as well as PUSH model. Additionally, the security scenarios supported by the AS are not fixed, but fully editable by the system administrator.

The core of the AS server is independent from environment and as well as the system platform. It is equipped with five types of interfaces: management interface, database interface, CA interface (also for other security solutions) and I/O scenarios files - additional interface for getting scenarios descriptions from filesystem. The last one is a communication interface for user that *understands* various technologies; such as SAML or Globus.

Obviously, there are still several open points in the AS architecture. For example, the structure of the AS scenarios is not completely designed yet, so the structure of database is not known. The scenario language is also not defined at the moment.

The other open point is the efficiency of the system, which may be influenced by the complexity of applied scenarios. Very complex scenarios, when applied to the system, may be the cause of the slow down of the overall authorization process.

All these open points, as well as many others, will be fully investigated in the nearest future.

References

- [1] M. Adamski, M. Chmielewski, S. Fonrobert, A. Gowdiak, B. Lewandowski, J. Nabrzyski, T. Ostwald, and J. Pukacki. *WP6 Security: Initial Requirements*, April 2002.
http://www.gridlab.org/Internal/Drafts/wp6_requirements_draft.pdf.
- [2] Sinan Si Alhir. *UML in a Nutshell*, September 1998.
- [3] R. Butler, D. Engert, K. Jackson, M. Lorch, R. Olsen, and V. Welch. *Multiple Credentials: Scenarios and Requirements*. University of Auckland, Sun Microsystems, July 2002.
- [4] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. *The Physiology of the Grid*, June 2002.
<http://www.globus.org/research/papers/ogsa.pdf>.
- [5] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. *A Security Architecture for Computational Grids*, 1998. <ftp://ftp.globus.org/pub/globus/papers/security.pdf>.
- [6] Martin Fowler and Kendall Scott. *UML Distilled: A Brief Guide To The Standard Object Modeling Language*, 2000. RFC 2350.
- [7] GridLab project. *Description of Work*, August 2001.
<https://www.gridlab.org/Internal/Documents/Annex1.pdf>.
- [8] P. Hallam-Baker, E. Maler, and VeriSign. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)*. OASIS, May 2002.
<http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf>.
- [9] J. Hodges and E. Maler. *Glossary for the OASIS Security Assertion Markup Language (SAML)*. OASIS, May 2002.
<http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-01.pdf>.
- [10] P. Mishra and Netegrity. *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)*. OASIS, May 2002.
<http://www.oasis-open.org/committees/security/docs/cs-sstc-bindings-01.pdf>.
- [11] Object Management Group, Inc. *OMG-Unified Modeling Language, v1.4*, September 2001.
<http://www.omg.org/cgi-bin/doc?formal/01-09-67.pdf>.
- [12] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. *A Community Authorization Service for Group Collaboration*, 2002.
http://www.globus.org/security/CAS/CAS_2002_Revised.pdf.
- [13] R. Shirey. *Internet Security Glossary*, May 2000.
<http://www.ietf.org/rfc/rfc2828.txt>.