

31.01.07

Deliverable DJ1.2.4: PerfSONAR AA Service Specification



Deliverable DJ1.2.4

Contractual Date: 31/01/07
Actual Date: 31/01/07
Contract Number: 511082
Instrument type: Integrated Infrastructure Initiative (I3)
Activity: JRA1 – Performance Monitoring and Measurement
Work Item: 2
Nature of Deliverable: R – Report
Dissemination Level: PU – Public
Lead Partner: DANTE
Document Code: GN2-06-327v6

Authors: Maurizio Molina (DANTE) Andreas Solberg (UNINETT), Diego Lopez, Jose' Manuel Macias (RedIRIS), Jeff W. Boote (Internet2)

Abstract

This document is the result of an intense interaction between JRA1 and JRA5 and is meant to guide perfSONAR designers and developers through the task of exploiting the eduGAIN AA Infrastructure and code to add AA functionalities in perfSONAR. Hopefully, it will also serve as a reference for other GN2 activities having the same need of exploiting the eduGAIN framework and implementation.

Table of Contents

0	Executive Summary	v
1	General	1
1.1	Involved entities and their relationship	1
1.2	High level description of the AA interactions	3
1.2.1	Request phase	3
1.2.2	Authentication phase	3
1.2.3	Authorization phase	4
1.2.4	Single Sign On	5
2	Trust Model	6
3	Detailed Flow Sequence Description	7
3.1	Automated client (AC)	7
3.1.1	Before the AA process	8
3.1.2	AA sequence description	9
3.2	User behind a Client (UbC)	11
3.2.1	Before the AA process	12
3.2.2	AA sequence description	12
3.2.3	Single Sign On	13
3.3	Client in a Web containEr (WE)	14
3.3.1	Before the AA process	14
3.3.2	AA sequence description	15
3.3.3	SAML Constructs for Relayed Trust	16

	3.3.4	Lookup Service and Single Sign On	18
4		Implementation Guidelines	19
	4.1	User	20
	4.2	Client	21
	4.2.1	Querying the eduGAIN MDS	21
	4.2.2	SASL interface	21
	4.2.3	Sending and receiving the operation request	22
	4.2.4	X.509 certificate format	22
	4.3	perfSONAR resource	22
	4.3.1	Receiving/replying the operation request	22
	4.3.2	Sending/receiving the AuthZ request	22
	4.4	Authentication Service	23
	4.4.1	X.509 certificate format	23
	4.4.2	Receiving/replying the AuthZ request	23
	4.4.3	Security token validation	23
	4.4.4	Application of the AuthZ policy	23
	4.4.5	Sending/receiving the AuthN request	24
	4.4.6	Sending/receiving the Attr request	24
5		Conclusions	25
6		Acronyms	26
7		References	27

Table of Figures

Figure 1 – Relevant entities for perfSONAR’s AA interactions. R-BE, H-BE and MDS have an eduGAIN interface. The dotted line from the user to the resource represents the first user’s attempt to access it.	2
Figure 2 – AA interactions when an automated client accesses a perfSONAR resource.	8
Figure 3 – AA interactions when a human directed client accesses a perfSONAR resource.	12
Figure 4 – AA interactions when a Web-contained client is used to access perfSONAR resources.	14
Figure 5 – perfSONAR components that need to integrate AA functionality	20

0 Executive Summary

This document is meant to guide perfSONAR designers and developers through the task of exploiting the eduGAIN AA Infrastructure and code to add AA functionalities in perfSONAR.

The intense discussion among JRA1 and JRA5 first led to the detailed functional description of an additional perfSONAR service, the “Authentication and Authorization Service” that is needed in the perfSONAR services suite for adding AA functionality in perfSONAR. This service has the role of a concentrator of trust for perfSONAR resources and is positioned at the border of the perfSONAR and eduGAIN trust zones. Also, it has the role to centrally take authorization decisions, which can thus be consistent for all the resources (i.e. the perfSONAR services) of a domain.

Furthermore, some requirements brought by perfSONAR led to major extension of the eduGAIN design, namely the three additional profiles described in sections 3.1, 3.2 and 3.3. These profiles let all the different types of perfSONAR users and clients exploit the eduGAIN infrastructure.

The document progresses in an increasing level of detail, so that readers may stop at the point useful for them. It starts with a high level description of the entities (sec. 1.1), a high level description of the interactions among those entities (sec. 1.2) and an explanation of their trust relationships (sec 2). Then, it enters into a more detailed design of the interactions among those entities (sec. 3), and finally it explains how to make use of the code implementing the eduGAIN architecture that JRA5 developed (sec. 4).

1 General

The perfSONAR architecture consists of many interdependent services in a highly distributed and dynamic environment. For example, some network measurements need to run on the end users host. The perfSONAR architecture allows for services to be deployed in this distributed fashion.

The perfSONAR architecture was designed with federated authentication in mind. Therefore, one of the perfSONAR services (the AA Service) was specifically designed to act as an authentication proxy of sorts for the other services within a given deployment (usually a domain).

PerfSONAR's authentication and authorization functionality exploit the concepts, software components and infrastructure developed by JRA5. This infrastructure is called "eduGAIN".

1.1 Involved entities and their relationship

For the purposes of discussing authentication and authorization, the perfSONAR architecture can be simplified by saying that there are really only 5 entities of interest:

1. User or Client – The entity making a request for resources of some kind. Usually this will be software acting on behalf of a human user. However, perfSONAR will also have automated clients. These cases will be explicitly differentiated later on. When not relevant, however, the terms user and client are used interchangeably.
2. Identity Provider and Attribute Store. This is where the identities and associated attributes belonging to the user/client are stored. The Identity Provider and Attribute Store also identify the user's Home Domain for administrative purposes. EduGAIN provides a common interface to access the Identity Provider and Attribute Store, which is referred as H-BE (Home Bridging Element).
3. perfSONAR Resource (pSR)– a perfSONAR service. Actually, this document is not concerned about the whole functionality of the pSR, but only in the portion controlling the access to it, restricting the usage only to known and trusted users. The pSR could be information that should only be released to authorized entities, or it could be a consumable or schedulable

resource. The resource, in general, belongs to a domain different from the user's domain. The resource's domain is therefore called *Remote* (being remote with respect to the user).

4. Authentication and authorization Service (AS) – This is the entity that is expected to most directly interact with the eduGAIN infrastructure. It acts as a proxy between the user's Identity Provider and attribute store (accessible through the H-BE) and the perfSONAR services that provide resources (pSR). The AS is in the remote domain, and is therefore also referred as R-AS. The R-AS interfaces with the H-BE through an eduGAIN interface, and is called R-BE (Remote Bridging Element). This highlights the fact that there are two realms of trust (Home and Remote, see **Figure 1**) and this entity provides the connection between them.
5. Metadata Service (MDS) – This is a service accessible through an eduGAIN interface, and assists the user in the identification of the H-BE interface. Its role will become clearer in the following, when the detailed flow diagrams will be presented

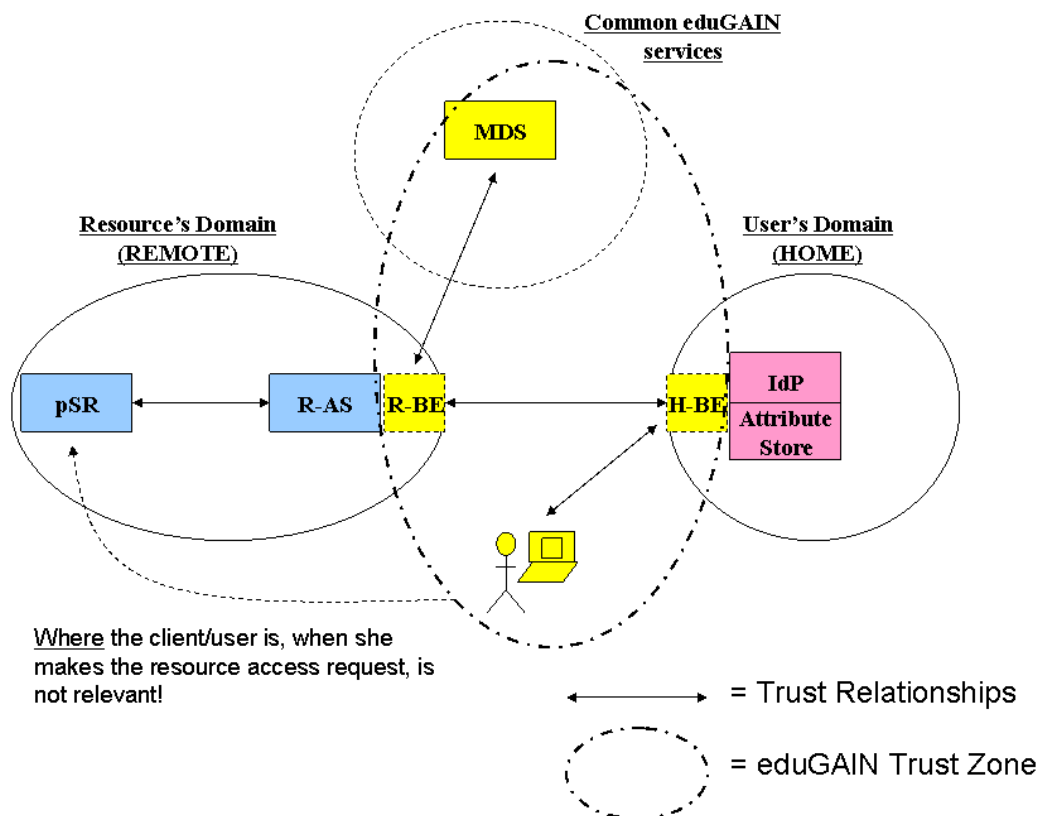


Figure 1 – Relevant entities for perfSONAR's AA interactions¹. R-BE, H-BE and MDS have an eduGAIN interface. The dotted line from the user to the resource represents the first user's attempt to access it.

¹

Figure 1 defines the trust relationships: any perfSONAR Resources (pSR) that are deployed within a domain will have a trust relationship with the AS of that domain only. Those pSR will not need to have a direct eduGAIN trust relationship with the H-BE, since the pSR can come and go too quickly for that model to be reasonable.

One way to illustrate this model is thinking about the way apache modules are deployed within the apache web server. There is an implicit trust relationship between the module handlers providing resources (web pages and such) and the modules providing authentication (mod_auth_basic). The modules providing resources rely on the authentication modules to identify and authorize the client. The model is fundamentally the same here but the modules are distributed across different services and potentially different hosts.

1.2 High level description of the AA interactions

This high level description is intentionally short and without the support of flow diagrams (these will be introduced in section 3).

1.2.1 Request phase

In the perfSONAR architecture a pSR will get requests from clients. If access to this resource is subject to authorization², then it will notify the client that an AuthN assertion (that is, a “proof of identity”) is required. The client will need to go to the H-BE to get that assertion.

If access to the resource is not subject to authorization, access will be granted without the need of going through the iterations described in this document.

1.2.2 Authentication phase

The way used to perform authentication differs depending if the client is an automated client or a human directed client (i.e. if there is a human user behind that client). The main difference is that for a fully automated client one cannot make the assumption that there can be an interactive request for credentials (e.g. opening a dialog window and inputting a username and password). In some cases, the appropriate location of the eduGAIN component authoritative for authenticating an automated client or a user must be established. The purpose of the eduGAIN Metadata Service (MDS) is enabling the discovery of this location by means of data pieces (provided by the user or the client) known as Home Locators (HL). In the case of an automated client, HLs can only be stored in the client’s credentials (e.g. in a client’s certificate), while in the human-directed cases it is possible to make an interactive request for such data as well.

² At the end, all that matters for resource access is AuthoriZation, i.e. mapping some user's attributes to the resource’s access policies. But this requires prior Authentication of the user (to the Identity Provider), and getting the user’s attributes from the Attribute Store.

Automated clients authentication will be supported by issuing an X.509 certificate to each of them. The certificate will be provided, in the first stage, by the eduGAIN root Certificate Authority (CA) or by one of the subordinate CAs defined in the eduGAIN trust model [2]. However, the eduGAIN trust model and the associated certificate validation libraries are open to the introduction of CAs that are not subordinate to eduGAIN's root CA. The client will implicitly pass this certificate in its initial request, by means of a TLS (secure) connection. The received certificate will be passed by the pSR to the AS in order to evaluate it. The AS will use the eduGAIN API to assess the validity of the certificate. The eduGAIN API will return a *component identifier* (CId) if the verification succeeds or an error otherwise. The AS may decide to perform authorisation by looking at this CId only, or to engage in further interactions with the H-BE (see below)

In the case of the human directed clients, different use cases arise depending on the type of client the user is accessing. Clients integrated in a Web container (CGI programs, servlets,...) can take advantage of the WebSSO (Single Sign On) facilities provided by the eduGAIN SSO profile [2]. Any other clients will on the contrary need to engage in a challenge-response authentication process with the H-BE to get a valid temporary certificate, before continuing onward in the above described process. For this purpose, eduGAIN H-BEs for organisations participating in eduGAIN will provide a SASL-enabled interface [4] to an on-line CA able to issue short-lived certificates [5]. These certificates will not contain the end user's name, for privacy reason, but only an opaque identifier that, if needed, the SASL can map to the user's identity (e.g. in case of a reported attempted abuse or misuse).

1.2.3 Authorization phase

If the CId is not sufficient for the authorisation decision, the R-AS must retrieve additional attributes for the user or client. Certificates to be used for authentication purposes will hold a specific extension, defining the interface to be used for accessing the corresponding H-BE, or at least an univocal key to retrieve the address of this interface from the eduGAIN MDS. MDS is a kind of lookup service for Authentication.

To support automated clients, H-BE (and therefore IdPs) will also need to support holding attributes for these automated clients: as automated clients will not be participating in a specific credential exchange, their attributes can be thought of as somehow more at risk, because the identity of the user is not checked in real time. However, it should be remembered that these are automated clients and their privacy (i.e.. the disclosure of client's attributes) should not be as much of a concern. Domains that don't wish to support automated clients do not need to.

The client will receive (or hold in the automated case) a valid certificate issued by the authenticating entity, which will contain a CId. The certificate is going to be presented by the client itself to other entities (to the pSR and to the R-AS) to show it has been authenticated. The CId can be used by these entities to retrieve additional information (attributes) about the client.

From the point of view of the pSR, however, the certificate is opaque. The pSR cannot know from the CId who the user is, nor whether it is authorized for accessing the resource. The pSR will completely rely upon a trust relationship with the R-AS/R-BE to determine authorization status using the certificate. Basically, what happens is that the pSR sends the certificate to the AS, asking for a "Boolean" Authorization (i.e. "Is the identity

associated with this certificate authorized to do what it requests?"). The AS will take the authorization decision. Before that, it can optionally make an appropriate attribute request to the client attribute store (through the H-BE), if this is needed to take the authorization decision. This aids in scalability because federation information only needs to be maintained at the AS service and not at all pSRs.

The Boolean requirement is intended to protect the privacy of the client/user. Attributes are not directly released to the pSR (which is "untrustworthy" from the point of view of the H-BE), but to the AS only. This allows for pSR such as measurement points to be deployed on end-user hosts temporarily and with a diminished trust requirement: the pSR does not need to deploy an eduGAIN interface and store all the root of trusts needed to validate the certificate. All it needs is to trust the AS. Once the AS has retrieved the attributes, it is up to it to decide whether a certain set of them (or the result of some postprocessing) will be returned to the resource. For example, it may return the max bandwidth a user is allowed to send for a BWCTL test (a test for assessing the achievable bandwidth along a path, which implies injecting a lot of test traffic...), but without disclosing the user's identity.

Note also that it is the AS, that belongs to the same domain of the pSR and has a trust relationship with it, that takes the Authorization decision. Centralizing the Authorization decision in an entity trusted by all the resources of a domain has the advantage that access resource policies can be easily coordinated among similar resources of a domain.

1.2.4 Single Sign On

The certificate held/obtained by the client/user constitutes its authentication token, and can be presented to other resources, in addition to the one accessed first. As long as the newly accessed resources are able to use the presented certificate to obtain from the R-AS (through the CId and, possibly, querying the H-BE) the Boolean authorization, single sign-on functionality is guaranteed. The certificate validity is limited to a short period of time in the case of human users. Revocation can be always checked by the R-AS through a query to the H-BE using the CId.

Note that in case of perfSONAR actions requiring authorization from several resources (e.g. in the case of a two-ended test), authorization must be obtained separately from the two resources. Some kind of nonce value can be passed from the client to each side, so simplistic verification can take place during the rendezvous. But, this is not part of the authorization protocol

2 Trust Model

In this section, the trust relationships among the entities introduced in **Figure 1** are briefly described. As the diagram demonstrates, there are two trust domains, the eduGAIN and the perfSONAR one, with the AS acting as a connector of the two:

1. the eduGAIN trust model is based on the standard eduGAIN trust fabric, as described by the eduGAIN architecture [1] and profile [2] documents. The AS acts as an eduGAIN R-BE and the IdP as an eduGAIN H-BE. The eduGAIN profiles mandate that all transactions have to be two-way-TLS validated
2. the trust between the pSR and the R-AS is the main addition required to the eduGAIN trust model. There will need to be an adequate trust relationship between the pSR and the R-AS. Because this is most often within the same domain, it is possible that IP address mask style authentication, or a shared secret, might be sufficient. However TLS style authentication can be implemented as well, even without the whole eduGAIN primitive set.

One future addition needed in the eduGAIN trust model to support the perfSONAR project is the ability for eduGAIN to install multiple root certificates. It is politically unlikely that all the diverse constituents of the perfSONAR project will be able to use certificates issued from the eduGAIN CA or even a sub-CA of it. It is likely that the current perfSONAR participants will need to use root certificates from at least the US DOE (Dept of Energy) as well as EDUCAUSE (US Universities). Other participants from Clara and APAN are also envisioned and will likely have similar concerns.

In principle, the design of the eduGAIN base libraries is taking into account the possibility of using several disjoint roots of trust. This means that a BE will be able to assess its trust on other BEs that identify themselves using different trust fabrics (different PKIs) just by their administrators defining the appropriate roots of trust (i.e. the roots of trust they want to rely upon) in their configuration. A root of trust consists of a root CA self-signed certificate.

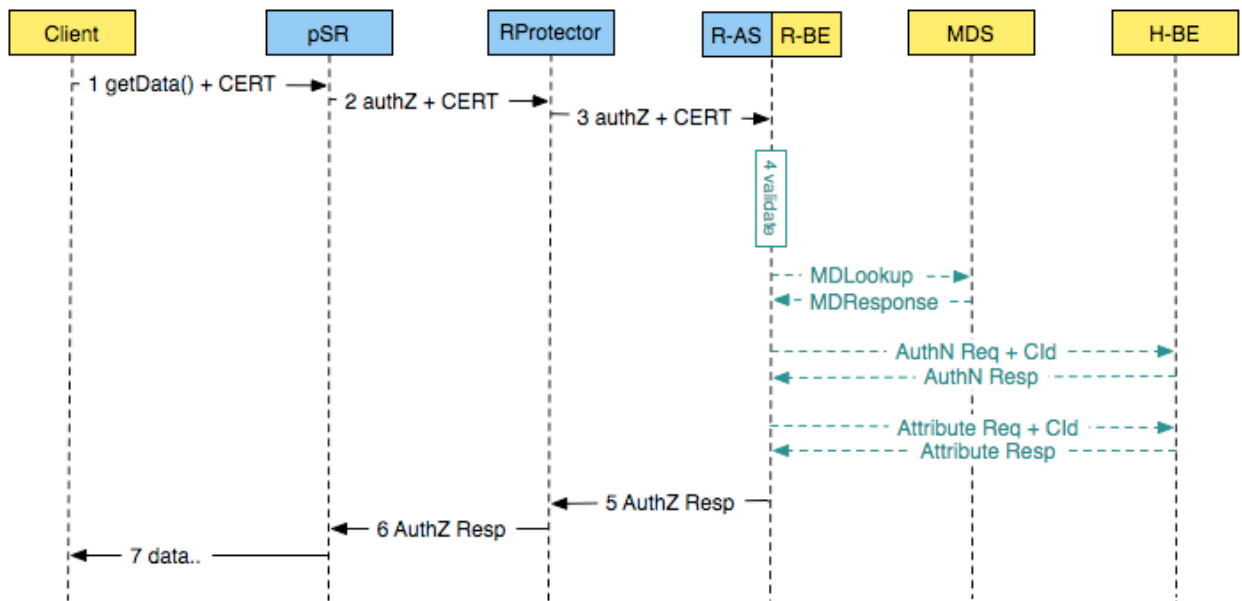
3 Detailed Flow Sequence Description

As introduced before, three cases are distinguished. The first one is an automated client needing to access a resource (sec. 3.1). When a human directed client accesses the resource, one must distinguish between the general case (sec. 3.2) and a more specific (and easier to handle, from an AA point of view) one, where the client is integrated in a web container (sec. 3.3).

3.1 Automated client (AC)

In the perfSONAR architecture, automated clients are entities (e.g. web services) interacting with other entities (or “resources”, from the AA point of view) to obtain services. Their distinctive feature, from an AA point of view, is that they act “autonomously”, i.e. without the involvement of a human user triggering their actions. So, the requests made by the client application itself needs to be authorized “on its own merit”, without any knowledge and possibility of identification of the human users ultimately using these services. Examples can be something as simple as a service monitoring script to ensure that a specific perfSONAR service is actually running. In particular, an example would be a Nagios monitoring application.

The flow sequence relative to this case it depicted in **Figure 2** below.



Automated client

Figure 2 – AA interactions when an automated client accesses a perfSONAR resource³.

Note that all entities in the picture above are WEB services, identified by URL. For example: <http://proxy.geant2.net/RRDMA-1.1/services/MeasurementArchive/Service>. The application protocol part of the URL (in the example, http://) defines what transport protocol is used by the underlying communication (e.g., for http, it is TCP) and possibly what specific port should be used to contact the service (if nothing is specified the default port of the application should be used...).

3.1.1 Before the AA process

Preceding the steps reported in **Figure 2**, there may be a phase where the client tries to access a resource, and gets back the indication that access “is not open”. In other words, the client gets back the indication that it has to access the resource with a valid certificate if it wants the resource to consider its “operation” request (e.g. get some data, or perform an action...). Clients must understand this “certificate required” response from a resource, and this is what triggers the entire AA process described hereafter.

However, in perfSONAR interactions, the client may not have enough information to perform this initial resource access, simply because it doesn’t know what the network address of the resource is. A reason for this can be that resources’ network addresses may be hidden to avoid DoS attacks on the resources. Or that the client

³ eduGAIN specific: green, dashed - perfSONAR specific black. getData() represent a generic operation request, not necessarily a data retrieval.

does not know what resources are available. The resource discovery process in perfSONAR is mediated by a service called Lookup Service (LS). The Lookup Service can be viewed as the first resource that the client tries to access. If the information being queried by the client requires authorization, the Lookup Service will answer with a “certificate required” message and trigger the AA process described above. The client must then return to the LS with the certificate. The LS will use it to complete authorization, and returns as resource data the list of the Network addresses of other resources

3.1.2 AA sequence description

Step 1) Operation request + Certificate submission

Each automated client owns an X.509 certificate signed by the eduGAIN common CA or its subordinate CA. The eduGAIN model is however open to introduce in the future other roots of trusts. The client must send this certificate to the resource in order to get access. Several options exist, including:

- Starting a TLS connection with mutual authentication with the resource.
- Sending the certificate in-line with the request headers or data.

Step 2) AuthZ request (sent by the resource) + Certificate submission

The resource forwards the received certificate to the AS (possibly through the Resource Protector) inside a “Boolean” AuthZ Request. Boolean means that the resource simply requests if the user/client presenting the certificate is allowed to do the requested operation. This means that:

- the resource doesn't try to “validate the certificate”. In other words, if the resource will receive a positive AuthZ Response, it will implicitly know that the presented certificate was valid
- the resource doesn't try to infer the identity of the holder of the certificate
- the resource doesn't try to directly get attributes about the holder of the certificate

From the above, it's clear that the resource must delegate a lot of responsibilities to the AS, and therefore it must *trust* the AS. This delegation has the advantage of both simplifying the resource's code for AA handling, and avoiding to build a direct trust relationship between all the resources and all the possible H-BEs (which would not scale well).

The type of operation for which authorization is required is specific for the resource being accessed, but as one of the parameters there MUST be the certificate obtained in the previous step.

Step 3) AuthZ Request (forwarded by the RP) + Certificate submission

Project:	GN2
Deliverable Number:	DJ1.2.4
Date of Issue:	31/01/07
EC Contract No.:	511082
Document Code:	GN2-06-327v6

For AA purposes, the RP can be viewed as a standalone service, or simply the part of the resource “consuming” the AuthZ Response. Actually, for most of the resources the RP is co-located with the resource. But in some cases the RP may be shared among multiple resources, and contain additional status information not known to the resource: for example, if other concurrent requests for active tests on a specific link have already been authorized and are being serviced.

Therefore, the RP has the potential to “block” (or delay) an AuthZ Request, or to modify it to adapt it to its status information. In case the request is blocked or delayed, this must be communicated back to the resource in a perfSONAR specific message. This response must (for the resource) be clearly distinguishable from a negative AuthZ resp conveyed in step 5.

Step 4) Certificate validation

The AS will then use the eduGAIN API to check that the certificate used by the client is a valid certificate and to get the eduGAIN component identifier (CId) of the client. This component identifier may be used to make an initial authorization decision. In addition, it may check the certificate against some revocation service. For a complete description of the validation library (eduGAINval) see [2].

Unnumbered optional step) H-BE location (search)

Valid certificates contain (inside a specific extension) either a direct reference to the H-BE of the client or an appropriate key to locate it through the eduGAIN MDS. MDLookup and Response perform this task. Knowing the H-BE interface is relevant if the AS is willing to perform additional checks on the certificate validity or the client attributes (see other optional steps below).

Unnumbered optional step) AuthN Request and response

In the simplest case, authorization can be accomplished by simply verifying the certificate and applying the local authorization policy based on CId, without any further queries to the H-BE, proceeding with step 5 below.

However, as an optional step, the AS may ask to the H-BE a direct authentication assertion about the received certificate, to check that it is still valid in this specific moment. For this, it must use the CId derived from the content of the certificate.

Unnumbered optional step) Attr Request and responses

In a more elaborate case, the AS may require one or more attributes about the user, to make its AuthZ decision. This depends on the AuthZ policies that are part of the AS logic.

Step 5) AuthZ resp (sent by the AS)

After (optionally) collecting attributes about the user the AS sends back a (Boolean) AuthZ response, applying a local policy for the resource. A centralized authorization policy guarantees fairness of resource usage within a domain. In case of a negative reply, the AS must send back some information about the reason of the failure. It is necessary to support at least three error reasons:

Project:	GN2
Deliverable Number:	DJ1.2.4
Date of Issue:	31/01/07
EC Contract No.:	511082
Document Code:	GN2-06-327v6

- Authentication required (the certificate was invalid or absent, i.e. the local validation in the AS in step 4 failed).
- Authentication Failure (H-BE didn't reply, or could not confirm identity: i.e. authN resp was negative)
- Access denied (the certificate was valid, attributes were retrieved, but resource cannot be accessed because of missing rights: i.e. the Authorization policy check in the AS failed)

Step 6) AuthZ Resp (forwarded by the RP)

The RP simply forwards the Boolean response (plus, possibly, other information helpful to the resource to handle this response). There should not be any blocking due to resource status, unless the status of the resource changed between step 3 and 6. But a good implementation of the RP, to avoid overloading the H-BE, should block (or queue) request in step 4 rather than sending them out in parallel and blocking them in step 6.

Step 7) Operation Response

Conveys back to the user the requested data, or gives resource specific information about how to access/control the resource, or simply acknowledges the success of the requested operation.

In case of a negative response a reason for it should be conveyed back, reflecting what has been reported in step 5).

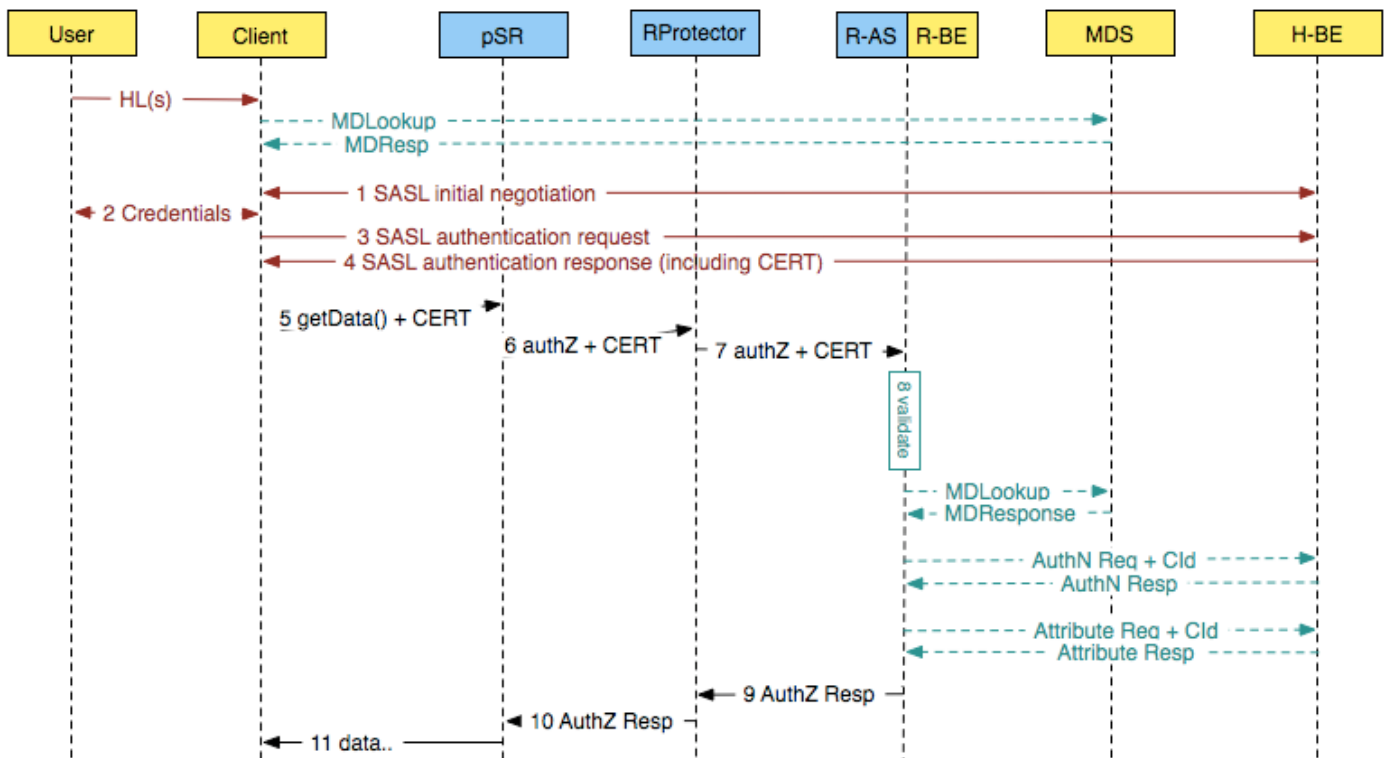
3.2 User behind a Client (UbC)

In the perfSONAR architecture, humanly directed clients are non Web Services (WS) entities acting directly upon requests coming from end users. These entities need to contact WS ("resources" from the AA point of view) to obtain services. An example is perfSONAR-UI that contacts several Measurement Archives to fetch data to be displayed to the user through a GUI. Their distinctive feature, from an AA point of view, is that they act under the instructions of a human user, and that resources will fulfil the requests only if they are satisfied with some level of knowledge of the end user⁴.

First, the case where the client is not running inside a web container (and thus has *not* Web redirect capabilities) is described. The case where the client is inside a web server and the user interacts with a browser (thus *with* Web redirect capabilities) is addressed in sec. 3.3.

The flow sequence relative to this case it depicted in **Figure 3** below.

⁴ This level of knowledge is resource-dependent, and may be as simple as knowing that the user has authenticated, or as complicated as checking a user's attributes against a specific policy.



Client on behalf of a user

Figure 3 – AA interactions when a human directed client accesses a perfSONAR resource⁵.

3.2.1 Before the AA process

Same as 3.1.1, except that the certificate needed to access the first resource (typically, the LS) needs to be obtained through the SASL interactions described below.

3.2.2 AA sequence description

The human user directed client case differs from the Automated Client case in how the certificate (CERT) containing the initial authentication assertion is obtained. After that, the (perfSONAR) client acting on behalf of a user does exactly the same operations as the (perfSONAR) automated client (compare the bottom part of Figure 2 and Figure 3).

⁵ Client-SASL specific: red - eduGAIN specific: green, dashed - perfSONAR specific: black. HL stands for “Home Locator”. getData() represent a generic operation request, not necessarily a data retrieval.

Therefore, only the initial steps of **Figure 3** are described.

Unnumbered steps) Location of user's H-BE

To let the user authenticate, the client needs to know the appropriate interface where user's collected credentials must be sent. This can be achieved in several ways, one of which implies the query of the eduGAIN MDS (MDLookup and MDRsp), using whatever hints (HLs) the user provides to the client.

Step 1) SASL initial negotiation

The client must forward the user's credentials to the H-BE in order to establish his/her identity (perfSONAR clients shall engage in a SASL [4] association with the H-BE in order to perform credential exchange). But preliminarily, the perfSONAR client starts a SASL negotiation with the H-BE, in order to establish the appropriate SASL environment (use of TLS, authentication methods required and supported, etc.).

BEs for eduGAIN federations participating in perfSONAR must incorporate a SASL responder able to follow the procedures described below. One such responder, SASL-CA [5] is available and undergoing improvement under the auspices of MACE [9].

Step 2) Credential request

According to the negotiation above, the client requests the user to provide the appropriate credentials.

Step 3) SASL authentication request

The client starts a SASL authentication with the H-BE forwarding the credentials it has received from the user in the above step.

Step 4) SASL authentication response

The H-BE, upon successful authentication is able to return an eduGAIN certificate containing a valid eduGAIN CId. This certificate shall have a short validity period, a transient and pseudonym CId (to preserve user's privacy) and the corresponding reference to the appropriate H-BE interfaces (for further authN or Attr requests), according to the procedures described for the automated client case.

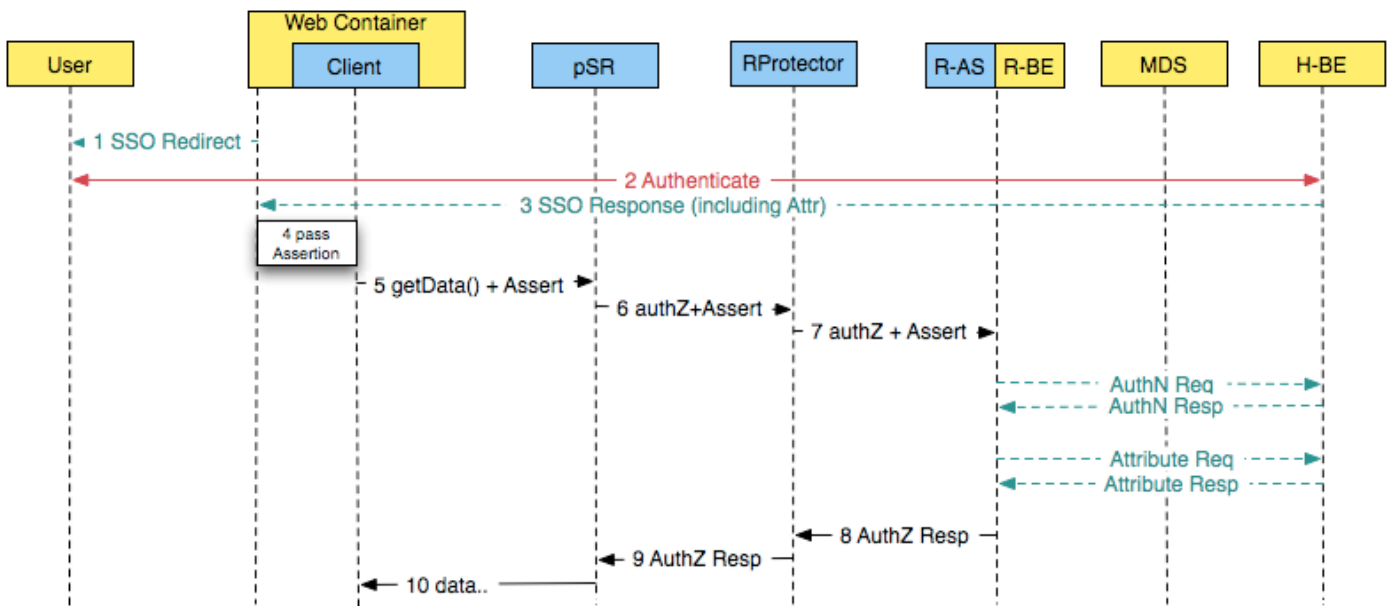
3.2.3 Single Sign On

After having accessed the first resource (typically, the LS) and having obtained the certificate from the SASL, the Client, within the validity time and scope of the certificate, can directly present the certificate to other resources, thus beginning the interaction at step 5 of **Figure 3**.

3.3 Client in a Web containEr (WE)⁶

A client within a Web container (a servlet, a PHP script, a program offering a CGI interface, etc.) can use its container to redirect users to their H-BE and make them authenticate there using their local federation procedures. This procedure is accomplished using the eduGAIN WebSSO profile.

The flow sequence relative to this case is depicted in Figure 4 below.



Client in a Web container

Figure 4 – AA interactions when a Web-contained client is used to access perfSONAR resources⁷.

3.3.1 Before the AA process

The user intends to access a perfSONAR client located within a Web container and protected by eduGAIN-aware federation software. The attempt of access triggers the WebSSO process according to eduGAIN.

⁶ While the natural acronym would be WC, we preferred WE for avoiding naming conflicts with other contexts

⁷ Home federation specific procedures: red - eduGAIN specific: green, dashed - perfSONAR specific: black. getData() represent a generic operation request, not necessarily a data retrieval.

3.3.2 AA sequence description

In this case, user authentication is performed by means of the same Web browser used to access the client. After the SSO steps, the client is able to send a proof of his identity (as asserted by the H-BE) to the resource (and hence to the AS).

In comparison with the cases described before, the AS is able to directly access the identity data (or use it for retrieving additional attributes) without processing any certificate, but using authentication assertions in SAML, which is the means by which the eduGAIN components exchange data to enable web SSO. Note that once the client is able to send the operation request (`getData()`) along with the SAML assertion, the behaviour is the same described before, just the received authentication assertion is sent instead of the certificate (compare the bottom part of **Figure 2** and **Figure 3** with that of Figure 4).

Step 1) SSO redirect

The client container (through the appropriate federation/eduGAIN mechanisms) redirects the user browser to the appropriate H-BE for authentication. In doing so, uses the eduGAIN (Shibboleth 1.3 compatible [10]) profile for WebSSO.

Step 2) User authentication

Applying the local procedures at the Home federation, the user is authenticated exchanging credentials at his/her local authentication point, without disclosing any sensitive information to entities other than those at the home domain.

Step 3) SSO response

According to the local ARPs (Attribute Release Policies), the H-BE sends back to the client container an identity assertion.

Step 4) Assertion pass

The container uses whatever local procedure to pass the received data to the client, preserving the original SAML `AuthenticationAssertion` received from the H-BE as part of the SSO response.

The client must use the received SAML `AuthenticationAssertion` to build a “relayed-trust” SAML assertion according to the profile described in section 3.3.3.

Step 5) Resource access

The client accesses a resource, with a resource specific operation request (`get_data()`) as in the other cases. The difference in this case lies in the fact that what is passed along with the request is a SAML `AuthenticationAssertion`, built as described in the step above.

Step 6) AuthZ request

As in the other cases, the resource relies on a central component (the AS) to perform authorization, because it can't directly trust the received attributes, and because a centralized authorization policy guarantees better fairness of resource usage.

Other steps are similar to the other cases (only, a SAML assertion is used to perform the AuthZ steps instead of an X.509 certificate) and thus are not detailed any further.

3.3.3 SAML Constructs for Relayed Trust

The SAML construct used in this case must be able to convey information about the user accessing the resource and fulfil two essential constraints:

- It must be bound to the client by the H-BE, so it is possible to check that the information about the user that it contains has been legally obtained,
- It must be bound to the resource by the client, so a potentially malicious resource can not use this information to further impersonate either the client or the user.

To comply with these two requirements, the client will build a SAML `AuthenticationAssertion` with:

- A valid audience restricted to the resource it is addressed to, through a SAML `AudienceRestrictionCondition` element containing an URI uniquely identifying the resource.
- A statement that this specific method of relayed trust must be used to evaluate the assertion, through a specific value in the SAML element `ConfirmationMethod`.
- The SAML `AuthenticationAssertion` received from the web container as evidence for this confirmation process, as part of the SAML element `SubjectConfirmationData`.

A sample SAML assertion following the above procedures for a given client with the eduGAIN CId:

`urn:geant:edugain:component:perfsonarclient:NetflowClient10082`

Acting on behalf of a user that it is identified by a BE with CId:

`urn:geant:edugain:be:uninett:idp1`

And connecting to a resource identified by:

`urn:geant:edugain:component:perfsonarresource:netflow.uninett.no/data`

Should have a content as the one displayed below (some line breaks and indentation added to improve readability):

```
<?xml version="1.0" encoding="UTF-8"?>
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:SAML:1.0:assertion
```

Project:	GN2
Deliverable Number:	DJ1.2.4
Date of Issue:	31/01/07
EC Contract No.:	511082
Document Code:	GN2-06-327v6

```

file:/Users/andreas/Documents/UNINETT/AAISpecs/SAML-1.1/oasis-sstc-saml-schema-assertion-1.1.xsd"
MajorVersion="1" MinorVersion="1" AssertionID="100001"
Issuer="urn:geant:edugain:component:perfsonarclient:NetflowClient10082"
IssueInstant="2006-12-03T10:00:00Z">

<!-- An audience restriction, that will restrict this security token to be valid for
one single resource only. -->
<Conditions>
  <AudienceRestrictionCondition
    <Audience>urn:geant:edugain:component:perfsonarresource:netflow.uninett.no/data</Audience>
  </AudienceRestrictionCondition>
</Conditions>

<!-- The authNstatement issued by the client itself -->
<AuthenticationStatement AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
  AuthenticationInstant="2006-12-03T10:00:00Z">
  <Subject>
    <NameIdentifier
      Format="urn:mace:shibboleth:1.0:nameIdentifier">aksjc7e736452829we8</NameIdentifier>
    <SubjectConfirmation>
      <ConfirmationMethod>relayed-trust</ConfirmationMethod>
      <SubjectConfirmationData>
        <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
          xmlns:xsi="http://www.w3.org/2006/XMLSchema-instance"
          MajorVersion="1" MinorVersion="1" AssertionID="_200001"
          Issuer="urn:geant:edugain:be:uninett:idpl"
          IssueInstant="2006-12-03T10:00:00Z">

<!-- This inner assertion is limited to only be valid for the client performing the WebSSO
authentication. This inner assertion cannot be reused or used at all by others than the
NetflowClient10082 instance. But NetflowClient10082 can use it as an evidence when used inside an
assertion issued by NetflowClient10082 using the relayed-trust confirmationMethod. -->
        <Conditions>
          <AudienceRestrictionCondition>
            <Audience>urn:geant:edugain:component:perfsonarclient:NetflowClient10082</Audience>
          </AudienceRestrictionCondition>
        </Conditions>

<!-- This is the inner authNstatement proving the authentication itself. These elements and attributes
must
be identical in the inner and outer assertion:
- AuthenticationStatement@AuthenticationMethod
- AuthenticationStatement/Subject/NameIdentifier
The inner assertion confirmationMethod must be urn:oasis:names:tc:SAML:1.0:cm:bearer. -->
        <AuthenticationStatement AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
          AuthenticationInstant="2006-12-03T10:00:00Z">
          <Subject>
            <NameIdentifier
              Format="urn:mace:shibboleth:1.0:nameIdentifier">aksjc7e736452829we8</NameIdentifier>
            <SubjectConfirmation>
              <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</ConfirmationMethod>
            </SubjectConfirmation>
          </Subject>
        </AuthenticationStatement>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<!-- Signed by the IdP (or Home Bridging element) -->
        <SignedInfo>
          <CanonicalizationMethod Algorithm="..."/>
          <SignatureMethod Algorithm="..."/>
          <Reference>
            <DigestMethod Algorithm="..."/>
            <DigestValue/>
          </Reference>
        </SignedInfo>
        <SignatureValue/>
      </Signature>
    </SubjectConfirmationData>
  </SubjectConfirmation>

```

Project:	GN2
Deliverable Number:	DJ1.2.4
Date of Issue:	31/01/07
EC Contract No.:	511082
Document Code:	GN2-06-327v6

```
</Subject>
</AuthenticationStatement>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<!-- Signed by client -->
  <SignedInfo>
    <CanonicalizationMethod Algorithm="..." />
    <SignatureMethod Algorithm="..." />
    <Reference>
      <DigestMethod Algorithm="..." />
      <DigestValue/>
    </Reference>
  </SignedInfo>
  <SignatureValue/>
</Signature>
</Assertion>
```

3.3.4 Lookup Service and Single Sign On

See 3.2.3. It is worth noting that SSO can be accomplished even before any interaction with perfSONAR resources, as the user may have been authenticated much in advance to access any other resource (a corporate portal or a library resource, or simply the client running on his/her local machine, for example).

4 Implementation Guidelines

This section details where, in the perfSONAR architecture, AA related functionality needs to be added, which transactions are part of the eduGAIN API and which java classes of the edugGAIN suite of libraries can be used. [7]. Note that even when the transactions do *not* require the use of an eduGAIN class, perfSONAR may want to re-use it instead of doing its own implementation.

Figure 5 below summarizes where, in the “human directed client” case described in 3.2, AA functionalities must be added in perfSONAR components. The automated case is similar, only the user is not present. The Client in a WEB container is slightly different, and when needed specific details for it are given.

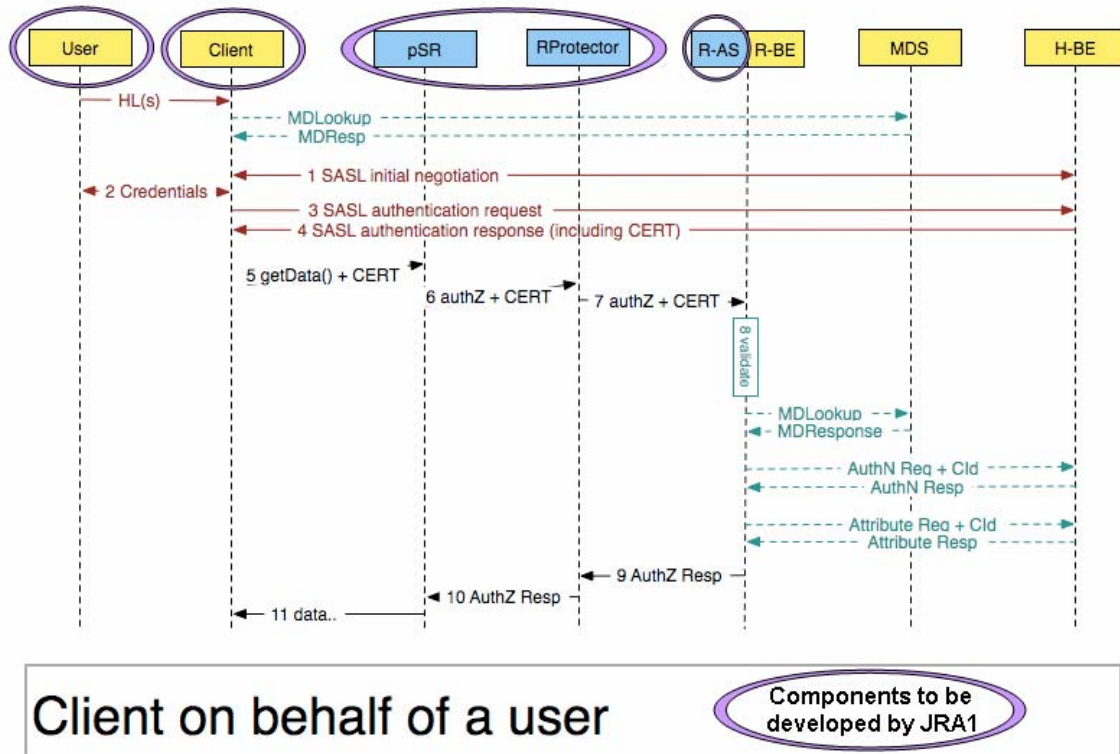


Figure 5 – perfSONAR components that need to integrate AA functionality

4.1 User

The user doesn't need to have specific "AA capabilities", but he/she must have the appropriate credentials for the federation (s)he belongs to.

Project:	GN2
Deliverable Number:	DJ1.2.4
Date of Issue:	31/01/07
EC Contract No.:	511082
Document Code:	GN2-06-327v6

4.2 Client

4.2.1 Querying the eduGAIN MDS

Part of the eduGAINMeta API Access to the MDS is provided by the method `net.geant.edugain.meta.rest.RESTClientImpl.lookupMetaData()`, that returns an object of class `net.geant.edugain.meta.BEMetaData`, that provides accessor methods to the interface definitions.

The client may need to query the eduGAIN MDS in order to locate the institution capable of authenticating the user. The eduGAINMeta API provides a set of methods for this purpose, allowing to perform metadata lookup and searching operations. The choice of the method depends on the parameters that can be provided. The method `net.geant.edugain.meta.rest.RESTClientImpl.lookupMetaData()` should be used when ID(s) of the BE and/or federation to which the user belongs are known, by providing them as parameters for the method call. If the ID of the user's H-BE is not known, but additional information is obtained from the user in the form of home locators (HLs), the method `net.geant.edugain.meta.rest.RESTClientImpl.searchMetaData()` should be used. The HLs must be provided as parameters to the method, which will use them in order to retrieve from the MDS the SAML 2.0 Metadata document corresponding to the H-BE. These methods return objects of the class `net.geant.edugain.meta.BEMetaData`, which provides access to all the relevant information from the metadata document.

The eduGAIN MDS supports three types of HLs:

- Home domain: A domain name indicating the user's home location.
- URN: A value encoded in URN format. Mainly intended for eduGAIN component identifiers.
- netID: An e-mail-like identifier in the form `localPart@FQDN`, though not necessarily a valid e-mail address of any kind.

In the AC case, certificates can hold either a direct interface URL for attribute queries, or a CId, that can be used as HL, so it is possible to move the H-BE interfaces without re-issuing the corresponding certificates. In the UbC case, the client may ask the user for some HLs (typically, home domain or netID) in order to establish the H-BE to which initiate the SASL connection. In the WE case, HLs shall be used during the Web SSO phase.

4.2.2 SASL interface

This is only applicable in the non-automated case. This is not part of the eduGAIN API, but there are many freely available implementations, like the standard one, using Java [3].

4.2.3 Sending and receiving the operation request

This is not part of eduGAIN API. In the AC and UbC case, the certificate should be passed in the request by means of a TLS connection, so the client can prove it holds the appropriate private key. Other functionally equivalent methods (see 3.1.2, step 1) may be considered. In the WE case the SAML assertion should be passed along with the other data in the request, in a separate element or section of the message.

4.2.4 X.509 certificate format

Certificates shall comply with the format specified for eduGAIN components [2], and shall hold an appropriate value in the `SubjectInformationAccess` extension [6], either:

- a) An URL specifying the appropriate H-BE interface for the client
- b) An URN to be used through the eduGAIN MDS to locate the appropriate H-BE interface(s) for the client.

4.3 perfSONAR resource

4.3.1 Receiving/replying the operation request

This is not part of the eduGAIN API. In the AC and UbC case it is advisable that the certificate is passed in PEM format [8] as an additional parameter. In the WE case, the received SAML assertion must be passed unchanged to the AS.

4.3.2 Sending/receiving the AuthZ request

This is not part of the eduGAIN API, but the eduGAINBase classes for authorization can be used for this. Class `net.geant.edugain.base.AuthorizationRequest` can be used to encapsulate an authorization request. Class `net.geant.edugain.base.AuthorizationRequester` can be used to perform the request against an eduGAIN-aware authorization responder. In all cases, the security token (certificate in the AC and UbC cases, SAML assertion in the WE case) must be sent as part of the `AttributeValueList` of the request.

Class `net.geant.edugain.base.AuthorizationResponse` can be used to encapsulate an authorization response and access its contents.

4.4 Authentication Service

4.4.1 X.509 certificate format

See 4.2.4.

4.4.2 Receiving/replying the AuthZ request

Class `net.geant.edugain.base.AuthorizationResponder` provides a servlet that can be configured properly using an adaptor class to receive and process the authorization requests, generating a `net.geant.edugain.base.AuthorizationResponse` object. The use of the adaptor is mandatory, and it must implement the `net.geant.edugain.base.AuthorizationRequestAdaptor` interface.

4.4.3 Security token validation

For the AC and UbC cases, the class `net.geant.edugain.validation.eduGAINvalidator` provides the method `validate()`, that returns an eduGAIN component identifier(CId) of the certificate (if valid) or an exception (if invalid). An additional method, `getInterfaces()`, for collecting the H-BE interface/locator, shall be available. In the WE case, the class `net.geant.edugain.base.AuthenticationResponse` can be used to process the SAML assertion, verify its security properties and access the data it contains (including the H-BE interface/locator). A custom `eduGAINBase` adaptor for the WE profile may be developed to make these interactions more straightforward.

In any case, the adequate configuration of `eduGAINVal` requires that the `getInstance()` method of class `net.geant.edugain.base.eduGAINConfiguration` must be used, passing to this method a `java.lang.String` containing the path to a file, which stores all the properties that rule the behaviour of the library. These properties point to the java keystores containing the certificates/trust paths conforming the eduGAIN trust model.

4.4.4 Application of the AuthZ policy

For using an eduGAIN-aware authorization service, see 4.3.2.

If the policy is based simply on the component identifier within the certificate, an easy extension of `net.geant.edugain.validation.eduGAINvalidator` could provide this for the AC and UbC cases, by overloading the `validate()` method. In the WE case, accessors exist for the appropriate elements in `net.geant.edugain.base.AuthenticationResponse`.

4.4.5 Sending/receiving the AuthN request

A `net.geant.edugain.base.AuthenticationRequest` must be created and configured, then the class `net.geant.edugain.base.AuthenticationRequester`⁸ is used to send it (through its `request()` method) and receive the response, in the form of a `net.geant.edugain.base.AuthenticationResponse` object.

4.4.6 Sending/receiving the Attr request

A `net.geant.edugain.base.AttributeRequest` must be created and configured, then the class `net.geant.edugain.base.AttributeRequester` is used to send it (through its `request()` method) and receive the response, in the form of a `net.geant.edugain.base.AttributeResponse` object.

⁸ `AuthenticationRequester` is not in the repository yet (still under development). Basically it performs similar operations as the `AttributeRequester`, which is already included in the repository.

5 Conclusions

This document was written to guide perfSONAR designers in the task of adding AA functionalities in perfSONAR, making use of the eduGAIN AA Infrastructure and code base.

It is the result of an intense interaction between JRA1 and JRA5, and to support some of the perfSONAR's use cases JRA5 had even to extend eduGAIN, through the definition of additional profiles. The most challenging and novel requirement coming from perfSONAR was to also allow web services (i.e. "Automated Clients") issue Authentication and Authorization requests. This was extensively described in sec. 3.1. Another requirement was to avoid that every single perfSONAR service (or resource) must have a full trust relationship with all the possible sources of Authentication and Authorization assertion. This led to the concept of a new perfSONAR service, called "AA Service" that can act as a proxy of trust for all the resources of a domain. The AA Service also centrally takes authorization decisions, which can thus be consistent for all the resources of a domain.

Detailed flow sequences for all the perfSONAR use cases were included in the document, as well as (in sec. 4) an indication of what parts of the existing eduGAIN codebase can be readily integrated by the perfSONAR developers.

JRA1 and JRA5 developers have started implementing this design into code to be integrated in the perfSONAR services suite.

Hopefully, the document it will also serve as a reference for other GN2 activities having the same need of exploiting the eduGAIN framework and implementation.

6 Acronyms

AC	Automated Client
AS	Authentication and authorization Service
AuthN	Authentication
AuthZ	Authorization
BWCTL	BandWidthCoNTroL
CA	Certificate Authority
CId	Component Identifier
H-BE	Home Bridging Element
HL	Home Locator
LS	Lookup Service
MDS	Meta Data Service
R-AS	Remote AS – same as AS
SAML	Security Assertion Markup Language
SASL	Simple Authentication and Security Layer
SSO	Single Sign On
TLS	Transport Layer Security
UbC	User behind a Client
URL	Uniform Resource Locator
WE	Web ContainEr
WS	Web Services

7 References

- [1] Deliverable DJ5.2.2: GÉANT2 Authorisation and Authentication Infrastructure (AAI) Architecture
- [2] Deliverable DJ5.2.3,1: Best Practice Guide - AAI Cookbook - First Edition Guidelines for Connecting to the eduGAIN AA Infrastructure
- [3] <http://java.sun.com/j2se/1.5.0/docs/guide/security/sasl/sasl-refguide.html>
- [4] RFC2222. Simple Authentication and Security Layer (SASL)
- [5] <http://lionshare.its.psu.edu/support/documentation/developers/sasl-ca-deployment-guide>
- [6] RFC3280. Internet X.509 Public Key Infrastructure
- [7] <http://www.rediris.es/app/eduGAIN-base/javadoc/>
- [8] RFC 1421-1424 Privacy Enhancement for Internet Electronic Mail: Parts I to IV
- [9] MACE: Middleware Architecture Committee for Education - <http://middleware.internet2.edu/MACE/>
- [10] The Shibboleth project: <http://shibboleth.internet2.edu/>