Authentication and Authorization in distributed systems: a survey on AAI technologies.

Technical Report

Jesús Luna García
PhD Student, Polytechnic University of Catalonia
jluna@ac.upc.edu

Abstract

These last years security problems related with authentication and authorization on distributed systems were widely studied, but as separate features. However today researchers have begun to deal with them as a single problem, using existing mechanisms and developing new ones according to this new integrated vision. The term conceived to reference this kind of technologies is "Authentication and Authorization Infrastructures" or AAIs. These new mechanisms gave birth to a broad range of new problems related with interoperability and scalability issues, and even with the need to reuse already implemented technologies to reduce its adoption costs. Nowadays heterogeneous and widely distributed environments as computational Grids are the most affected by these problems, and new research is aimed to solve them while keeping a balance between security, usability and performance.

This technical report precisely reviews a couple of AAI oriented conceptual frameworks to introduce basic terms and technologies; finally some relevant systems in this area are presented. Our objective is to identify design goals and problems, existing mechanisms and even mention some challenges so future designs might use this information as reference.

Table of contents.

1. Introduction.

The problem of authentication (AuthN) and authorization (AuthZ) in computer systems and more specifically in distributed systems has been an area of growing interest these last years. Its beginnings can be distinguished since 1984, when Henry Levy introduced the concept of "capabilities" in computer systems [1] making a through survey of early capability-based and object-based hardware and software systems. On his book, Levy defines a capability as a "token, ticket, or a key that gives the possessor permission to access an entity or object in a computer system" and as we shall see later, nowadays this term is still valid to define AuthZ schemes for a wide variety of distributed systems ranging from Web Services to the computational Grid. However we can find some interesting common properties shared between current implementations, mainly the integration of the AuthN and AuthZ concepts in one system often called "Authentication and Authorization Infrastructure" or AAI as we shall call it along this paper.

Many AAI implementations also have developed secure mechanisms to express user's attributes and resource's use-conditions, in such a way that a decision capable engine may derive a correct, traceable, unambiguous and promptly response for an AuthN or AuthZ query. However the development of such a wide range of solutions has caused a lot of interoperability problems and "collisions" or in other words, identical functionalities being implemented with different technologies on the same distributed system resulting also in design and performance problems, and even the impossibility to leverage an uniform security level between all of them.

Being focused on that problem we present in this report a review of the most important conceptual frameworks and base technologies used to implement AAI solutions in today's distributed systems –including the computational Grid-; also we have classified these mechanisms according to its main functionality. Finally this report reviews some existing AAIs and concludes about its main research challenges.

This report is organized in the following way: section 2 introduces some basic concepts through a brief review of two commonly accepted conceptual authorization frameworks; section 3 explains some AAI's basic technologies classified according to one of those frameworks; next on section 4 are presented some widely used AAI solutions, and finally section 5 summarizes some conclusions about current research challenges on this area.

2. Authorization Frameworks.

Base requirements for AAIs can be specified in frameworks describing basic architectural entities along with its trust relationships, message sequences and use case scenarios, privilege and policy management, and in some cases even enforcement mechanisms. In this section we introduce two conceptual authorization frameworks, which are important to study because of their relevance in designing and building several widely used AAIs. It is worth to note that authentication is not directly referenced in these frameworks, however is understood by them as a process closely tied to authorization.

2.1. RFC 2904 Authorization Framework.

This document [2] presents an architectural framework for understanding the authorization of Internet resources and services, and derives requirements for authorization protocols. Even though it does not explicitly consider the identification stage, is expected that this work may be extended to a more comprehensive model and that the scheme described can be incorporated into a more general framework that includes authentication, accounting and auditing (AAA). Also highlighted is the fact that the authentication method used by the participating parties influences the requirements found in the subsequent authorization process.

2.1.1. Authorization Entities and Trust Relationships.

The basic conceptual entities that may be participants in an authorization process are:

- A User who wants access to a service or resource.
- A User Home Organization (UHO) that has an agreement with the User and checks whether it is allowed to obtain the requested service or resource. This entity may carry information required to authorize the User, which might not be known to the Service Provider (such as a credit limit).
- A Service Provider's AuthZ Server which authorizes a service based on an agreement with the UHO without specific knowledge about the individual User. This agreement may contain elements that are not relevant to an individual user (i.e. the total agreed bandwidth between the UHO and the Service Provider).
- A Service Provider's Service Equipment which provides the service itself.

Authorization decisions are based on bilateral agreements between pairs of organizations involved in a transaction. Note that these agreements may be implemented by hand configuration or by evaluating a *Policy* data stored in a Policy database. The point here is that there must be a *set of known rules* in a place between entities in order for authorizations transactions to be executed. Trust is necessary to allow each entity to "know" that the policy it is authorizing is correct. Authentication servers may be used to establish trust relationships (i.e. Kerberos or X.509 digital certificates).

2.1.2. Authorization sequences.

In general, the User Home Organizations and the Service Provider are different entities or different "administrative domains". However the following sequences are similar to simpler scenarios (i.e. same entity, same domain).

2.1.2.1. The Agent Sequence.

In the agent sequence (figure 1), the Service Provider AuthZ Server functions as an agent between the User and the service itself. This server receives a request from the User and forwards authorization and possibly configuration information to the Service Equipment. The following dialog occurs in this scenario:

1. The User sends a request to the Service Provider's AuthZ Server, which will apply a policy associated with the User and the particular service being requested.
2. The AuthZ Server sends a request to the Service Equipment, and this sets up whatever is requested.
3. The Service Equipment the responds to the AuthZ Server acknowledging that is has set up the Service for the User.
4. The AuthZ server replies to the User telling it that the Service is set up.

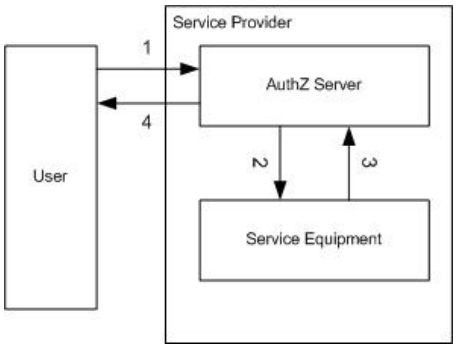

Figure 1. Agent Sequence.

2.1.2.2. The Pull Sequence.

The following steps take place in this sequence (figure 2):

1. The User sends a request to the Service Equipment.
2. This request is forwarded to the Service Provider's AuthZ Server.
3. The Service Provider's AuthZ Server evaluates and returns an appropriate response to the Service Equipment.
4. The User obtains an answer from the Service Equipment.

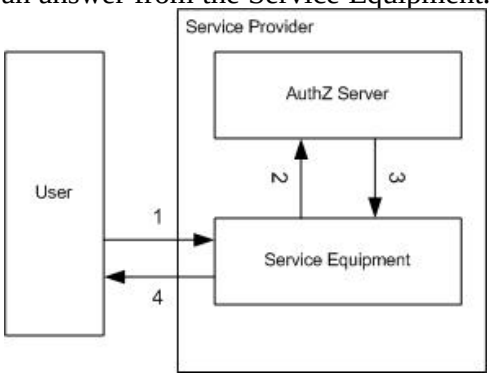

Figure 2. Pull Sequence.

2.1.2.3. The Push Sequence.

Figure 5 shows the Push Sequence, which uses the following steps:

1. The User ask the AuthZ Server for a ticket or certificate verifying that it is fine to have access to the service. This ticket usually has a validity period included and depending on the application may be used for more than one request.
2. The AuthZ Server returns a ticket to the User.
3. The User then includes this ticket in the request to the Service Equipment.
4. The Service Equipment uses the ticket to verify that the request is approved by the Service Provider's AuthZ Server and returns an appropriate answer to the User.

Note that in this sequence the communication between the AuthZ Server and the Service Equipment is relayed through the User rather than directly between themselves.

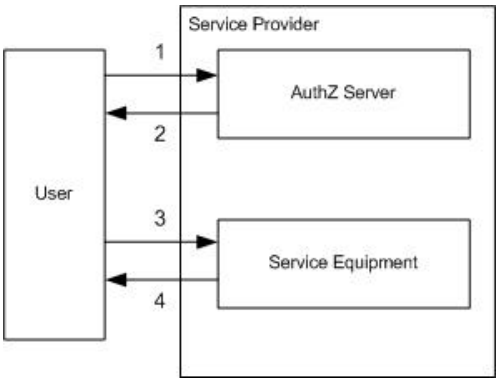

Figure 3. Push Sequence.

2.1.3. Authorization Policies.

One view of an authorization is that it is the result of evaluating policies of each organization that has an interest in the AuthZ decision. It is very important to establish the mechanisms required to *represent*, *manage* and *share* policies and policy information in a vendor-independent, interoperable and scalable manner all along the system. Policy definitions are *maintained* and *stored* in a policy repository by (or on behalf of) the organization that requires them.

This document considered the following operating over an independent policy:

- *Policy Retrieval:* Policy definitions must be retrieved in order to be evaluated and enforced. This operation is typically done by the administration that defines the policy or by an agent acting for that administration.
- *Policy Evaluation:* This requires access to information referenced by the policy, even though this information is not available in the administration where the policy was retrieved.
- *Policy Enforcement:* This task is typically done by the Service Provider on the Service Equipment.

In other terms: a policy is retrieved by a *Policy Retrieval Point (PRP)* from a Policy Repository, evaluated at a *Policy Decision Point (PDP)* or Policy Consumer, and enforced at a *Policy Enforcement Point (PEP)* or Policy Target. Information against which policy conditions are evaluated may be accessible at Policy Information Points (*PIPs*) and might be accessed using Policy Information Blocks (*PIBs*). An interesting question arises about the optimal location of each one of these servers on a distributed system. Figure 4 shows a common solution.

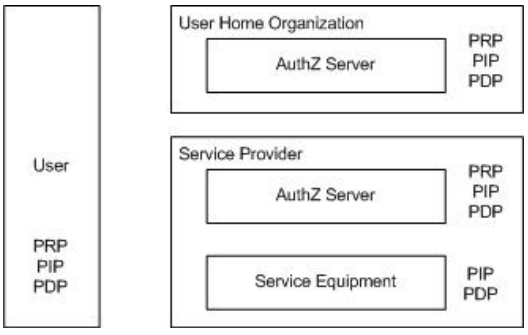

Figure 4. Common location of different Policy Elements.

2.1.4. Resource Managers.

The resource manager is the component which tracks the state of sessions associated with an AuthZ Server or Service Equipment. It also may allocate resources to a session and may track use of resources allocated by peer resource managers to a session. The resource manager also provides interfaces to allow the User to acquire or release authorized sessions. In other words, it is the anchor point in the AuthZ Server from which a session can be controlled, monitored, and coordinated even if that session is consuming network resources or services across multiple Service Provider administrative domains.

2.1.5. Other security issues.

This framework also considers the following security features:

- AuthZ Server Message Forwarding and Delivery: it is desired to use also a message delivery systems supporting unicast capabilities, data integrity and error correction, reliable data transport assurance, sequenced data delivery and adaptable to network congestion feedback.
- End-to-end security: when the AuthZ servers communicate through intermediate AuthZ Servers such as brokers, it may be necessary that a part of the payload be secure between the originator and the target AuthZ Server.
- Streamlined Authorization Process: To minimize Internet transversals in order to reduce latency in AuthZ Servers messages, data fields necessary for both authentication and authorization should be able to be carried in a single message set. Furthermore, it should be possible for the brokers (intermediate AuthZ Servers) to allow end-to-end (direct) authentication and authorization; this means that whatsoever two AuthZ Servers must be allowed to interact directly.
- General concerns about authorization protocol's security: It is responsibility of the protocol designers to elaborate their protocols to be resilient against well-known types of attacks.

A final word about this framework: an architecture based on this recommendation has been proposed in [3] for a policy-based admission control mechanism.

2.2. Conceptual Grid Authorization Framework and Classification.

This document [4] is aimed to the Grid community however its ideas can be extended to several distributed systems even as Web Services. The authors introduce basic concepts and models of authorizations, specifying a conceptual Grid AuthZ framework and classifying existing and proposed authorization mechanisms with regard to it. Also this document considers the various authorization concepts and their relationships and describes a framework that is expected to be general and abstract enough to allow for a classification of any type of AuthZ system. Abstract entities and functions are defined and fundamental communication sequences for authorization requests and decisions are shown. As in the case of the previous framework, this document does not consider authentication directly, but is noted as an essential element for the authorization process.

It shall be noticed that this framework is very similar to the one introduced previously, which is not coincidence as the authors adapted and extended it to the Grid environment with concepts that however can be carried back again to several distributed systems.

2.2.1. Authorization Entities.

In principle authorization decisions are made based on authorization information provided by authorities. These authorities must have a direct or a delegated relationship with either the authorization subject, or with the resource that is the target of the request that prompted the authorization, or with both. These relationships may be implemented using a trust mechanism based on some cryptographic method or may be implemented completely off-line.

This observation brings us to the definition of the three basic high level entities involved in authorization:

- *Subject*: An entity that can request, receive, own, transfer, present or delegate an electronic authorization as to exercise a certain right. The subject may define a set of policies that determine how its authorization is used.
- *Resource*: A component of the system that provides or hosts services and may enforce access to these services based on a set of rules and policies defined by entities that are authoritative for the particular resource. Access to resources may be enforced by a Resource itself or by some entity that is located between a resource and the requestor and thus protecting the resource from being accessed in an unauthorized fashion.
- *Authority*: An administrative entity that is capable of and authoritative for issuing, validating and revoking an electronic means of proof such that the named subject of the issued electronic means is authorized to exercise a certain right or assert a certain attribute. Right(s) may be implicitly or explicitly present in the electronic proof. A set of policies may determine how authorizations are issued, verified, etc. based on the contractual relationships the Authority has established. Currently three types of authorities are in common use: Attribute Authorities, Policy Authorities and Identity Authorities.

The component performing the evaluation of the executable policy by computing an authorization decision on behalf of the authorities is sometimes referred to as an AuthZ Server. Typically this entity may make or do a combination of: an authorization decision, an authorization lookup, and the delegation or proxy of an authorization decision to another AuthZ Server.

2.2.2. Authorization sequences.

Entities introduced in this framework can be mapped into its correspondent according to [2]:

- User ? Subject.
- Service Provider ? Resource.
- AuthZ Server ? itself.

Considering this mapping the use cases for authorization sequences are described next.

2.2.2.1. Push Sequence.

With this sequence, the Subject first requests an authorization from an Authority. The Authority may or may not honor the Subject´s request. It then may issue and return some message or secured message that acts as a proof of right (AuthZ Assertion). Typically such an assertion has a validity time window associated with it. The assertion may subsequently be used by the Subject to request a specific service by contacting the

Resource. The Resource will accept or reject the authorization assertion and will report this back to the requesting Subject.

2.2.2.2. Pull Sequence.

In this sequence, the Subject will contact the Resource directly with a request. In order to admit or deny the service request, the Resource must contact its Authorization Authority, which will perform an authorization decision and return a message that obtains the result of this operation. The Resource will subsequently grant or deny the service to the Subject by returning a result message. Later on this report we will explain the Akenti and PERMIS systems, which implement this sequence.

2.2.2.3. Agent Sequence.

Using the agent sequence, the Subject will contact a higher level agent with a request to obtain a service authorization. This agent will make an authorization decision based on the rules established by the authorization authority and if successful it will contact the Resource to provision a certain state as to enable the service. After receiving a successful feedback from the service, the agent will report success to the requesting Subject. This model is relevant to Grid users when requesting a certain QoS from the Grid system (i.e. resource reservation through a scheduler), as only then they may interact directly with the resource to access the service.

2.2.2.4. Hybrid authorization sequence.

Even though the previous three sequences are fundamental, they do not cover all possible authorization situations. Sometimes, when studying a certain sequence, one may find that the framework model does not entirely match one of the above sequences. In situations like this it may be desirable to combine certain basic sequences (i.e. the push and pull models).

2.2.3. Domain considerations.

An administrative domain is a definition of the scope of authority. In Grid environments we frequently see scenarios where there are separate domains for identity, subject attributes, resource policy, and community policy authorities. In some advanced scenarios a *community or virtual organization (VO) domain* is present. A VO domain can provide Authorities that perform privilege management for all the members of a VO. Grid Service Providers may also provide resources to users in multiple VO or home domains enhancing in this way the complexity of the authorization infrastructure required.

Almost in every case contractual relationships (often involving agreements) between the domains of the different Subjects, Authorities and Resources are frequently necessary to enable the acceptance and issuing of authorizations.

2.2.4. Authorization policies and Attributes.

Authorization information such as policies, attributes, identities and environmental parameters are utilized and combined when making authorization decisions. Many policies use the concepts of conditions and actions which have to be evaluated with respect to the actual request, the requesting subject's identity and the attributes this subject holds.

Authorization Attributes are statements about properties bound to an entity that implicitly or explicitly define the entities allowed actions on some resource. Attributes

can be grouped into descriptive and privilege attributes. Descriptive attributes associate a characteristic with an entity, while privilege attributes define directly applicable access rights of an entity with respect to a resource.

2.2.5. Authorization Architecture.

This framework considers an AuthZ architecture consisting of a set of entities and functional components that allow authorization decision to be made and enforced based on attributes, parameters and policies that define authorization conditions. Figure 5 shows precisely an authorization architecture based on the pull model, even though similar structures can be generated for the push, agent and hybrid ones. This architecture will be used to explain the rest of this framework.

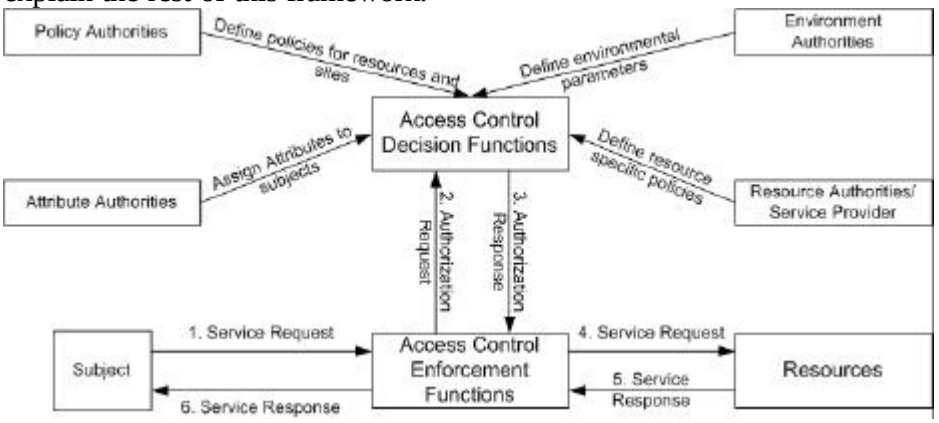

Figure 5. Authorization Architecture based on the pull sequence.

2.2.5.1. Authorization functions.

This framework used the following two access control functions:

- Access control decision function (ADF): Makes authorization decisions about a subject's access to a service. It is equivalent to the Policy Decision Point (PDP) defined in the previous framework (RFC 2904). It is normally part of an AuthZ Server and is independent of the Resource or Application; however it may be co-located with the Access Control Enforcement Function.
- Access Control Enforcement Function (AEF): Mediates access to a resource or service. It is equivalent to the Policy Enforcement Point (PEP) defined in RFC 2904. It is most often either integrated into or located in front of the Resource it protects.

Obviously the ADF, AEF, Subject and Resources may be embedded inside one or more administrative domains in a variety of combinations. In any case, there are three categories of information that may need to be passed between the subject, resource and various attribute authorities:

- Attributes: The subject may get the attributes from the attribute authority, the authority could pass the attributes directly to the relevant ADF, or it could put them in an attribute repository. When the ADF needs an attribute to make and authorization decision, it may get it from the subject either as part of the original request or during a negotiation phase or may pull it from either a local repository, or a repository associated with the attribute authority, the subject

or virtual organization. Security considerations about attributes and its management should be taken into account.

- ?? Policy flow: Policies are typically stored in a repository or provisioned directly to the decision functions by the policy authorities. The transferring of policies has similar requirements to the transferring of attributes as it is imperative to securely establish the authority of the issuer and to protect the integrity of a policy.
- ?? Authorization queries and responses: Authorization requests and responses are similar to attributes in that it is necessary to provide for a secure binding. An AuthZ request must be securely bound to a subject and the subject's service request; also the response must be securely bound to a request and when required also to its originator.

2.2.5.2. Policy Subsystem.

Evaluating a policy is the heart of the authorization decision process. The components responsible for expressing, storing, retrieving and evaluating policies can be thought of as a policy subsystem.

Policy expression is usually done by a policy language which contains the vocabularies to express various policy artifacts. If the policy language is extensible, then domain specific vocabularies and characteristics can be incorporated in the future without requiring fundamental changes. Access policy for resources is written by policy authorities which generally get their authority from the owner of the resource. The policy processing engines are likely to be proprietary to the various systems.

Some authorization decision may be indeterminate because there are conditions in the policy that the ADF cannot evaluate, involving the current state of the resource. In this case the conditions may be passed back to the AEF to evaluate, requiring the AEF to understand the policy language that is used to express these conditions.

Policy repository is also important to consider as once written policies must be stored for use by an ADF. For security and efficiency reasons frequently it is collocated with the ADF.

Finally, to exchange policies a substrate consisting of messages and protocols is required. Similar security considerations as in the case of attributes flow are required for policies transport.

2.2.6. Framework components.

2.2.6.1. Trust Management.

In this framework trust management defines authorities shown in figure 5 and specifies what they should be trusted to do.

Policy and resource authorities both issues policy about resources, but the policy authority operates at a higher level and may issue access control policy for a whole site or VO. Attributes authorities assign attributes to subjects and may belong to the subject's domain or to a VO. Environmental authorities may define things about the resource environment, such as disk usage, or about the distributed environment such as the security of the connection.

Once a VO or resource domain knows how to represent various authorities, it needs to define which ones are to be trusted and for what purposes. Who defines these trust

relationships is determined by the risk management strategy being used. The AEFs need to known which ADFs to trust for authorization decisions.

A final item that comes under the category of trust management is policy about who can create proxies which have all or some of the rights of the delegating entity and who can delegate rights to other entities.

2.2.6.2. Privilege Management.

Privilege management covers the definition, assignment, storage, presentation, delegation and revocation of both privilege and descriptive attributes. For the management of privilege and descriptive attributes there are three distinct phases: granting the privilege, using the privilege, removing the privilege. For privilege attributes there are two primary actors: the authority granting/removing the privilege, and the subject requesting-using the privilege. For descriptive attributes a third actor, the policy authority, is required to associate authorization semantics in the access control policies with the descriptive attribute.

2.2.6.3. Attribute Authorities.

An attribute is granted to some entity by an authority for the relevant home domain of the entity. Sources of authority, their delegates and the domains that will accept the attributes must have a common understanding of the authority's scope. This should be expressed in a privilege management policy, which among others, defines which authority may grant what attributes with what values to what entities. An attribute authority may run on behalf of a number of authorities –delegation-, such as a real or a virtual organization. A method for querying the privilege management policy in order to determine the appropriate source of authority for a given attribute is required. It is important to note that the Source of Authority may delegate portions of its authority to various agents.

2.2.6.4. Privilege assignment.

This operation describes the process of defining who is allowed which access rights. Privileges can be assigned by issuing a policy component describing direct access rights of a subject. The policy rule defining such a privilege typically consists of the three-tuple (subject , resource, action). If scalability is desirable then the presence of a descriptive subject attribute may be required (i.e. roles or group membership); in this case the policy rule consists of a three-tuple (attribute, resource, action) and the attribute is a two-tuple (subject, attribute). Hierarchical role assignment extend this concept even more by allowing for access right inheritance from less privileged to more privileged roles. As part of privilege management also shall be considered the revocation of such.

2.2.6.5. Attribute management.

Attribute assertions are proofs of the right to assert a descriptive attribute or privilege attribute. The list of attributes one has is generated from the list of attribute assertions one possesses and this may allow the subject to act as a source of authority for those attributes within its own domain (even allowing delegation as described by the correspondent attribute authority). These assertions must be stored pending their use in the so-called "attribute repository", which may need special security considerations.

As attributes must be understood by relying parties in often multiple administrative domains, a common schema for its syntax is required. In any case the attribute authority may choose to issue attributes in a variety of ways, but it is recommended to keep them separated from identity tokens (i.e. identity certificates). Finally, these attributes must be

asserted to the ADF by some protocol (special considerations arise in the case of distributed attributes management).

2.2.6.6. Policy management.

Policy management for static resources is commonly solved by the use of a repository. However creating a policy for dynamically created objects is even more challenging and in many cases is preferable to control access to a whole class of objects by a static policy and then just create the new ones within these classes.

Some of the issues that are addresses by policy management are who can create, modify and delete policy for each resource, how quickly policy can be revoked, and where does the ADF find the policy (also considering trust relations). In many cases an additional challenge is raised with distributed policy management (i.e. how to find all the relevant pieces to take a decision, displaying the current policy to the resource owner or to anyone trying to add to the policy).

2.2.6.7. Authorization context.

This consists of those properties of the Authorization Request which are neither provided via Authorization Attributes nor included in Authorization Policies, but which are relevant to the decisions made by the Authorization Server. The Authorization Context may include environment information (i.e. disk usage) and authentication information (i.e. key lengths, etc.).

2.2.6.8. Authorization server.

The Authorization Server is an entity that evaluated authorization requests and issues responses, taking into account relevant attributes, policies and environmental parameters. Commonly the authorization algorithm used takes all or some of the following inputs:

- Nature of request.
- Attributes of requestor.
- Attributes of resources.
- Authorization context.
- Environmental factors.
- Policy statements.

Authorization responses are typically the output from these algorithms and may be language specific.

2.2.6.9. Enforcement mechanisms.

This work is done by limiting the operations performed on resources on behalf of a subject to those permitted by an authoritative entity. A common practice on these mechanisms focuses only on user's access to an application. However this is not completely certain in Grid environment, where applications and services can be provided by software components that are staged by the user to the compute resource and are not necessarily trusted by the resource owners. The resources act as hosting environments for these services, which often are transient, migrating between different resources to meet performance criteria. In other word, it is important to control not only the access of a subject to a service but also the access of a service to its current service hosting environment.

Enforcement functions can either receive the set of authorized operations as part of the service request, or by querying an ADF (it depends on the message sequence being used). In any case, enforcement mechanisms can be characterized in two different groups:

- Application dependent enforcement mechanisms: these are often integrated in the application or service and perform enforcement functions before the application attempts to access underlying operating system resources.
- Application independent enforcement mechanisms: are separate from the service or application and take the approach of running the service in a very constrained execution environment (maybe being enforced by the kernel or by intercepting the respective call before being passed to the operating system).

2.3. Other authorization frameworks.

2.3.1. ISO 10181-3.

This authorization framework [5] defined four roles for components participating in an access request. They are Initiators, Targets, Access Control Enforcement Functions (AEF) and Access Control Decision Functions (ADF). An Internet Attribute Certificate Profile for Authorization provides a means to bind arbitrary attributes to identities.

An Authorization API –called Generic Application Interface for Authorization Frameworks- defines a standardized API for the interface between AEF and the ADF as defined in [2]. The IETF proposed a standard Generic Authorization and Access Control API (GAA API) as a definition of a simple interface between a resource gateway and an authorization module or server and a policy language in which stakeholders express their access requirements to the authorization server.

3. Generic classification of existing AAI foundation technologies.

The previous section introduced basic authorization terminology through the explanation of two conceptual frameworks. However as we mentioned it in the introductory section, a wide range of AAI supporting mechanisms currently exists which complicates the work of the designer or developer when trying to choose the appropriate for its solution. From those base mechanisms we have found a special interest in three categories, as they provide core functionalities for several AAI technologies –current and future-. Using a generalization of the framework presented in [4] this section will cover the former cited categories:

- Assertion oriented mechanisms:
  - Attribute assertions.
  - Authorization assertions.
- Policy expression mechanisms.
- Policy processing mechanisms.

Also we are introducing a new category related with the authentication component of the AAI:

- Identity federation mechanisms.

Other categories strongly depending on the AAI implementation (i.e. architecture, enforcement functions, decision functions and other programming interfaces) will not be presented in this section and instead shall be covered in section 4. In any case it is worth to note that technologies categorized here evolve very fast and in many cases cover more than one aspect of framework [4], so for example one mechanism may be suitable not only to write authorization policies but also may covers privilege management. It is up to the reader planning to design its own AAI to choose those features that best fit his needs.

3.1. Assertion oriented mechanisms.

In this field a lot of work has been done since a few years ago. Worth to mention are the Attribute Certificates as specified in [5] which make a good job when it comes to attribute information exchange between principals, however nowadays they have become limited due in part to their lack of support to express authorization assertions just as other mechanisms can. For this reason we will focus our review on the following:

- OASIS's Security Assertion Markup Language (SAML).
- Web Services Security.

3.1.1. Security Assertion Markup Language.

The OASIS consortium created the Security Assertion Markup Language (SAML) as a XML encoded language based on assertions exchange between clients and servers, so security information (Authentication acts, Attributes and Authorization decisions) about subjects -an entity that has an identity in some security domain- could be used to take the appropriate security decisions. In the following sections we will cover the main features of SAML as published in version 1.1 [6].

3.1.1.1. Use cases and requirements.

Use cases and scenarios defined by the OASIS consortium for SAML include [7]:

- Single Sign-On with the following scenarios:
  - Push;
  - Pull;
  - Third-Party Security Service and
  - User Session.
- Authorization Service with the Application Chain scenario.
- Back Office Transaction, including the Third-Party Security Service.

Also the following requirements were defined for the SAML standard:

- Authentication: SAML should define a data format for authentication assertions, including descriptions of authentication events. This includes time of authentication event and authentication protocol.
- Authorization: SAML should define a data format for authorization attributes of a principal, which are used to make authorization decisions about it.
- Authorization Decision: SAML should define a data format for recording authorization decisions.
- User Session: SAML shall support web user sessions.
- Anonymity: SAML will allow assertions to be made about anonymous principals, where "anonymous" means that an assertion about a principal does not include an attribute uniquely identifying it.
- Pseudonymity: SAML will allow assertions to be made about principals using pseudonyms for identifiers.
- Message: SAML should define a message format and protocol for distributing SAML data.
- Push Message: SAML's messaging protocol should support "pushing" data assertions from an authoritative source to a receiver.
- Pull Message: SAML's messaging protocol should support "pulling" data assertions from an authoritative source to a receiver.

?? Reference: SAML should define a data format for providing references to authentication and authorization assertions.
?? Multidomain: SAML should enable communication between zones of security administration.
?? Single Domain: SAML should enable communication within a single zone of security administration.
?? Signature: SAML assertions and messages should be authenticable.
?? Open: SAML should not be dependent on any particular security or user database format.
?? XML: SAML should be defined in XML.
?? Extensible: SAML should be extensible.
?? Bindings: SAML should allow its messages to be transported by standard Internet protocols, defining bindings to at least the following:
    - Standard commercial browsers.
    - HTTP as a transport protocol.
    - MIME as a packaging protocol.
    - XML Protocol as a messaging protocol.
    - ebXML as a messaging protocol.

The main elements of SAML are its assertions, which will be reviewed next.

3.1.1.2. Assertions.

As mentioned above, SAML security information is expressed in the form of assertions about subjects, allowing security decisions to be taken by the principals involved. An assertion is a package of information that supplies one or more statements made by a SAML authority.

We can identify three different types of assertions:

?? Assertions that express information (when? and how?) about *authentication* acts previously performed by subjects (AuthN).
?? Assertions about *attributes* stating which have been granted to a subject.
?? *Authorization* decision assertions (AuthZ) that states which access to which targets has been granted or denied to the subject.

Each one of these is issued by its respective SAML authority (Authentication Authority, Attribute Authority and Policy Decision Point).

SAML authorities can use various sources of information, such as external policy stores and assertions that were received as input in request, in creating their responses. In such a way, there is defined a *request-response protocol* between clients (assertion's consumers) and SAML authorities (assertion's consumers or producers).

3.1.1.3. Structure.

The outer structure of an assertion is generic, providing information that is common to all of the statements within it. Within an assertion, a series of inner elements describe the authentication, authorization decision, attribute, or user-defined statements containing the specifics. The issuer's name, date and time of issue, an assertion ID and finally the version of SAML that the assertion conforms to, must accompany every assertion. It is even possible that a single assertion contains several different internal statements about authentication, authorization, and attributes.

3.1.1.4. Authorization Decision Types.

A SAML authority is able to generate one of the three following responses to an Authorization decision statement:

- Permit: The specified action is permitted.
- Deny: The specified action is denied.
- Indeterminate: The SAML authority cannot determine whether the specified action is permitted or denied. This answer is used in situations where the SAML authority requires the ability to provide an affirmative statement that is not able to issue a decision.

In some cases, the validity of the assertion is dependent on the sub-elements and attributes provided (mainly when the `<Conditions>` element is present in the structure).

3.1.1.5. SAML signed messages.

SAML assertions, protocol request and response messages may be digitally signed, which bring all the usual benefits from this feature (integrity, non-repudiation and authentication). However the OASIS group has identified two scenarios where signing may not be required:

- Signature inheritance or nesting: i.e. a non-signed assertion contained into a signed protocol response message.
- SAML authorities trust relations: i.e. a SAML relying party obtaining a message directly from a SAML responder through a secure channel.

XML Signatures are intended to be the primary SAML signature mechanism, but the specification attempts to ensure compatibility with profiles that may require other mechanisms. At this point we shall notice that XML Signatures used in SAML messages are incompatibles with the XAdES format.

3.1.1.6. Statements.

A SAML statement is an extension point that allows other assertion-based applications to reuse the SAML assertion framework. It contains the `<Subject>` element that allows a SAML authority to describe a subject (i.e. the subject identification by its name and security domain) or even to supply data that allows its authentication (i.e. protocol used to authenticate the subject). We can identify three types of statements:

- Authentication: This element describes a statement by the SAML authority, asserting that the statement's subject was authenticated by a particular means at a particular time.
- Attribute: Describes a statement by the SAML authority asserting that the statement's subject is associated with the specified attributes. The value of a specified attribute allows any well-formed XML to appear as the content of the element.
- Authorization decision: This element describes a statement by the SAML authority asserting that a request for access by the statement's subject to the specified resource has resulted in the specified authorization decision, on the basis of some optionally specified evidence.

3.1.2. Web Services Security.

This document proposed by IBM and Microsoft [8], introduces a technical strategy and roadmap whereby the industry can produce an implement a standards-based architecture that is comprehensive yet flexible enough to meet the Web services security

needs or real business. In that paper is presented a broad set of specifications that cover security technologies including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation and auditing across a wide spectrum of application and business topologies. These specifications provide a framework that is extensible, flexible, and maximizes existing investments in security infrastructure.

The proposal in this document is based on [9], which defines the core facilities for protecting the integrity and confidentiality of a message, as well as mechanisms for associating security-related claims with the message. Outlined as a comprehensive and modular solution, this document when implemented will allow customers to build interoperable and secure Web Services that leverage and expand upon existing investments in security infrastructure while allowing them to take full advantage of the integration and interoperability benefits Web service technologies have to offer. Also positive will result the integration through the abstractions of a single security model enabling organizations to use their existing investments in security technologies while communicating with organizations using different technologies.

3.1.2.1. Mechanisms.

To address the challenges presented previously, this document proposes an evolutionary approach to creating secure, interoperable Web Services based on a set of security abstractions that unify formerly dissimilar technologies. This enables specializations to particular customer requirements within an overall framework while at the same time permitting technologies to evolve over time and be incrementally deployed.

The proposed specification is layered above [9], shown in figure 6 and can be summarized as follows:

- *WS-Security:* describes how to attach signature and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages.
- *WS-Policy*: will describe the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints.
- *WS-Trust*: will describe a framework for trust models that enables Web Services to securely interoperate.
- *WS-Privacy*: will describe a model for how web Services and requesters state privacy preferences and organizational privacy practice statements.
- *WS-SecureConversation*: will describe how to manage and authenticate message exchanges between parties including security context exchange and establishing and deriving session keys.
- *WS-Federation*: will describe how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.
- *WS-Authorization*: will describe how to manage authorization data and authorization policies.

Even though this architecture is still in development, the proposal for *WS-Authorization* is planned to describe how access policies for a Web service are specified and managed. In particular it will describe how claims may be specified within security tokens and how these claims will be interpreted at the endpoint. This specification will be designed to be flexible and extensible with respect to both authorization format and

authorization language. This enables the widest range of scenarios and ensures the long-term viability of the security framework.

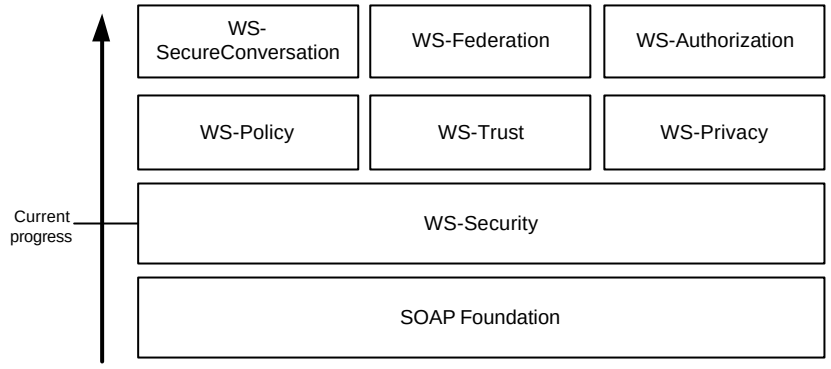

Figure 6. Web Services Security Specifications.

3.2. Policy expression.

Policy expression or policy writing as we shall call it along this section relies in the use of a language flexible, extensible and able to be understood by human and computers so its rules can provide the same functionalities to the supporting AAI. A few years ago Abstract Syntax Notation 1 (ASN.1) was widely used to accomplish this task, however nowadays the eXtensible Markup Language or XML has superseded it in most modern policy writing tools.

Extracted from [10] next subsection covers the basic advantages of using XML for policy writing. We will also present a second syntax used for policy specification (i.e. in the SPOCP Authorization Server) called S-expressions in subsection 3.2.2.

3.2.1. XML for Policy Representation.

Extensible Markup Language (XML) is a rapidly maturing technology with powerful real-world applications, particularly for the management, display and organization of data. XML is a technology concerned with the description and structuring of data and it is a subset of Standard Generalized Markup Language (SGML), with the same goals, but with much less complexity. XML is not a language but a standard for creating languages that meet the XML criteria. It describes a syntax used to create proprietary languages or security policies.

Data is separated from presentation in XML. XML structures the data, while style sheets format the data presentation making it easier to use for multiple purposes. The same stylesheet can be used with multiple documents to create a similar appearance among them. Alternatively, multiple stylesheets can be applied to an XML document to provide different forms of presentation of the data. There are a variety of languages that can be used to create stylesheets such as Extensible Stylesheet Language Transformations (XSLT).

3.2.1.1. XML DTDs and Schemas.

XML includes two methods of checking the validity of an XML document: *document type definitions* (DTDs) and *schemas.* A document is valid if its XML content complies with a definition of allowable elements, attributes and other document pieces. By utilizing special 'Document Type Definition' syntaxes or DTDs, it is possible to check the content of a document type with a special parser.

A schema is the XML construct used to represent the data elements, attributes, and their relationships as defined in the data model. By definition, a DTD and a schema are very similar. However, DTDs usually define simple, abstract text relationships, while schemas define more complex and concrete data and application relationships. A DTD doesn't use a hierarchical formation, while a schema uses a hierarchical structure to indicate relationships. XML schema definitions are also commonly referred to as XSD.

3.2.1.2. XSLT.

XSL is used to create stylesheets. An XSL engine uses these stylesheets to transform XML documents into other document types, and to format the output.

Extensible Stylesheet Language Transformations (XSLT) is a language which can transform XML documents into any text-based format, XML or otherwise. Stylesheets define the layout of the output document and the location of the data in the source document. That is, "retrieve data from this place in the input document; make it look like this in the output". In XSL parlance, the input document is called the source tree, and the output document the result tree.

3.2.1.3. Advantages of XML for the Policy Specification Language.

Some of the advantages that would accrue by using XML for writing an AuthN or AuthZ policy are as follows:

- **Tools:** Use of XML for specification of the AAI policy lends itself to be used with the freely available, verified, tested and user-friendly tools. These tools include among others, XML editors, parsers, validators, translators etc. The availability of such tools and the extensive use of XML in modern communication protocols and other programs will enable users to manipulate XML files easily. Wide availability of such tools will also help in creating and maintaining the policy files over diverse systems without the need for an application specific editor.
- **Platform Independence:** It is possible to edit, maintain and distribute the XML policy file across different OS platforms.
- **Single Data Multiple Presentation:** Once we represent the policy in an XML format it is possible to extract relevant information and present it in different forms that are more intuitive and useful to the administrator or the user. XSLT style sheets can be written and associated with the policy file to generate different presentation formats. Presenting it in a more understandable, graphic format will help the administrator identify any inaccuracies, inconsistencies, or contradictions in the policy file. Intelligent agents can also be written to audit the policy file and signal the administrator for errors in the policy file.
- **Consistency and Accuracy:** XML schemas and/or DTDs can be used to validate the XML file to see if it matches our specifications. Validating the policy file with a well-defined schema will enable errors to be picked up. This will trap all errors without having to go through the entire file manually. The use of generic schema generators and validators only makes this an easier task. This will also support users in their verification of policy files received across the networks.
- **Extensible Format:** An XML format will allow the extension of the policy file to include new constructs. Additional tags can be defined for elements and attributes as and when the need to incorporate them arises. This would not require changes to the application code as long as the structure of the document is maintained.

?? **Ease of Use:** The hierarchical nature of XML layout results in an easy to use and easy to manipulate format. It makes the file more modular and more easily understandable.

?? **Semantic Content Use:** The semantic content of the policy file enables future deployment of intelligent agents or roaming agents that can read policy files and report problems, and that can resolve conflicts between multiple systems by highlighting for instance the difference in the policies between them.

3.2.2. S-Expressions.

Introduced in [11], S-expressions are data structures suitable for representing other arbitrary complex data. Here are the design goals for S-expressions:

?? Generality: S-expressions should be good at representing arbitrary data.

?? Readability: it should be easy for someone to examine and understand the structure of an S-expression.

?? Economy: S-expressions should represent data compactly.

?? Transportability: S-expressions should be easy to transport over communication media (such as email) that are known to be less than perfect.

?? Flexibility: S-expressions should make it relatively simple to modify and extend data structures.

?? Canonicalization: it should be easy to produce a unique "canonical" form of an S-expression, for digital signature purposes.

?? Efficiency: S-expressions should admit in-memory representations that allow efficient processing.

Informally, an S-expression is either:

?? An octet-string, or

?? a finite list of simpler S-expressions.

An octet-string is a finite sequence of eight-bit octets. There may be many different but equivalent ways of representing an octet-string (i.e. hexadecimal, Base-64, etc.). A list is a finite sequence of zero or more simpler S-expressions. A list may be represented by using parentheses to surround the sequence of encodings of its elements, as in:

(abc (de #6667#) "ghi jkl")

Just as with octet-strings, there are several ways to represent an S-expression. Whitespace may be used to separate list elements, but they are only required to separate two octet strings when otherwise the two octet strings might be interpreted as one, as when one token follows another. Also, whitespace may follow the initial left parenthesis, or precede the final right parentheses.

3.2.2.1. Representation types.

There are three types of “representations” for S-Expressions:

?? Canonical representation: This is used for digital signature purposes, transmission, etc. It is uniquely defined for each S-expression and is not particularly readable, but that is not the point. It is intended to be very easy to parse, to be reasonably economical, and to be unique for any S-expression. The "canonical" form of an S-expression represents each octet-string in verbatim mode, and represents each list with no blanks separating elements from each other or from the surrounding parentheses. The following is an example of this representation:

(6:issuer3:bob)

?? Basic transport representation: There are two forms of the "basic transport" representation:
   - o The canonical representation
   - o An RFC-2045 Base-64 representation of the canonical representation, surrounded by braces.

   The transport mechanism is intended to provide a universal means of representing S-expressions for transport from one machine to another. Here is one example of an S-expression represented in basic transport mode:

(1:a1:b1:c)

?? Advanced transport representation: The "advanced transport" representation is intended to provide more flexible and readable notations for documentation, design, debugging, and (in some cases) user interface. The advanced transport representation allows all of the representation forms described above, include quoted strings, base-64 and hexadecimal representation of strings, tokens, representations of strings with omitted lengths, and so on.

This draft paper also introduces a syntax for each one of these representations and also some examples to optimize its implementation.

3.3. Policy processing.

Even though each AAI system may implement its own decision functions (ADF) to process the authentication or authorization policies, it is convenient to choose a convenient expression language (see previous subsection) and access control decision language, which will be categorized in this section as a *policy processing language.* This kind of languages specifies not only a syntax but also a complete semantic to evaluate an AuthN or AuthZ query into the AAI.

We review in this section the main features of the following policy processing languages:

?? eXtensible Access Control Markup Language (XACML).

?? KeyNote.

?? PONDER.

3.3.1. eXtensible Access Control Markup Language (XACML).

XACML is an OASIS standard that describes [12] both a policy language and an access control decision request/response language (both written in XML). The policy language is used to describe general access control requirements, and has standard extensions points for defining new functions, data types, combining logic, etc. The request/response language allows to form a query to ask whether or not a given action should be allowed, and interpret the result. The response always includes an answer about whether the request should be allowed using one of four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, so a decision cannot be made) or Not Applicable (the request can't be answered by this service).

In addition to providing request/response and policy languages, XACML also provides the other pieces of these relationships, namely finding a policy that applies to a given request and evaluating the request against that policy to come up with a yes or no answer. The advantages of XACML over other similar mechanisms are:

?? Standard: This allows XACML to easily interoperate with other applications using the same standard language.
?? Generic: This means that rather than trying to provide access control for a particular environment or a specific kind of resource, it can be used in any environment. One policy can be written which can then be used by many different kinds of applications, and when one common language is used, policy management becomes much easier.
?? Distributed: Means that a policy can be written which in turn refers to other policies kept in arbitrary locations. The result is that rather than having to manage a single monolithic policy, different people or groups can manage sub-pieces of policies as appropriate, and XACML knows how to correctly combine the results from these different policies into one decision.
?? Powerful: Even thought it can be extended, the base language already supports a wide variety of data types, functions, and rules about combining results of different policies.

3.3.1.1. Policy and PolicySet.

At the root of all XACML policies is a Policy or a PolicySet. A PolicySet is a container that can hold other Policies or PolicySets, as well as references to policies found in remote locations. A Policy represents a single access control policy, expressed through a set of Rules. Each XACML policy document contains exactly one Policy or PolicySet root XML tag.

Because a Policy or a PolicySet may contain multiple policies or Rules, each of which may evaluate to different access control decisions, XACML needs some way of reconciling the decisions each makes. This is done thorough a collection of Combining Algorithms representing a different way of combining multiple decisions into a single decision. There are Policy Combining Algorithms (used by PolicySet) and Rule Combining Algorithms (used by Policy). The former are used to builds up increasingly complex policies, and while there are seven standard algorithms, a developer may design its own.

3.3.1.2. Targets and Rules.

A Target is basically a set of simplified conditions for the Subject, Resource and Action that must be met for a PolicySet, Policy or Rule to apply to a given request. These use boolean functions to compare values found in request with those included in the Target. If all the conditions of a Target are met, then its associated PolicySet, Policy, or Rule applies to the request. In additions to being a way to check applicability, Target information also provides a way to index policies, which is useful if you need to store many policies and the quickly sift through them to find which ones apply. Note that a Target may also specify that it applies to any request.

Once a Policy has been found and verified to apply to a request, its Rules are evaluated. A policy can have any number of Rules which contain the core logic of an XACML policy. The heart of most Rules is a Condition, which is a boolean function. If the Condition evaluated to true, then the Rule's Effect (a value of Permit or Deny that is associated with successful evaluation of the Rule) is returned. Evaluation of a Condition can also result in an error (Indeterminate) or discovery that the Condition doesn't apply to the request (NotApplicable). A Condition can be quite complex, built from an arbitrary nesting of non-boolean functions and attributes.

3.3.1.3. Attributes, Attribute Values and Functions.

Attributes are named values of known types that may include an issuer identifier or an issue date and time. Specifically, attributes are characteristics of the Subject, Resource, Action or Environment in which the access request is made. A Policy resolves attribute values from a request or from some other source through two mechanisms: the AttributeDesignator and the AttributeSelector. An AttributeDesignator lets the policy specify an attribute with a given name and type, and optionally an issuer as well, and then the PDP will look for that value in the request, or elsewhere if no matching values can be found in the request. There are four kinds of designators, one for each of the types of attributes in a request: Subject, Resource, Action, and Environment. Because Subject attributes can be broken into different categories, SubjectAttributeDesignators can also specify a category to look in. AttributeSelectors allow a policy to look for attribute values through an XPath query. A data type and an XPath expression are provided, and these can be used to resolve some set of values either in the request document or elsewhere.

Both the AttributeDesignator and the AttributeSelector can return multiple values (since there might be multiple matches in a request or elsewhere), so XACML provides a special attribute type called a Bag. Bags are unordered collections that allow duplicates, and are always what designators and selectors return, even if only one value was matched. In the case that no matches were made, an empty bag is returned, although a designator or selector may set a flag that causes an error instead in this case.

Once some Bag of attribute values has been retrieved, they need to be compared in some way to expected values to make access decisions. This is done though a powerful system of functions. Functions can work on any combination of attribute values, and can return any kind of attribute value supported in the system. Functions can also be nested, so you can have functions that operate on the output of other functions, and this hierarchy can be arbitrarily complex. Custom functions can be written to provide an ever richer language for expressing access conditions.

3.3.2. KeyNote.

KeyNote [13] is not only a system used to write and process authorization policies, but in the broad sense of the word is a Trust-Management System. This kind of systems provide applications with a standard interface for getting answers to authorizations assertions, and provide users with a standard language for writing the policies and credentials that control what is allowed and what is not. KeyNote provides a simple language for describing and implementing security policies, trust relationships, and digitally-signed credentials. Entities can be identified by cryptographic public keys and can be granted limited authorization to perform specific kinds of trusted actions. When a "dangerous" action is requested by a KeyNote-based application, it submits a description of the action along with a copy of its local security policy to the KeyNote interpreter which "approves" or "rejects" the action according to the rules given in the application's security policy.

It is worth to note that policies are written in a standard language that stays the same across different applications, that is defined outside of the application's code and that can be altered by the responsible user whenever needed. An specially powerful concept in KeyNote is the unification of the notion of security policy with that of security credential. Distributed applications can easily create policies that defer to a remotely-managed

authority, which can change the policy simply by issuing new credential (certificates). In KeyNote, a policy can become a remotely-usable credential simply by being signed.

3.3.2.1. Assertions.

The basic unit of KeyNote programming is the assertion (policies and credentials) which contains predicates that describe the trusted actions permitted by the holders of specific public keys. KeyNote assertions are essentially small, highly-structured programs. A signed assertion is also called a "credential assertion" which also serves the role of certificates, have the same syntax as policy assertions but are also signed by the principal delegating the trust. The policy and credential language is concise, highly expressive, human readable and writable, and compatible with a variety of storage and transmission media.

An assertion identifies the principal that made it, which other principals are being authorized, and the *conditions* under which the authorization applies. A special principal, whose identifier is "POLICY", provides the root of trust in KeyNote. Assertions issued by this principal are called "policy assertions" and are used to delegate authority to otherwise untrusted principals. The KeyNote security policy of an application consists of a collection of policy assertions.

An important principle in KeyNote's design is "assertion *monotonicity*"; the policy compliance value of an action is always positively derived from assertions made by trusted principals. Removing an assertion never results in increasing the compliance value returned by Keynote for a given query.

A KeyNote assertion contains a sequence of sections called "fields", each of which specifies one aspect of the assertion's semantics. Fields start with an identifier at the beginning of a line and continue until the next field is encountered. All the assertions are encoded in ASCII. Informally, the semantics of KeyNote evaluation can be thought of as involving the construction of a directed graph of KeyNote assertions rooted at a POLICY assertion that connects with at least one of the principals that requested the action.

3.3.2.2. Action Attributes.

Trusted actions to be evaluated by KeyNote are described by a collection of name-value pairs called the "action attribute set". Action attributes are the mechanism by which applications communicate requests to KeyNote and are the primary objects on which KeyNote assertions operate. An action attribute set is passed to the KeyNote compliance checker with each query. The semantics of attribute's named and values are not interpreted by KeyNote itself and are represented by arbitrary-length strings.

The exact mechanism for passing the action attribute set to the compliance checker is determined by the KeyNote implementation. Depending on specific requirements, an implementation may provide a mechanism for including the entire attribute set as an explicit parameter of the query, or it may provide some form of callback mechanism invoked as each attribute is dereferenced.

3.3.3. PONDER.

Ponder [14] is a declarative, object-oriented language that can be used to specify both security and management policies. Key concepts of the language include domains to group the objects to which policies apply, roles to group policies relating to a position in an organization, relationships to define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an

organizational unit such as a department. It supports obligation policies that are event triggered condition-action rules for policy based management of networks and distributed systems.

The next subsections provide a quick review of Ponder's basic features.

3.3.3.1. Domains.

Domains provide a means of grouping objects to which policies apply and can be used to partition the objects in a large system according to geographical boundaries, object type, responsibility and authority or for the convenience of human managers.

Membership of a domain is explicit and not defined in terms of a predicate on object attributes. A domain does not encapsulate the objects it contains but merely holds references to objects. A domain is thus very similar in concept to a file system directory but may hold references to any type of object, including a person.

An advantage of specifying policy scope in terms of domains is that objects can be added and removed from the domains to which policies apply without having to change the policies. Domains have been implemented as directories in an extended LDAP service.

3.3.3.2. Ponder primitive policies.

Authorization policies define what activities a member of the subject domain can perform on the set of objects in the target domain. A positive authorization policy defines the actions that subjects are permitted to perform on target objects. A negative authorization policy specifies the actions that subjects are forbidden to perform on target objects. The existence of negative authorization policies in a system may result in conflicts with positive authorization policies. These conflicts are modality conflicts and can thus be always detected through static analysis of the policy specification. Although this adds the need to analyze policies for conflict detection, this kind of conflicts may however indicate potentially unforeseen problems with the specification.

The language provides reuse by supporting the definition of policy types to which any policy element can be passed as a formal parameter. Also policies can be declared directly without using a type.

Ponder also supports a number of other basic policies for specifying security policy: *Information filtering* policies can be used to transform input or output parameters in an interaction. *Delegation policy* permits subject to grant privileges, which they possess (due to an existing authorization policy), to grantees to perform an action on their behalf. *Refrain* policies define the actions that subjects must refrain from performing (must not perform) on target objects even though they may actually be permitted to perform them. Refrain policies act ac restraints on the actions that subjects perform and are implemented by subjects.

Obligation policies are event-triggered condition-action rules which define the activities subjects (human or automated manager components) must perform on objects in the target domain.

3.3.3.3. Ponder composite policies.

These facilitate policy management in large, complex enterprises. They provide the ability to group policies and structure them to reflect organization structure, preserve the natural way system administrators operate or simply provide reusability of common definitions. This simplifies the task of policy administrators.

Roles provide a semantic grouping of policies with a common subject, generally pertaining to a position within an organization. A role can also specify the policies that apply to an automated component acting as a subject in the system.

A relationship groups the policies defining the rights and duties of roles towards each other. It can also include policies related to resources that are shared by the roles within the relationship. It thus provides an abstraction for defining policies that are not the roles themselves but are part of the interaction between the roles.

Ponder supports the notion of management structures to define a configuration in terms of instances of roles, relationships and nested management structures relating to organizational units. A management structure is thus a composite policy containing the definition or roles, relationships and other nested management structures as well as instances of these composite policies. Ponder also allows the specialization of policy types, through inheritance. When a type extends another, it inherits all of its elements, adds new elements and overrides elements with the same name. This is particularly useful for specialization of composite policies.

3.3.3.4. Additional features.

The class hierarchy of the language allows new policy classes that may be identified in the future to be defined as sub-classes of existing policy classes. The model also provides a convenient means of translating policies to structured representation languages such as XML. The XML representation can then be used for viewing policy information with standard browsers or as a means of exchanging policies between different managers or administrative domains.

Ponder is a stronglytyped language as every identifier and expression can be checked to be type consistent, and errors can be automatically detected if an operator or function is applied to the wrong type of data. Although most of the type checking occurs statically at compile time, Ponder also supports a form of dynamic type checking by accessing type definitions in a type repository. Statically strongly typed is an important characteristic for a policy language. Note that Ponder does not allow casting, and thus avoids the loopholes created because of casting in typed languages. Policies might be used in situations where their validity is important (e.g. security policies). Since program testing is usually not going to be an option in a policy-based system, static type checking is important to capture type inconsistencies that might cause failure of a policy at runtime.

3.4. Identity federation mechanisms.

The problem of authentication in distributed systems as been widely studied just as long as the authorization problem and even its base mechanisms have evolved in a similar way, resulting in several technologies (i.e. PKI, Kerberos, etc.) which resulted appropriate on a per-domain basis, however several problems seemed to arise when inter-domain authentication took place between different networks. One these problems directly related to AAI technologies, reefers to users being in possession of several (and possible different) online identities. Researches found that this problem can be solved with a *Federated Network Identity* model, even ensuring that critical private information is used by appropriate parties and also providing Single Sign-On technology (SSO) through federated identities.

This section will cover two federation mechanisms that may be integrated in current or even future AAIs deployments:

?? Liberty Alliance's Identity Federation Facet.
?? Grid Security Infrastructure Authentication Model.

3.4.1. The Liberty Identity Federation Facet (ID-FF).

The Liberty Alliance Project was born precisely with this main objective in mind, and has implemented the *Liberty Identity Federation Framework (ID-FF)* [15] which will be explained in this section.

3.4.1.1. ID-FF Objectives and architecture.

The Liberty Project was created with the following key objectives:

?? To enable consumers to protect the privacy and security of their network identity information.
?? To enable business to maintain and manage their customer relationships without third-party participation.
?? To provide an open single sign-on standard that includes decentralized authentication and authorization from multiple providers.
?? To create a network identity infrastructure that supports all current and emerging network access devices.

To achieve these goals Liberty proposed (i) the definition of trust relationships between the businesses and (ii) users federating the otherwise isolated accounts they have with these businesses (known as their local identities). This implies the need for a *circle of trust* build as a federation of *service providers* and *identity providers* that have business relationships based on the Liberty architecture and operational agreements, and with whom users can transact business in a secure and apparently seamless environment using the proposed protocols and bindings. This is in a few words the ID-FF architecture.

3.4.1.2. ID-FF Facets.

The Liberty ID-FF user experience has two main facets:

?? Identity Federation: is based upon linking users' otherwise distinct service provider accounts (note that this only occurs the first time a user logs in to a service provider using an identity provider). This account linkage, or identity federation, in turn underlies and enables the other facets of the Liberty ID-FF user experience.
?? Single Sign-On: enables users to sign on once with a member of a federated group of identity and service providers (a member of provider's circle of trust) and subsequently use various websites among the group without signing on again.

Along with these facets, ID-FF architecture and protocols must be capable of performing also the following activities: authentication, use of unique pseudonyms (handles), support for anonymity and global logout (users will have the ability to terminate federations, or defederate identities).

It is important to note that identity federation implies a layered authentication security policy between providers, that's because *ideally* the same trust level must be given to different authentication mechanisms (i.e. certificate based and password based).

3.4.1.3. Identity Federation.

The first time that users use an identity provider to log in to a service provider, they must be given the option of federating an existing local identity on the service provider with the identity provider login to preserve existing information under the single sign-on. It is critical that, in a system with multiple identity providers and service providers, a

mechanism exists by which users can be (at their discretion) uniquely identified across the providers: while multiple identities can be federated to each other, an explicit link exists between each identity forming a *chain* –and thus users can be differentiated-.

3.4.1.4. Identity defederation.

Users will have the ability to terminate federations, or defederate identities. When defederation is initiated at an identity provider, the identity provider is stating to the service provider that it will no longer provide user identity information to the service provider, and that the identity provider will no longer respond to any requests by the service provider on behalf of the user.

On the other hand, when defederation is initiated at a service provider, the service provider is stating to the identity provider that the user has requested that the identity provider no longer provide the user identity information to the service provider and that the service provider will no longer ask the identity provider to do anything on the behalf of the user.

3.4.1.5. Single Sign-On.

Single sign-on is initiated once a user's identity provider and service provider identities are federated. The ID-FF document [15] highlights several additional security issues about single sign-on:

- ?? Orthogonal behavior between authentication mechanisms and single sign-on.
- ?? Identity providers need to maintain authentication state information for principals.
- ?? Security issues related to user's authentication credentials (called artifacts).
- ?? All authentication assertions should include an authentication type that indicates the quality of the credentials and the mechanisms used to vet them. Support for the Liberty specifications is not itself sufficient to ensure effective interoperability between arbitrary identity providers and user agents using arbitrary methods, and must instead be complemented with data obtained from other sources. Note that this may generate the need to develop new mechanisms able to do this security evaluation –multilayered authentication- in an automatic way; we will return later to this problem.
- ?? Mutual authentication between user and provider (identity provider or service provider).
- ?? Liveness validation, which refers to the action of reauthenticate a user depending on the time that has passed between his last authentication event and the moment when an authorization event must take place.
- ?? The security mechanisms enabled in the communication channel between principals (user, identity provider and service provider).

3.4.1.6. Single Logout.

The single logout and related profiles synchronize session logout functionality across all sessions that were authenticated by a particular identity provider. This process can be initiated at either the identity provider or the service provider, both in any case the identity provider will then communicate a logout request to each service provider with which it has established a session for the user. Note that this may generate a lot of traffic on the network.

3.4.2. Grid Security Infrastructure Authentication Model.

One of the best-known security approaches for Grid computing can be found within the Globus Toolkit, a widely used set of components used for building grids. The toolkit, developed by the Globus Project, offers authentication, authorization, and secure communications through its Grid Security Infrastructure (GSI). The GSI uses public key cryptography, specifically public/private keys and X.509 certificates, as the basis for creating secure grids.

Among the GSI's key purposes are to provide a single sign-on for multiple grid systems and applications; to offer security technology that can be implemented across varied organizations without requiring a central managing authority; and to offer secure communications between varied elements within a grid.

GSI offers users a secure authentication option by creating a time-stamped proxy based on the user's private key. Users can not submit jobs to run or transfer data without creating the proxy. Once created, the proxy is used to grant –or deny- access to resources found throughout the grid. Because the proxy is used across the system, this gives the end user the ability to sign on only once. One alternative to this authentication approach is to use GSI-enable OpenSSH, which uses the same authentication mechanism, but this lies outside of the core Globus Toolkit functions.

3.4.2.1. A Grid Security Policy for Authentication.

In paper [16] the authors define a security policy for grind computing as follows:

- The grid environment consists of multiple trust domains.
- Operations that are confined to a single trust domain are subject to local security policy only (no additional authentication is required).
- Both global and local subjects exist. *For each trust domain, there exists a partial mapping from global to local subjects*. This means that each user of a resource will have two names, a global name and a potentially different local name on each resource. The mapping of a global name to a local name is site-specific. The existence of the global subject enables the policy to provide single sign-on.
- Operations between entities located in different trust domains require mutual authentication.
- An authenticated global subject mapped into a local subject is assumed to be equivalent to being locally authenticated as that local subject. In other words, within a trust domain, the combination of the grid authentication policy and the local mapping meets the security objective of the host domain.
- All access control decisions are made locally on the basis of the local subject.
- A program or process is allowed to act on behalf of a user and be delegated a subset of the user's rights. It is also needed to support the creation of processes by other processes.
- Processes running on behalf of the same subject within the same trust domain may share a single set of credentials. Due that Grid computations may involve hundreds of processes on a single resource, this policy component enables scalability of the security architecture to large-scale parallel applications, by avoiding the need to create a unique credential for each process.

3.4.2.2. User and resource proxies.

Grid computations may grow and shrink dynamically, acquiring resources when required to solve a problem and releasing them when they are no longer needed. Each time a computation obtains a resource, it does so on behalf of a particular user. However, it is frequently impractical for that "user" to interact directly with each such resource for the purposes of authentication: the number of resources involved may be large, or, because some applications may run for extended period of time, the user may wish to allow a computation to operate without intervention. Hence, the concept of a user proxy that can act on the user's behalf without requiring user intervention is introduced.

A user proxy is a session manager process given permission to act on behalf of a user for a limited period of time. The user-proxy acts as a stand-in for the user. It has its own credential, eliminating the need to have the user on-line during a computation and eliminating the need to have the user's credentials available for every security operation. Furthermore, because the lifetime of the proxy is under control of a computation, the consequences of its credentials being compromised are less dire than exposure of the user's credentials.

Within the architecture is also defined an entity that represents a resource, serving as the interface between the grid security architecture and the local security architecture.

A resource proxy is an agent used to translate between interdomain security operations and local intradomain mechanisms.

With these concepts in mind the allocation resource protocol for a grid user consists of the following four steps:

i. The user "log on" to the grid system creating a user proxy.
ii. The user proxy can then allocate resources (and hence create processes).
iii. A process created can allocate additional resources directly.
iv. A mapping exists from global to local subjects.

From the federation point of view only the last step shall be detailed in the next section.

3.4.2.3. Mapping Registration Protocol.

A central component of the security policy and the resulting architecture is the existence of a "correct" mapping between a global subject and a corresponding local subject. This conversion is achieved from a global name (i.e. a ticket or certificate) into a local name (i.e. a login name or user ID) by accessing a *mapping table* maintained by the resource proxy. While this table can be created by the local system administrator, this approach imposes a certain administrative burden and introduces the possibility for error. Hence GSI introduced a technique that allows a mapping to be added by a user.

The basic idea is for user to prove that he holds credentials for both global and local subject. This is accomplished by authentication both globally and directly to the resource using the local authentication method. The user then asserts a mapping between global and local credential. The assertion is coordinated through the resource proxy, since it is in position to accept both global and local credentials. It is desirable that this mapping remains in place for the lifetime of either the global credentials or the user's local account. The mapping protocol is only as secure as the local authentication method.

3.4.2.4. Security implications.

Mapping Grid identities to local user IDs is a way to enable a user to have a single Grid sign-on and yet support legacy access control mechanisms on those sites that require it. This implies that a user must have a local ID at all sited that require one, and that the site administrator and the Grid administrator agree on the mapping to be used by the Grid gateway server. There are several security implications [17] raised by this model: it requires users to have local accounts on any machine they want to use (even if these accounts are temporary); it may give the user more access to the host than he needs; it requires the Grid administrators to trust the host's access control and accounting procedures, and the local site to trust Grid Certificate Authority to correctly identify users, and the Grid software to authenticate them.

On the other hand, many existing compute centers require that the user has an account with them and then rely on the underlying OS to do the authorization based on the userid. Also a mechanism for allowing the local administrator to specify trust relations with various CA's and other sited could be used rather than a direct mapping of ID's.

3.4.2.5. The GT3 Security Model for OGSA.

We now turn to the question of how Grid security challenges can be addressed within the context of the Open Grid Services Architecture (OGSA) a set of technical specifications that align Grid technologies with emerging Web services technologies [18]. Web services technologies allow software components to be defined in terms of access methods, bindings of these methods to specific communication mechanisms, and mechanisms for discovering relevant services. While particular mechanisms and methods are not prescribed, some mechanisms are emerging as ubiquitous. In particular, the Simple Object Access Protocol (SOAP) provides a means of messaging using XML envelopes to encapsulate payloads, with HTTP the most commonly used underlying protocol. Another example is the Web Services Description Language (WSDL), which provides a method for expressing operation signatures and bindings to protocols and endpoints in an XML document (groups of operations bundled together to form a "Web service"). OGSA defines standard Web service interfaces and behaviors that add to Web services the concepts of stateful services and secure invocation, as well as other capabilities needed to address Grid-specific requirements that are not relevant for this report. These interfaces and behaviors define what is called a *"Grid service"* and allow users to manage the Grid service's life-cycle, as allowed by policy, and to create sophisticated distributed services. Grid services can define, as part of their interface, service data elements (SDEs) that other entities can (again, subject to policy) query or subscribe to. OGSA introduces both new opportunities and new challenges for Grid security. Emerging Web services security specifications address the expression of Web service security policy (i.e. XACML), standard formats for security token exchange (i.e. WS-Security and SAML), and standard methods for authentication and establishment of security contexts and trust relationships (i.e. WSSecureConversation, WS-Trust). These specifications may be exploited to create standard, interoperable methods for these features in Grid security. But they may, in some cases, also need to be extended to address the Grid security requirements listed above.

4. Review of current some relevant AAI technologies.

The final section of this report aims to present briefly some Authentication and Authorization Infrastructures which we have considered *different* from other technologies in the sense that i) they have implemented or planned original security mechanisms for AuthN and AuthZ, ii) nowadays are used in real-world scenarios and because of that are constantly improved and updated, finally iii) they have been opened (in the broad sense of the word) to the research community so it is easy to find documentation and even test-drive them.

We will review the following AAIs:

?? Akenti.
?? PERMIS.
?? Shibboleth.
?? AA Request-Responder.
?? Cardea.

Obviously there some others AAI initiatives in developing or even in fast deployment out there, so this section may be improved in the near future.

4.1. Akenti.

Akenti [19] is an authorization infrastructure from the Lawrence Berkeley National Laboratory in the USA, which can be considered a trust management infrastructure or even an AAI as most of its security is build around the previous authentication process.

It represents the authorization policy for a resource as a set of (possibly) distributed certificates digitally signed by unrelated stakeholders from different domains. The policy certificates are independently created by authorized stakeholders. When an authorization decision needs to be made, the Akenti policy engine gathers up all the relevant certificates for the user and the resource, verifies them, and determines the users rights with respect to the resource.

4.1.1. Authorization model.

The Akenti model consists of resources that are being accessed via a resource gateway (PEP) by users. These users connect to the resource gateway using the SSL handshake protocol to present authenticated X.509 certificates. The stakeholders for the resource express access constraints on the resources as a set of signed certificates, a few of which are self-signed and must be stored locally or remotely but in a secure way. These certificates express the attributes a user must have in order to get specific rights to a resource (called *Use-condition certificates*), who is trusted to create use-condition statements and who can attest to a user's attributes (called *Policy Certificates*). At the time of the resource access, the resource gatekeeper (PEP) asks a trusted Akenti server (PDP), what access the user has to the resource. The Akenti server finds all the relevant certificates, verifies that each one is signed by and acceptable issuer, evaluates them, and returns the allowed access. This authorization model can be seen in figure 7.

4.1.2. Policies.

Akenti policies [20] and [21] are distributed and hierarchical. Each policy compromises two components: Use-condition certificates and Policy Certificates. The former comprise a root policy certificate, and optionally subordinate policy certificates that inherit from the root policy. Akenti sees the target as a tree of resources (like a filesystem with subdirectories). Each of the branches (subdirectories) can have a policy

of its own, but in addition to that the policy of the superior branch (directory) is inherited. Each of the policies can be issued by a different Stakeholder.

A stakeholder is a special kind of authority that is trusted to issues Use-Condition certificates. Each stakeholder can impose his own access control requirements independently of other stakeholders. One of the stakeholders signs the Policy Certificate.

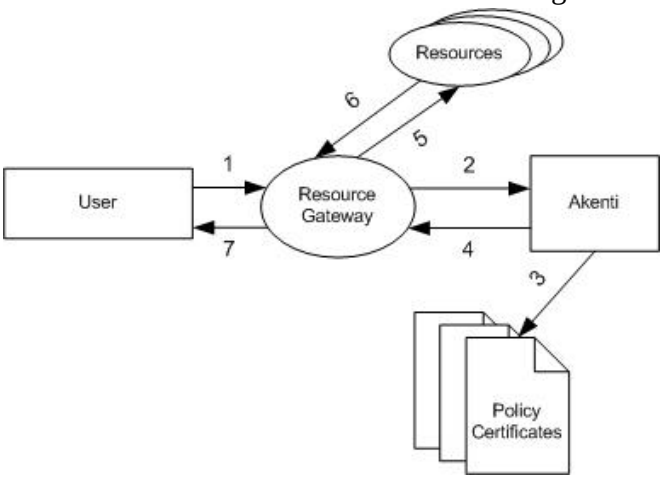

Figure 7. The Akenti Authorization model.

Use-Condition certificates contain the name of the target resource, a condition (which can be a constraint), a critical flag, the authorities of the certificates with the attributes to be matched against these conditions, plus a list of rights/privileges that are granted. Conditions may include identity attributes that the users must have, role or group membership and environmental parameters. Rights are comma separated lists of actions on targets. Action names have to be unique for the whole domain of resources, irrespective of the target type. The authority trusted to issue each attribute value must be specified exactly, but each Use-Condition can include different authorities for each attribute.

In Akenti conditions are placed on which attributes certificates can be trusted and on which attributes have which attributes have which privileges and when (in the Use-Condition certificates).

The attributes issued to users in attribute certificates are independent of each other, and cannot form a role hierarchy. Akenti supports the distributed management of attribute certificates and an external Attribute Authority may assign attributes to users if the Use-Condition certificate lists it under the relevant attribute value.

Akenti's Policy Certificates consists of:

- Name of the resource to which this policy applies.
- List of information about trusted CAs including:
  - Distinguished name.
  - Public key certificate (can be self signed).
  - List of places to search for identity certificates issues by this CA (optional).
  - List of locations where CA stores its CRLs (optional).
- List of Use-Condition issuers (defines the resource stakeholders).
- List of URLs to search for Use-Condition certificates.

?? Optionally URLs to search for user attribute certificates.
?? Maximum time in seconds that any certificates that are used in satisfying conditions for this resource may be cached.

All Akenti credentials are built in a proprietary format using XML syntax.

4.1.3. Decision making and other features.

Decision making is done in a single step, but the system is able to make different types of decisions. The client may ask "What can a user do?", as well as the traditional "Can this user perform this action on this target?".

Akenti always embeds its authorization responses in a Capability Certificate for export back to the client. The Capability Certificate comprises the public key of the user, his DN and his CA's DN, the name of the resource, and the privileges that the user enjoys, optionally with a lists of conditions attached to each of them. The Capability Certificate is then signed by Akenti and given to the client. The user can subsequently present this to a gatekeeper for improved performance. The gatekeeper, which holds the Akenti public key, merely needs to check the signature on the capability, then ask the user to sign a challenge, before granting (or denying) the user access to the resource.

Akenti is written in C++ and its compliance checker can be called either via a function call in the gateway or as a standalone server via TCP/IP.

The previous authentication process is PKI-dependent and the user must present an X.509 public key certificate at authentication time, so the authorization process can be invoked later on.

Trust chains in Akenti begin at the root policy, which is signed by a trusted stakeholder and must be securely configured into Akenti itself.

4.2. PERMIS.

PERMIS [22] is an authorization infrastructure from the EC funded PrivilEge and Role Management Infrastructure Standards validation (PERMIS) project. As in the case of Akenti, PERMIS is also considered a trust management infrastructure. Its compliance checker is invoked as a Java object in the gateway, and credentials are built according to the latest X.509 standard.

Being authentication agnostic, PERMIS leave to the application the task to determine what type of mechanism should be used for this function. Its policies are held in one policy X.509 Attribute Certificate and has implemented role based access controls.
The next subsections will review in more detail PERMIS main features.

4.2.1. Architecture.

The PERMIS architecture comprises a privilege allocation subsystem and a privilege verification subsystem (figure 8). The privilege allocation subsystem is responsible for allocating privileges to the users. There can be any number of these in a full system. The privilege allocation issues X.509 role assignment attribute certificates to users and stores these in an LDAP directory for subsequent use by the privilege verification subsystem. In addition to privilege allocation, each user will also need to be issued with an application specific authentication token.

If a PKI is being used, this will be a digitally signed public key certificate, if a conventional authentication system is being used, it will be a username/password pair. Authorization is performed in an application-independent manner according to the

PERMIS RBAC authorization policy. In this way one policy can control access to all resources in a domain.

4.2.2. The authorization policy.

The PERMIS policy is one subject, and is stored in an LDAP directory as a policy attribute certificate. It supports classical hierarchical RBAC, in which roles are allocated to users and privileges to roles. Superior roles inherit the privileges of subordinate roles in the hierarchy. Multiple disjoint role hierarchies can be specified. PERMIS has a very loose definition of a role; a role may be any attribute assigned to a user, not just a conventional organizational roles. The system also supports the distributed management of attribute certificates, and multiple external SOAs can be trusted to issue roles/attributes.

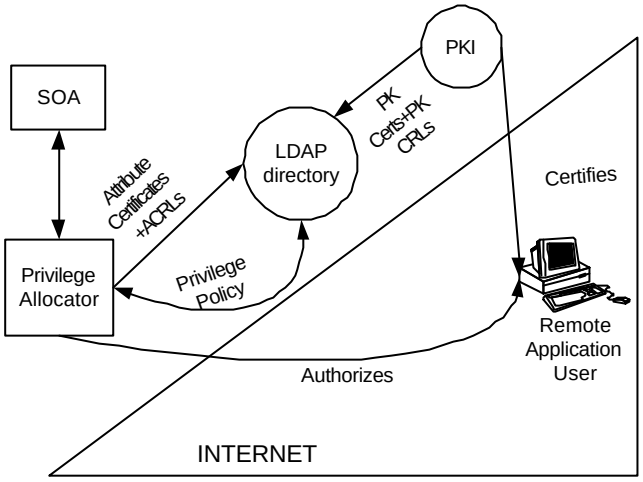

Figure 8. The privilege allocation subsystem.

The policies are kept in the LDAP entry of the policy issuer and different policies are distinguished by their unique object identifier (OIDs). There is no need for the policies to be kept securely, since PERMIS engine validates the policy at run time to ensure it is the correct one. However the name of the SOA has to be securely configured into the PERMIS application at start up. Even though policy hierarchies are not supported by PERMIS, they can be either enforced organizationally by management, or by the application instantiating several PERMIS decision engines, one per level of the hierarchy, and ensuring that each level grants permission.

The PERMIS policy components comprise:

- Policy OID, so the policy can be distinguished among others stored in the SOA's entry.
- Subject domains, specified s LDAP subtrees, which are the subjects who can assert roles.
- Target domains, specified as LDAP subtrees, which are the targets governed by the Target Access Policy.
- List of the distinguished names of trusted external attribute certificate issuing authorities (SOAs) which are treated as roots of the delegation trees.
- Role hierarchy specification (list of the roles as ordered attribute values).

?? Role assignment policy, telling which attribute authorities are trusted to issue which roles to which subject domains, and whether delegation is supported or not.
?? Action policy, saying what the actions and their parameters are, so they can be referenced in the Target Access Policy.
?? Target Access Policy, which specifies the set of roles/attributes required to perform a particular action along with any conditions.

PERMIS policy is specified in XML.

4.2.3. Decision making and other features.

PERMIS operates in multi-step decision making mode (figure 9). In step 1, *getCreds*, the user's credentials are obtained and validated, and roles conforming to the policy are passed back to the calling application for caching. This typically takes place during user login. In step 2, *decision*, the requested action and target are passed, along with the user's validated roles, and a simple boolean decision is returned, either granting or denying access. Step 2 can be repeated as often and as many times as required for different targets and different actions, as the user attempts to perform different tasks. In this model, a user accesses resources via an application gateway. The AEF[1] authenticates the user, and the asks the ADF[2] if the user is allowed to perform the requested action on the particular target resource. The ADF accesses one or more LDAP directories to retrieve the authorization policy and the role Attribute Certificates –ACs- for the user, and bases its decision on these.

The PERMIS API has been specified in Java and due to its structure (the Target and the AEF are co-located or can communicate with each other across a trusted LAN), the authorization token carried between them doest not need any protection at all. Also was assumed that only a single authorization service will be available, and that the API does not need to support the export of AuthZ tokens, as ACs are already in an exportable format.

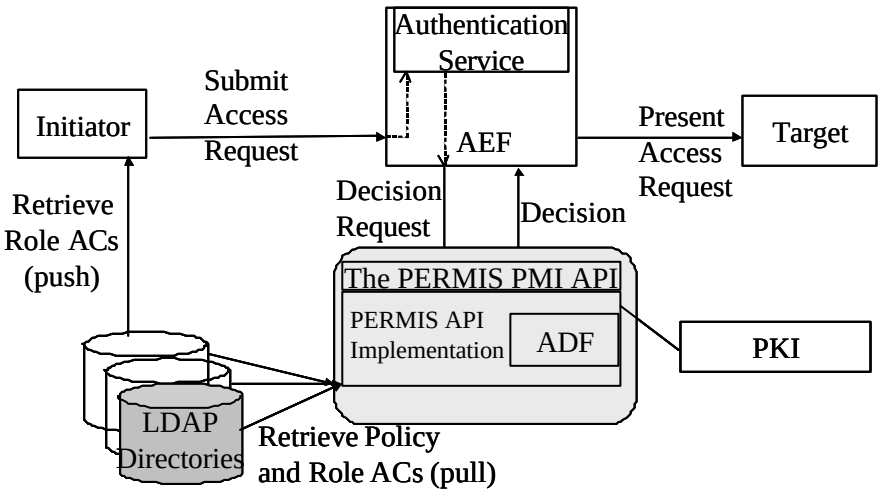

Figure 9. The Privilege verification subsystem.

[1] Access Control Enforcement Function.
[2] Access Control Decision Function.

4.3. Shibboleth.

Shibboleth [23] is a joint project of Internet2/MACA (Middleware Architecture Committee for Education) and IBM. It aims to develop an architecture for standard-based vendor-independent web access control infrastructure that can operate across institutional boundaries.

The focus of Shibboleth is on supporting inter-institutional authentication and authorization for access to web-based applications. The intent is to build upon existing heterogeneous security systems in use on campuses today, rather than mandating particular schemes like Kerberos or PKI based on X.509. Project Shibboleth will produce an architectural analysis of the issues involved in providing such inter-institutional services, given current campus realities; it will also produce a pilot implementation to demonstrate the concepts.

In this system the origin site (where the browser user belongs) is responsible for authenticating the user, and for providing attribute information about the user to the target. The target site (where the web resource manager belongs) is responsible for comparing these statements against the policy rules associated with the desired resource. Both sited will need to satisfy themselves as to the true origin of a request or a set of assertions. Every site that joins Shibboleth can decide to participate as an origin, as a target, or as both. In any case the joining institution must make available a PKI certificate containing the public key of their Shibboleth signing entity. All statements from the institution must be signed using the matching private key. When a site receives a signed request from another site, it must be easy for the recipient to securely obtain the requestor's certificate, in order to validate the signature.

Shibboleth will use pseudononymous identifiers in conversations between the origin and target sites, thus protecting user's privacy. In the case of Single Sign-On (SSO), Shibboleth will not provide such a mechanism but will provide a way for intercampus SSO systems to interoperate between them.

4.3.1. Architecture.

The primary design principles for Shibboleth are [24]:

- ?? No single central piece of infrastructure required, scalable.
- ?? Data protection and privacy are of importance for Shibboleth.
- ?? The User is guided by 'HTTP redirect' from the Resource to the Authentication Server and back to the Resource for the authorization

Shibboleth uses a federated administration, a Resource Owner leaves the administration of User identities and attributes to the User's Home Organization, which is also responsible for providing attributes about a User (possibly but not necessarily including a username) that the Resource Owner can use in making an access control decision when the User attempts to use a Resource. Users are registered only at their Home Organizations, and not at each Resource.

Shibboleth is a system for securely transferring User attributes from the User's Home Organization to the site of the Resource Owner, provided the Resources are accessible via standard web browsers. In addition, Shibboleth enables the Users to decide which information about them gets released to which site. The Users therefore have to balance access and privacy.

All Shibboleth's protocols are SAML compliant.

The major components of Shibboleth are (figure 10):

?? Where Are You From Server (WAYF): Redirects the User to the HS at his/her Home Organization. At least one WAYF server is needed but it may be replicated as desired.
?? Handle Server (HS): Authenticates a local User according to the methods of the Home Organization and provides an opaque handle identifying the User.
?? Attribute Authority (AA): Retrieves the attributes, which a User allows to be given to a Resource (according to the User's Attribute Release Policy) and passes them to the SHAR on behalf of the Resource.
?? Shibboleth Indexical Reference Establisher (SHIRE): Makes sure that the Resource gets a "pointer" (handle) back to the User without requiring more knowledge about the User. In case is missing it refers the User via the WAYF Server back to his/her HS to get one.
?? Shibboleth Attribute Requester (SHAR): Contacts the AA to fetch the available attributes describing the User and passes them on to RM.
?? Resource Manager (RM): decides on access to the Resource based on the information received and where necessary the information about sessions of the same User.

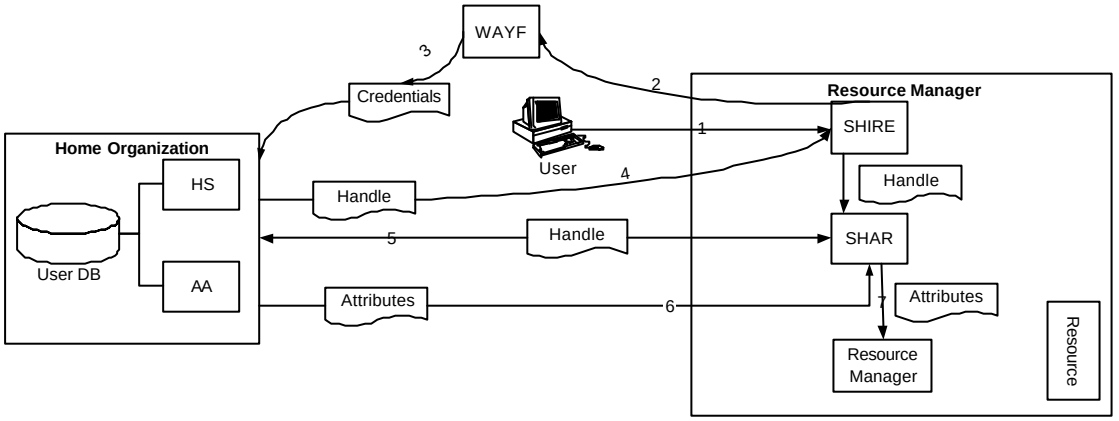

Figure 10. Shibboleth interactions.

4.4. AA Request-Responder (AA-RR).

The purpose of the AA-RR [25] and [26] is the use of some sort of metadata describing the requirements of a certain infrastructure in order to validate (or make at least an assessment on) the interoperability of a certain component with other(s). The availability of this system will translate into:

?? Easier interoperability efforts.
?? A coherent collection of profile data, applicable not only to different pieces of software but to whole infrastructures.
?? Simpler (con-)federation mechanisms, since requirements on syntax and semantics can be published and shared in a normalized way.

The AA-RR has to be built according to the following assumptions:

?? AA interactions are based on the (trusted) exchange of properties (usually called attributes) about users, either humans or application, and resources.

?? There is a common consensus in using open standards to guide AA interactions, not only in the middleware NREN[3] community, but also in the Grid community and the industry.
?? Any standard leaves much open issues, so different profiles can be applicable to accomplish a certain task.

The AA-RR will be able to use specific definitions to simulate the external behavior of different components of several infrastructures in order to assess the interoperability of a certain element with them. This implies the AA-RR can be used to learn of those elements it connects to impersonate, each one corresponding to one of the components in a general AA architecture:

?? Attribute sources.
?? Attribute requesters.
?? Authorization engines.

The broad application fields to which AA interactions are aimed implies a diverse variety of potential protocols to be used, especially if we take into account that currently none of the proposed standards is clearly dominant over the others. In other words, the AA-RR will use a protocol-agnostic approach. The following bindings shall be considered:

?? XML-based protocols (i.e. SAML and XACML) over SOAP/HTTP.
?? The SPOCP protocols, either for authorization engine or attribute requester/source interactions.
?? RADIUS and/or its derivatives, for attribute requester/source interactions.

The AA-RR behavior will be controlled by a set of rules that specify which queries/responses will be accepted and/or sent, which attributes and values will be requested/asserted, which error conditions will be detected, and which final results will yield according to the evolution of the interaction.

4.4.1. Architecture.

The AA-RR will be able to load a profile in XML defining which role (of the tree mentioned) has to play for a certain session, and the relevant parameters governing its behavior:

?? Protocol binding to be applied.
?? Configuration of the protocol binding.
?? Environment settings.
?? Rules controlling the actual evolution of the AA interactions.

Figure 11 depicts the proposed architecture of the AA-RR. The functions of each of the components are as follow:

?? The Configuration Processor reads profile data and instantiates the required components, including the applicable protocol binding.
?? The Profile Manager controls the execution of the different elements in the profile, directly starting the requesters or initiating the responders for the required AA interactions.
?? The Rule Processor applies the rules defined to specify the behavior of the AA-RR for the required interactions.

[3] National Research Educational Networks

?? The Diagnostic Module logs information about the running interactions and about their results.

?? The Protocol Adaptor provides the other components a uniform interface to the different protocol bindings.

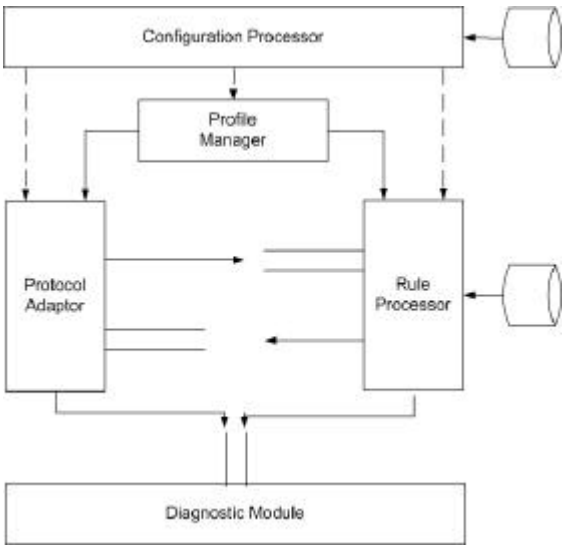

Figure 11. AA-RR architecture.

4.4.2. Configuration.

The configuration of the AA-RR will be made by means of XML files that define the profile to be applied and the defined behaviors by means of *rulesets.* Its contents consists of three sections, each one defined by a corresponding element:

The *Protocol Configuration* section that will contain protocol-specific definitions. Each of these definitions will be held by the corresponding element and its attribute.

The *Environment configuration* section that will contain the following elements:

- The type of entity to be emulated by the AA-RR (attribute source, attribute requester or an authorization engine).
- The absolute path where other files will be read and/or written.
- The detail of which events will be logged during the execution of the AA-RR, and the method to perform it.
- A name (to be used in references) and a path where data bases containing values to be applied during the AA-RR executions can be found.
- The value of the wait timeouts to be applied when rulesets are executed.

The *Rulesets definition* section that will contain a non-empty list of ruleset elements.

Once the configuration processor has loaded and parsed the configuration file, and all the components are appropriately instantiated, the profile manager will take control of the AA-RR, initializing and connecting the event queues of the rule processor and the protocol adaptor, and giving both modules access to the diagnostic queue. Rulesets are executed sequentially, according to the order in which they are referenced in the configuration file. The current execution of the AA-RR finishes when the last ruleset has been run.

4.4.3. Applying rules.

A ruleset definition consists of a non-empty list of *state* elements, which have the attribute *name*, used to assign an identifier to the state, so it can be directly accessed from other states in the ruleset. Only one state is *active* at a given moment. The initial state for a ruleset is the first that is found in the ruleset definition. The contents of a *state* element consist of a non-empty list of *rule* elements.

A *rule* element can have an optional *name* attribute and contain a (single and optional) *conditions* element and a (single and mandatory) *actions* element. This element defines one of the following reactions of the AA-RR:

Finish: indicating that the ruleset must finish issuing the result defined as the value of the attribute.

Next: Identifying a state that will be the next active one.

Send: Identifying a protocol data unit to be sent.

The *conditions* element is used for controlling how the AA-RR behaves according to the events it detects. All constraints defined inside it must be satisfied in order to fire the rule that contains it. A rule with no conditions is always applicable.

4.5. Cardea.

Cardea [27] is a distributed authorization systems, developed as part of the NASA Information Power Grid, which dynamically evaluated authorization requests according to a set of relevant characteristics of the resource and the requester rather than considering specific local identities. Potentially accesses resources within an administrative domain are protected by local access control policies, specified with the XACML syntax, in terms of requester and resource characteristics. Further, potential users are identified by X.509 proxy certificates but only according to the characteristics they can reliably demonstrate. The exact information needed to complete an authorization decision is assessed and collected during the decision process itself. This information is assembled appropriately and presented to the PDP that returns the final authorization decision for the actual access requests together with any relevant details. Figure 12 shows Cardea's architecture.

The systems is currently implemented in JAVA, and contains a SAML PDP, one or more Attribute Authorities, one or more PEP, one or more references to an Information Service (IAS), an XACML context handler, one or more XACML PAP and a XACML PDP. Although all these components may be co-located on the same machine to use local communication paradigms, they may also be distributed across several machines and their functionality exposed as web service portTypes.

Communication between components is specified directly by the XACML and SAML standards, such as the request and response formats for obtaining information. Although XACML and SAML are transport independent, the initial implementation binds these protocols to SOAP.

To preserve message integrity the body of each SOAP message is signed using OASIS XMLSig before transmission to the intended recipient. As each message is signed only after processing is complete, the native format of the signed content is opaque to the signing process. Therefore, no dependencies between signature and content must be supported.

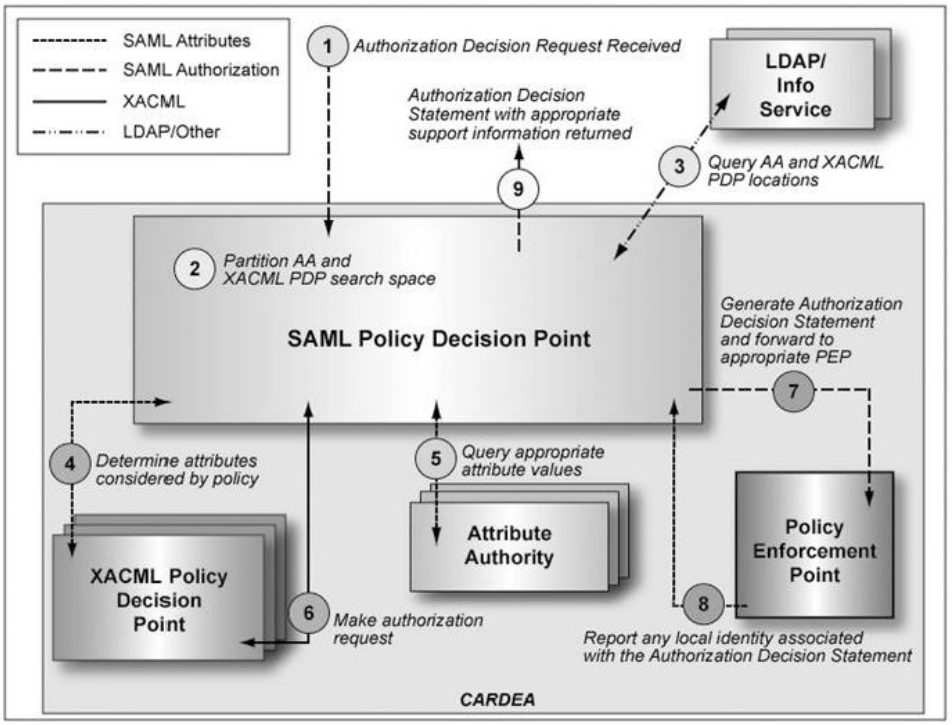

Figure 12. Cardea system architecture.

4.5.1. Systems goals.

Cardea attempts to provide a new resolution to portion of the overall authorization problem that existing system do not adequately address. The first goal is to model distributed authorization in a way that separates local identity from authorization data. Reaching this goal will permit representation and evaluation of access control decisions using strictly the attributes of the entities involved, offering also a method to provide an authorization enforcement point with sufficient information to manage access control with a variety of enforcement mechanisms.

The second system goal is supporting distributed authorization while interoperating with existing security infrastructures. This facilitates standard authorization communication between domains.

4.5.2. Authorization information.

Any characteristic of the subject, the requested resource, the desired action or the current environment may be considered in the authorization decision. The model adopted by Cardea represents these attributes as SAML assertions that are passed between components. Each component is fee to use the assertion data in any capacity it needs, such as transforming it to a different native internal format. However, when communicating the data between components, all characteristics are represented in this common format, regardless of the source or guarantor.

4.5.3. Initiating and enforcing the authorization decision.

Cardea leverages the XACML model for authorization evaluation and SAML for obtaining assertion data used during the evaluation process. Cardea assumes that the SAML PDP that accepts the initial request is responsible for providing the final authorization decision details to the PEP. The SAML PDP depends upon the content of

the initial request to determine the correct XACML PDP to evaluate the request. Then, the flow of communication between entities is specified by these relevant standards.

5. Conclusions.

In this report we have reviewed from conceptual frameworks to current implementations of Authentication and Authorization Infrastructures, covering also some of its fundamental technologies. This has allowed us to identify the following areas around this topic that are still in need of deep research:

- Authentication related:
  - Management of distributed and possibly different user's identities also referred in this report as *identity federation.*
  - Interoperability issues raised between authentication mechanisms implemented on heterogeneous environment (as Grid's virtual organizations).
  - Inter-domain exchange of authentication credentials used for authorization decisions.
  - Trust issues between different PKI-based authentication infrastructures, as in the case of computing Grids.
  - Authentication mechanisms based on *fully expressive* XML Signatures (i.e. OASIS XMLSign and XAdES).
- Authorization related:
  - Secure and flexible Authorization policies that can be created and understood not only by computer systems but also by humans.
  - Improvements on Authorization policies processing engines.
  - Formal frameworks able to validate Authorization policies previous to deployment (this point is particularly sensitive in distributed policy management).
  - Optimization of assertion based protocols (i.e. improving performance, but reducing network usage).
  - Distributed policies management.
  - Dynamic session management systems for authorization policy enforcement that relies only on authorization decision information rather than on unique identity for each potential user.
  - Other management tools (i.e. GUIs for ease the creation of authorization related information for subjects).
  - Accountability issues that may allow for a dynamic resource access system (i.e. based on resource usage and location).

As a general conclusion we shall mention that future efforts shall be aimed to provide interoperability between different technologies, however this should not be an barrier to develop new solutions that may prove more efficient an secure than existing ones. In particular it may be interesting to keep an eye on Cardea, SAML (version 2.0 of its specification has already been released), XACML, WS-Security and XML digital signatures as all of them are mechanisms that will support AAI development for emerging technologies as Grid Services.

6. References.

[1] "Capability-Based Computer Systems". Levy, Henry. Ed. Digital Press. 1984.

[2] "RFC 2904: AAA Authorization Framework". Vollbrecht J, et. al. August 2000.

[3] "RFC 2753: A Framework for Policy-Based Admission Control". Yavatkar, R., et. al. January 2000.

[4] "Conceptual Grid Authorization Framework and Classification". Lorch, Markus. et. al. Global Grid Forum. Rev. November 2004.

[5] "RFC 3218: An Internet Attribute Certificate Profile for Authorization". Farrel S., Housley, R.. April 2002.

[6] "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V1.1". OASIS Security Services Technical Committee. Version 1.1. September 2003.

[7] "Oasis Security Services Use Cases and Requirements Consensus Draft". Platt, Darren. et. al. May 2001.

[8] "Security in a Web Services World: A Proposed Architecture and Roadmap". Joint white paper from IBM Corporation and Microsoft Corporation. Version 1.0 April 2002.

[9] "OASIS Web Services Security: SOAP Messages Security 1.0". Nadalyn, Anthony, et. al. Version 1.0. March 2004.

[10] "An editor for adaptive XML-based policy management of IPSec". Mohan, Raj., et. al. December 2003.

[11] "Network Working Group Internet draft: S-Expressions". Rivest, R. May 1997.

[12] "OASIS eXtensible Access Control Markup Language (XACML) 2.0". Moses, Tim. et. al. December 2004.

[13] "RFC 2704: The KeyNote Trust-Management System Version 2". Blaze, Matt. et. al. Version 2.0 September 1999.

[14] "The Ponder policy based Management toolkit". Damianou, N. et. al. August 2002.

[15] "Liberty ID-FF Architecture Overview". Liberty Alliance Project. Version 1.2. 2003.

[16] "A security architecture for computational Grids". Foster, Ian. et. al. November 1998.

[17] "Security implications of typical Grid Computing Usage scenarios". Humphrey, Marty. Thompson, Mary. 2002.

[18] "Security for Grid Services". Foster, Ian. et. al. 2003.

[19] "Certificate-based Authorization Policy in a PKI Environment". Thompson, Mary. et. al. 2001.

[20] "Akenti Policy Language". Thompson, Mary. et. al. July 2001.

[21] "A comparison of the Akenti and PERMIS authorization infrastructures". Chadwick, David. et. al. 2003.

[22] "The PERMIS X.509 Role based privilege management infrastructure". Chadwick, David. Otenko, Alexander. June 2002.

[23] "Shibboleth Overview and requirements". Carmody, Steven. February, 2001.

[24] "TERENA TF-AACE: Deliverable B.1". López, Diego. et. al. Version 1. July 2003.

[25] “Design of an AA Request-Responder”. Lopez, Diego Rodriguez, Candido. 2004.
[26] “TERENA Task Force on Authentication and Authorization Coordination in Europe (TF-AACE): Final Report”. TF-AACE Work Group. August 2004.
[27] “Cardea: Dynamic Access Control in Distributed Systems”. NASA Advanced Supercomputing Division. November 2003.