SEVENTH FRAMEWORK PROGRAMME
Challenge 1
Information and Communication Technologies

Trusted Architecture for Securely Shared Services

Document type: D7.1

Title:	Design of Identity Management, Authentication and Authorization Infrastructure

Work Package: WP7

Deliverable Number: D7.1

Editor: David Chadwick, University of Kent

Dissemination Level: PU

Preparation Date: 31 December 2008

Version: 1.0

The TAS³ Consortium

Nr	Participant name	Country	Participant short name	Participant role
1	K.U.Leuven	BE	KUL	Coordinator
2	Synergetics nv/sa	BE	SYN	Project Manager
3	University of Kent	UK	KENT	Partner
4	University of Karlsruhe	DE	KARL	Partner
5	Technical University of Eindhoven	NL	TU/e	Partner
6	CNR/ISTI	IT	CNR	Partner
7	University of Koblenz-Landau	DE	UNIKOLD	Partner
8	Vrije Universiteit Brussel	BE	VUB	Partner
9	University of Zaragoza	ES	UNIZAR	Partner
10	University of Nottingham	UK	NOT	Partner
11	SAP research	DE	SAP	Partner
12	Eifel	FR	EIF	Partner
13	Intalio	FR	INT	Partner
14	Risaris	IR	RIS	Partner
15	Kenteq	BE	KETQ	Partner
16	Oracle	UK	ORACLE	Partner
17	Custodix	BE	CUS	Partner
18	Medisoft	NL	MEDI	Partner

Contributors

	Name	Organisation
1	David Chadwick, Lei Lei Shi, Stijn Lievens	University of Kent
2	Sampo Kellomaki	EIfEL/Symlabs
3	Danny De Cock	K.U.Leuven
4	Marc Santos	University of Koblenz-Landau

Table of Contents

EXECUTIVE SUMMARY **5**

1. INTRODUCTION **6**

2. OVERALL ARCHITECTURAL DESIGN OF THE IDENTITY MANAGEMENT, AUTHENTICATION AND AUTHORIZATION (IDMAA) INFRASTRUCTURE **8**

2.1. FEDERATED IDENTITY MANAGEMENT 8
2.1.1. Identities, Identifiers and Attributes *8*
2.2. Authoritative Sources *9*
2.3. Authentication in FIM *9*
2.4. Authorization in FIM *10*

3. BREAK THE GLASS INFRASTRUCTURE **16**

3.2 RESETTING THE BTG STATE 18
3.3 ADAPTIVE AUDIT CONTROLS 18

4. ATTRIBUTE AGGREGATION INFRASTRUCTURE **19**

4.1. CONCEPTUAL MODEL 19
4.1.1. Link Registration Phase *19*
4.1.2. Level of Assurance *20*
4.1.3. Link Release Policy *21*
4.1.4. Service Provision Phase *22*
4.1.5. Using the LoA in Service Provision *24*
4.2. MAPPING THE CONCEPTUAL MODEL TO STANDARD PROTOCOLS 25
4.2.1. Link Registration Protocol *25*
4.2.2. Service Provision Protocols *26*

5. MULTIPLE POLICY AUTHORIZATION EVALUATION INFRASTRUCTURE **28**

6. DYNAMIC DELEGATION OF CREDENTIALS INFRASTRUCTURE **30**

6.1. REQUIREMENTS FOR WEB SERVICES DELEGATION OF AUTHORITY 30
6.2. DESIGN OF A DELEGATION OF AUTHORITY WEB SERVICE 31
6.3. DESIGN OF A CREDENTIAL VALIDATION SERVICE 35
6.3.1. The Trust Model *35*
6.3.2. The Credential Validation Policy *36*
6.3.3. The CVS functional components *37*

7. DYNAMIC MANAGEMENT OF POLICIES INFRASTRUCTURE **40**

7.1. UPDATING THE COLLABORATION AUTHORISATION POLICY 44

8. ENFORCEMENT OF STICKY POLICIES INFRASTRUCTURE **46**

8.1. THE APPLICATION PROTOCOL ENHANCEMENT MODEL 47
8.2 THE ENCAPSULATING SECURITY LAYER MODEL 49
8.3. THE BACK CHANNEL MODEL 50
8.4. FUNCTIONALITY OF THE PEP 52
8.5. TRUST NEGOTIATION 53

9. EVENT HANDLING INFRASTRUCTURE & ITS APPLICATION TO ADAPTIVE AUDIT CONTROLS **54**

10. AUTHORIZATION ONTOLOGY **55**

10.1. OVERVIEW 55
10.2. THE AUTHORISATION ONTOLOGY MODEL 57

11. CONCLUSIONS **68**

12. GLOSSARY **70**

13. REFERENCES **72**

Executive Summary

This document describes the design of the identity management, authentication and authorization infrastructure which is needed in order to achieve the security, trust and privacy objectives of the TAS^3 project.

Section 2 of this document describes the overall architecture of the identity management, authentication and authorization infrastructure. Section 3 describes the design of the Break the Glass (BTG) infrastructure. BTG allows users who are not normally authorized to access resources, to gain access after first "breaking the glass" in the full knowledge that they will have to answer later to management about this. Section 3 also describes how adaptive audit controls can be supported in order to support BTG policies. Section 4 describes the design of a credential aggregation infrastructure where user credentials can be retrieved, aggregated and validated in dynamically changing environments, even when the user is known by different identities at different identity providers. Section 5 describes the multiple policy authorization evaluation infrastructure which will provide support for multiple authorization policies written in different languages to be evaluated and any conflicts between them to be resolved before the user is granted access to a resource. Section 6 describes the design of the infrastructure for the dynamic delegation of credentials between the various actors of the system. Section 7 builds on section 6 and describes how authorization policies can be dynamically managed & updated by multiple distributed dynamically allocated administrators. Section 8 describes how policies (especially privacy policies) can be "stuck" to information, and transported with the information throughout a distributed system. Section 9 describes the event management infrastructure which is used to support the passing of messages between system components, via the publish and subscribe paradigm. Section 10 describes the ontology for authorization and privacy policies. Section 11 concludes by describing the current limitations in the design to date, and indicating where further work will be done in future iterations of this deliverable, and where future research may still be needed at the end of the TAS^3 project. Section 11 also includes details of the standardization work that we have undertaken in the TAS^3 project in order to ensure that the authorization infrastructure is not only built on existing standards, but also contributes to future standards in this area.

1. Introduction

The overall objectives of WP7, as stated in the Technical Annex of the TAS^3 project [1] are to:

- build a fully dynamic authorization infrastructure that allows credentials to be dynamically created and delegated between users and administrators, and policies to be dynamically managed and updated
- incorporate sophisticated real-life authorization requirements such as Break the Glass policies, dynamic separation of duties, state based decision making and adaptive audit controls
- contribute to international standards development in the area of IdM and authorization protocols and profiles and authorization ontology

The above overall objectives have further been enumerated into the following set of specific objectives:

- allow context dependent and user-controlled credentials to be dynamically created and user controlled credentials to be dynamically delegated between the users,
- allow context dependent credentials and user-controlled credentials to be retrieved, aggregated and validated in dynamically changing environments, even when the user is known by different identities at the different attribute authorities;
- make authorization decisions based on multiple policies, written in different policy languages & provided by multiple policy authorities including the data privacy subject;
- allow authorization policies to be dynamically managed & updated by multiple distributed dynamically allocated administrators;
- support Break the Glass policies which will allow the normal authorization policies to be over-ridden in emergency situations;
- integrate adaptive audit controls into the authorization infrastructure;
- allow history (or state) based authorization decision making to take place in Sun's XACML PDP;
- define an ontology for authorization policies and have it quality assured;
- contribute to open standards development within the scope of this work package;
- ensure that the outputs of this work package are seamlessly integrated into the outputs of all the other work packages.

This document represents the first iteration of the official deliverable D7.1, whose purpose is to describe the design of the identity management, authentication and authorization infrastructure (hereby abbreviated to IdMAA) which is needed in order to achieve the above objectives.

IMPORTANT CAVEAT. *Because this is the first iteration of the design of an extremely complex authorization infrastructure that has never been created before, this first iteration is bound to contain bugs, flaws, omissions or extremely difficult to implement features that will have to be re-designed in subsequent iterations of this document. The reader should be aware that this document represents the output from only the first year of a four year project, and implementation experience gained*

in subsequent years of the project will further ensure that this first iteration can be improved in future versions.

This document is structured as follows. Section 2 describes the overall architecture of the IdMAA. Section 3 describes the design of the Break the Glass (BTG) infrastructure in more detail, which will allow users who are not normally authorized to access resources, to gain access after "breaking the glass". This section also describes how adaptive audit controls can be supported in order to support BTG policies. Section 4 describes the design of the credential aggregation infrastructure in more detail, where user credentials can be retrieved, aggregated and validated in dynamically changing environments, even when the user is known by different identities at the different attribute authorities. Section 5 describes the multiple policy authorization evaluation infrastructure in more detail, which will provide support for multiple authorization policies written in different languages to be evaluated and any conflicts between them to be resolved before the user is granted access to a resource. Section 6 describes the design of the infrastructure for the dynamic delegation of credentials between the various actors of the system. Section 7 builds on section 6 and describes how authorization policies can be dynamically managed & updated by multiple distributed dynamically allocated administrators. Section 8 describes how policies can be "stuck" to information, and transported with the information throughout the distributed system. Section 9 describes the event management infrastructure which is used to support the passing of messages between system components, via the publish and subscribe paradigm. Section 10 describes the ontology for authorization and privacy policies. Section 11 concludes this deliverable, by describing the current limitations in the design to date, where further work will be done in future iterations of this deliverable, and where future research will still be needed. It also includes details of the standardization work that we have undertaken in the project in order to ensure that the authorization infrastructure is not only built on existing standards, but also contributes to future standards in this area.

2. Overall Architectural Design of the Identity Management, Authentication and Authorization (IdMAA) Infrastructure

2.1. Federated Identity Management

2.1.1. Identities, Identifiers and Attributes

A person's identity is made up from a whole series of attributes that characterise him or her[1], such as: their physical characteristics and appearance, their past and present behaviours and reputation, their qualifications and group memberships, the names and identifiers used to label them etc. These identity attributes can be used by service providers (SPs) to grant or deny individuals access to their resources, for example, students from the University of Kent may download journals from the library. Unfortunately (or perhaps fortunately from a privacy perspective) no single person or system knows anyone's complete set of identity attributes, although individuals are most likely to know most of the attributes that serve to identify them. Even then, there are limitations in this, for example, individuals might not know how much others trust them. Invariably then, computer systems typically only hold the partial identities of people i.e. a subset of their digital identity attributes. These computer systems are known as Attribute Authorities (AAs) or Identity Providers (IdPs)[2].

Before proceeding further, we should clarify the difference between an *identifier* and an *identity*. An identifier serves to uniquely identify an individual within one domain or system, as no two individuals within a system can have the same identifier. However, this identifier is only one of the identity attributes that comprise that person's digital identity within the system. Different computer systems know different subset's of a person's identity attributes, but each computer system will have its own identifier which uniquely identifies this individual within this system. An individual whose identity is distributed throughout many systems will therefore have multiple identifiers such as: their passport number, login ID, social security number, national ID, email address etc., which are each unique within their own systems. Some systems may store the identifiers from remote systems, as well as their own. For privacy (and other) reasons, users are typically wary about releasing their identifiers to third parties, since these can uniquely identify them, whereas other identity attributes, such as age, usually cannot (unless the anonymity set is too small). Identifiers are therefore rather special identity attributes since on their own they can always uniquely identify the user.

---

[1] In order to be gender neutral, we will use "them" to refer to him or her in future. We will use *he* in the remainder of this document when we need to refer to a single person, but the person may be male or female.

[2] The difference between an AA and an IdP is that an IdP is an AA that also has the ability to authenticate the user so that the user can login and request his attribute assertions be issued to him in the form of digitally signed authorisation credentials

2.2. Authoritative Sources

Usually attributes have to be conferred on individuals by authoritative sources, known as Attribute Authorities (AAs). Whilst people may be trusted in some situations to assert some of their identity attributes themselves, for example, their favourite drink, they certainly won't be trusted in all situations to assert all of their identity attributes themselves, for example, their qualifications or criminal record. Thus different authoritative sources are usually responsible for assigning different attributes to individuals. For example, the university that one graduated from is the authoritative source of one's undergraduate degree attribute. An identity provider (IdP) is an attribute authority combined with a user authentication service, so that it can authenticate the user, and then issue the user with a digitally signed attribute assertion.

Authoritative sources may remove attributes as well as assign them. For example, a university may remove a degree from a student, if it was subsequently proved that the student had committed plagiarism in their dissertation. Similarly, in the UK, the General Medical Council is the only authoritative source of who is a doctor, and it keeps a register of them. If a doctor commits malpractice, the doctor may be *struck off* the register by the GMC. Thus in federated identity management systems, we cannot rely on the individual to assert his various attributes, otherwise he might lie about his various roles and capabilities, and omit to tell about negative attributes such as the points on his driving license. Similarly we cannot rely on a single identity provider to assert all a user's attributes, but only the ones they are authoritative for. For example, a credit card company would not normally be trusted to assert someone's degree qualification attribute. Consequently a set of authoritative sources may need to be consulted by service providers before the latter grant users access to their resources.

2.3. Authentication in FIM

In a centralised system, the user typically presents their identifier and an authentication token (such as a password or digital signature) to prove that they are entitled to be known by this identifier. The system then associates the user with this identifier and with all the attributes that it holds with this identifier. In a distributed system the user would typically have different identifiers in each local system, so if the user authenticated to one identity provider using his local identifier, this identifier would not be known by and therefore could not be used by the other local systems to grant the user access. When X.509 based PKI systems were first designed, they tried to solve this problem by allocating each user a globally unique identifier (called an X.500 distinguished name) which would be known by all local system and therefore could be used to grant the user access. Since this global identifier was bound to the user's public key in an X.509 public key certificate, a signature created by the user's private key could be used as an authentication token by each local system. One of the reasons this X.509 based identity management system failed was the privacy concerns about everyone knowing everyone else's globally unique identifier.

The breakthrough came when it was realised that a user's identifier did not need to be globally unique, but could remain local to the system that allocated it. Authorisation to use a remote federated system could be granted based on the user's identity attributes, rather than on the use of the identifier. If the identity attributes

are provided by a trusted authoritative source, after the user has successfully authenticated to it, then a service provider can be assured of the identity of the user, even if the user's identifier is unknown. Hence systems such as Shibboleth [1], SAMLv2 [11] and CardSpace [2] were born.

TAS^3 uses this model for authentication and authorisation. The user may authenticate to an IdP, then the IdP will issue the user with a digitally signed attribute assertion which the SP can trust and use for authorisation. Note that TAS^3 is not concerned about what mechanism the IdP uses to authenticate the user. This is a local matter between the IdP and the user, and TAS^3 will not concern itself with authentication issues, other than to use the level of authentication assurance (LoA) (see section 4.1.2.) as a means of the IdP informing the SP about the level of authentication that was used. Alternatively, the user may authenticate directly to the SP, and the SP may then ask the IdP/AA for digitally signed attribute assertions about the user.

2.4. Authorization in FIM

The authorization model paradigm that we have adopted is the well known "Subject – Action – Target" paradigm in which a subject attempts to perform some action on some target resource. We use an enhancement of the ISO Attribute Based Access Control (ABAC) model [2] (see below), in which the subject is granted or denied access to the target resource based on the attributes he possesses. Each subject represents a real world principal, which is the action performer. Subjects are often referred to as users, but they are not limited to human beings. Action is the operation that is requested to be performed on the target. It can be either a simple operation, or a bundle of complex operations that is provided as an integrated set. Target is the object of the action, over which the action is to be performed. A target represents one or more critical resources that need to be protected from unauthorized access.

In centralised ABAC systems, it is the same system that assigns attributes to users as assigns permissions to attributes and grants users access to resources, so there is implicit trust in the users' attributes. However, in federated identity management systems, the systems that assign attributes to users (i.e. the identity providers) are different from and remote to the systems that consume these and grant access to users (i.e. service providers). Thus trust needs to be established between identity providers and service providers.

Users are typically assigned attributes by various attribute authorities (AAs), for example, a degree attribute is assigned by a university, a driving license attribute is assigned by a government driving license authority, an employee attribute is assigned by an employer. Users can also be their own authorities for some of their attributes, for example, *my favourite drink* attribute. Users may claim various attributes when trying to access a target resource, but in a web services distributed system world we cannot assume that all the attributes claimed by a user are rightfully his. Consequently we have enhanced the ISO ABAC model so that subject attributes are presented as digitally signed attribute assertions (which we call authorization credentials) issued to the subject by one or more trusted (in the eyes of the resource owner) Attribute Authorities (AAs) or Identity Providers (IdPs). These trusted issuers are the authoritative sources of subject attributes. We call the service of issuing subject attributes, an authorization Credential Issuing Service (CIS). A corresponding authorization Credential Validation Service (CVS) is introduced at the

target site to validate these credentials and determine which of the attributes can rightfully be claimed by the subject (see Section 6.3). Each resource owner (called the Target Source of Authority in Figure 1) specifies the credential validation policy for gaining access to his resources.

ABAC is a generalization of the well known role based access control (RBAC) model [3], in which a role is not restricted to an organizational role, but can be any attribute of the subject, such as a professional qualification or his current level of authentication (LoA) [4]. Throughout this document when we will refer to a subject's roles, we mean any attributes that have been assigned to a subject. A subject can be the member of zero, one or multiple roles at the same time. Conversely, a role can have zero, one or more subject occupants at the same time. In RBAC a role is associated with a set of privileges, where each privilege is the right to perform a particular action on a particular target. The TAS^3 model is more flexible and allows sets of privileges to be assigned to sets of roles, rather than to single roles, since the latter is too restrictive in practice. For example if project managers have some organizational based privileges, project members have some project specific privileges, and project managers have some higher level project specific privileges, then without the ability to assign the latter to a combination of roles (project member and project manager), a new set of roles have to be specially created for each project manager. Thus each subject is authorised to perform the actions corresponding to his role memberships. Changing the privileges allocated to a set of roles will affect all subjects who are members of the role set (or who have been assigned the set of attributes).

The TAS^3 model supports hierarchical A/RBAC in which roles (or attributes) may be organized in a partial hierarchy, with some being superior to others. A superior role inherits all the privileges allocated to its subordinate roles. For example, if the role Staff is subordinate to Manager, then the Manager role will inherit the privileges allocated to the Staff role. A member of the Manager role can perform operations explicitly authorized to Managers as well as operations authorised to Staff. The inheritance of privileges from subordinate roles is recursive, thus a role r_o will inherit privileges from all its direct subordinate roles r_s, and indirect subordinate roles which are direct or indirect subordinate roles of r_s. Role hierarchies need not apply only to organizational roles, but can apply to any attribute, such as level of authentication or assurance (LoA), where there is a natural precedence in the attribute values, in which a higher value implies the privileges of the lower values. In the LoA case, a user who has been authenticated to LoA value 4 (the highest) can be assumed to inherit the privileges assigned to the lower levels of authentication.

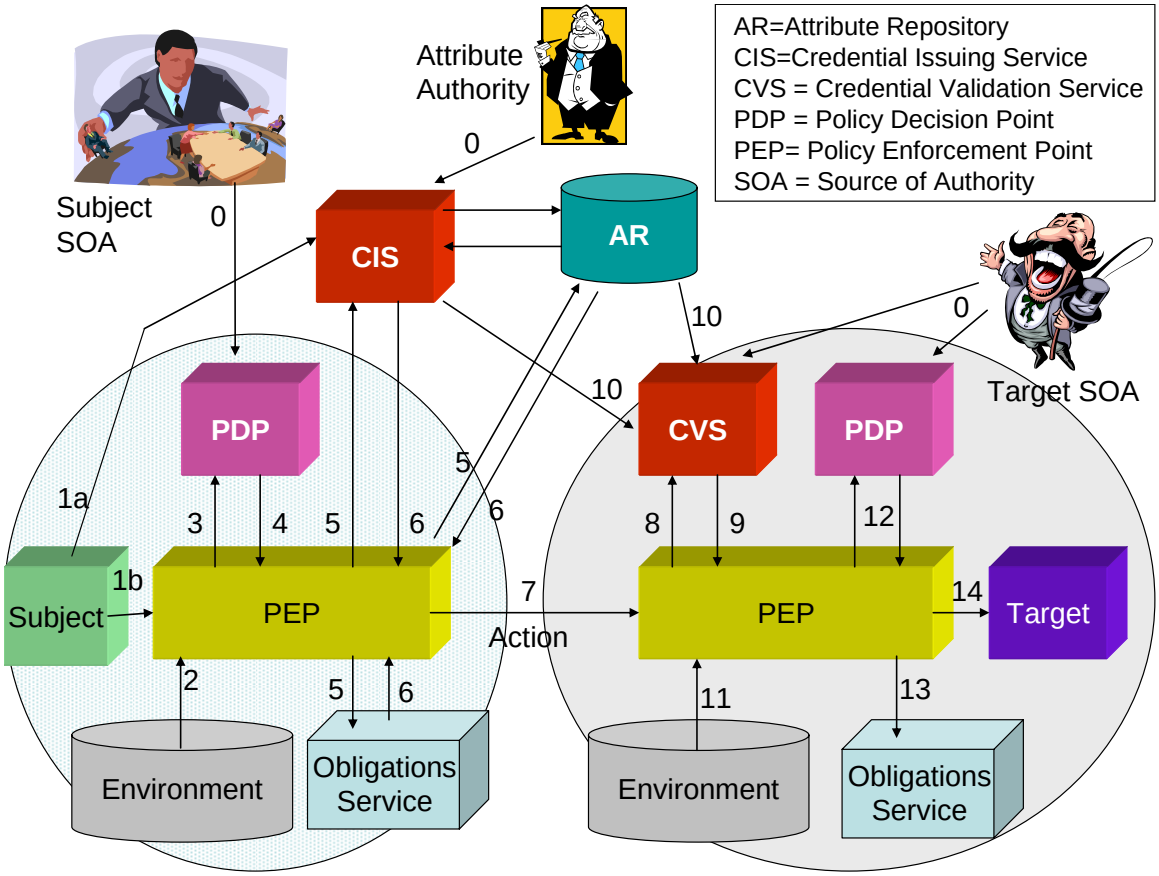


Figure 1. Request Authorization Model

Figure 1 shows our high level conceptual model for the TAS^3 authorization infrastructure when a subject issues an action request to access a remote target. Step 0 is the initialization step for the infrastructure, when the policies are created and stored in the various components. Each subject may possess a set of credentials from many different Attribute Authorities (AAs), that may be pre-issued, long lived and stored in a repository (AR in Figure 1) or short lived and issued on demand (step 1a), according to their Credential Issuing Policies. Section 4 describes the design of the credential aggregation infrastructure in more detail, where user credentials can be retrieved, aggregated and validated in dynamically changing environments, even when the user is known by different identities at the different attribute authorities. The Subject Source of Authority (SOA) may dictate which of the subject's locally issued credentials can leave the subject domain for each target domain. When a subject issues an application request (step 1b), the application independent policy decision point (PDP) can inform the application's policy enforcement point (PEP) which credentials to include with the user's request (steps 3-4). These may be provided by the user along with his request (in step 1b after fetching them in step 1a), or they may be collected from the Credential Issuing Service (CIS) or Attribute Repository by the PEP, either directly or with the help of an Obligations Service (steps 5-6). Obligations are actions that the PEP must enforce along with the decision returned by a PDP. The Obligations Service is the functional component that is responsible for enacting these obligations. The user's request is

transferred to the target site (step 7) where the target SOA has already initialized the Credential Validation Policy that says which credentials from which issuing AAs are trusted by the target site, and the Access Control policy that says which privileges are given to which attributes. The user's credentials are first validated (step 8). This may require the CVS to pull additional credentials from an AA's repository or issuing service (step 10). The valid attributes are returned to the PEP (step 9), combined with any environmental information, such as current date and time (step 11), and then passed to the PDP for an access control decision (step 12). If the decision is granted the user's request is allowed by the PEP (step 14), otherwise it is rejected. In either case, the PDP may also return a set of obligations, which must be enforced along with the access control decision (step 13). In more sophisticated systems there may be a chain of PDPs that are called by a master PDP, with each PDP in the chain holding a different policy possibly written by a different SOA and possibly written in a different policy language (see section 5). In this case the master PDP needs to hold a policy combining policy written by the target SOA, which determines the ultimate response to give to the PEP based on the set of granted, denied or don't know responses returned by the chain of PDPs. Application PEPs however should be shielded from needing to know about this more sophisticated functionality.

Whilst Figure 1 shows many (though not all) of the components of the TAS^3 authorization infrastructure, it does not show all the instances when applications may use it. It only depicts the case when a subject makes an outgoing action request to access a remote resource. In distributed workflow environments, when one service (acting on behalf of a user) makes use of an external service, we require the TAS^3 authorization infrastructure to be called or involved four times by the application during the service invocation. The calling service should check if the outgoing call is authorised, the called service should check if the incoming service request is authorised and if the outgoing response is authorized, and finally the calling service should check if the incoming reply is authorised. This is shown in Figure 2. In this way we can ensure that all applications make full use of the TAS^3 trusted infrastructure, and that all called and calling services can be sure that the other party is behaving in a trusted manner. Section 8 further describes this process, and how it can be utilised to enforce sticky policies.

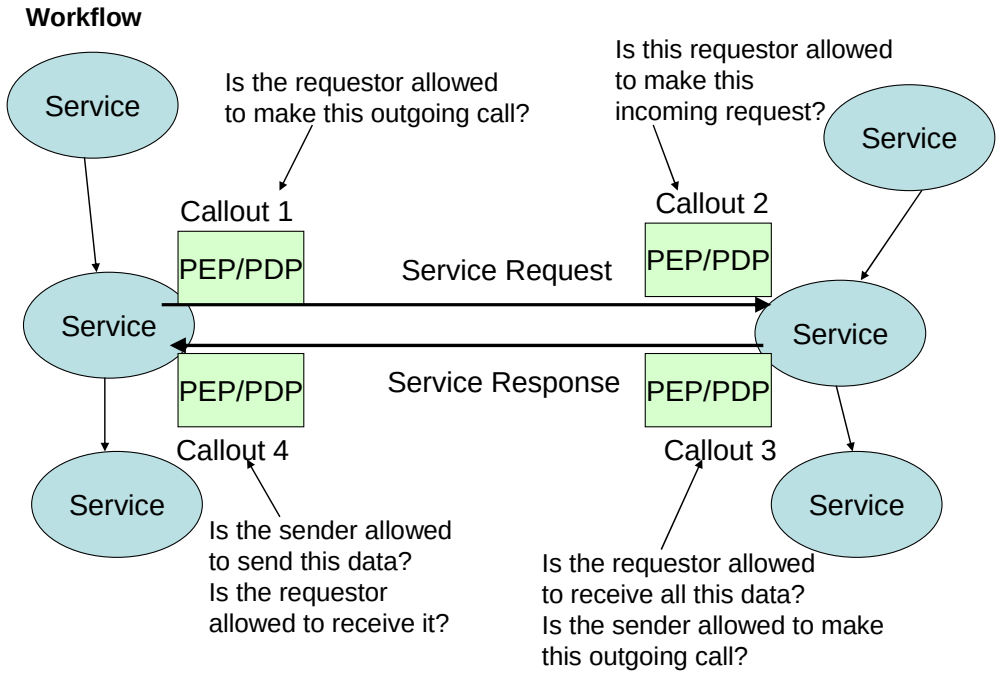


Figure 2 Application Callouts to the TAS³ Authorization Infrastructure

One objective that TAS³ would like to achieve is to make it as easy as possible for applications to implement the TAS³ trusted infrastructure. Therefore the more processing and code that can be removed from the application dependent PEP into the application independent infrastructure, the better. In current state of the art systems the only application independent code is the PDP which makes the authorisation decisions. For this reason, TAS³ has introduced the concept of an application independent PEP (AIPEP). The AIPEP will perform as many of the application independent security functions as possible, that cannot be performed by the PDP. So the AIPEP will call the CVS and CIS instead of the PEP calling them, and the AIPEP will also process any application independent obligations that can be enforced before the PEP is given the authorisation decision. This is shown in Figure 3.

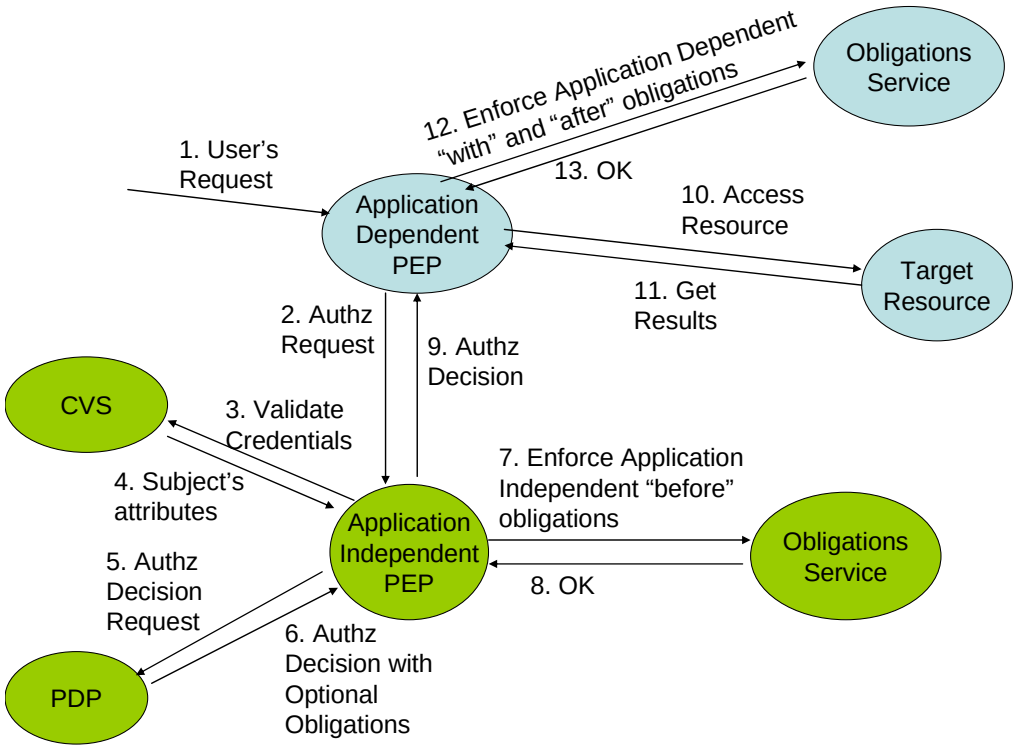


Figure 3. The AIPEP and Obligation Enforcement

By introducing this architecture, we start to differentiate between the types of obligations that can be defined in the authorisation policies. This was first documented by Chadwick et al in [5], who described "before", "with" and "after" obligations. This is now a feature of the latest development work on XACMLv3 [6]. This functionality can be exploited in an application independent way, when defining Break the Glass policies, as described in Section 3.

3. Break the Glass Infrastructure

Break the Glass (BTG) is a functionality that allows a user who is not authorised to access something, to do so in exceptional situations, on the full understanding that he will have to explain this to the appropriate authorities at a later point in time. BTG is useful in at least the following scenarios: the successful mitigation of an emergency situation such as accessing a confidential patient record in a hospital accident and emergency department; the successful mitigation of an exceptional situation such as when the policy writer made a mistake and did not grant access to a user who should have been granted access and it takes a significant amount of time to alter the current active policy; and when the policy writer knows that some other users (but not precisely who) might wish to be granted access under emergency or exceptional situations and wants to be notified when this is the case so that he can monitor how often these situations occur.

We can model BTG policies by introducing a BTG state variable that is normally false, but which switches to true when the glass is broken by an authorised user. We can have a policy rule which says which users are authorised to set the BTG state to true. This could be all users e.g. all employees, or only a small subset of users e.g. emergency room doctors. The policy rule for accessing the resource for this set of users, is conditional on the BTG state being true. We thus have three classes of users in our BTG policy:

- class A) users who are authorised to access the resource regardless of the BTG state,
- class B) users who are only authorised to access the resource if the BTG state is true,
- class C) users who are not authorised to access the resource regardless of the BTG state.

Class B) users should be given two permissions, namely

i) permission to break the glass (set the BTG state to true) for the protected resource, and

ii) permission to access the protected resource when BTG is true.

If we put the BTG state handling code and the Obligation Service code in the application independent layer, by coordinating all this via the AIPEP (as in Figure 3), then we it should be possible to make BTG policy enforcement completely application independent. This of course depends upon what obligations need to be enforced when the glass is broken, in addition to setting the BTG state to true. If all the obligations are application independent, such as switching on the auditing function, then all the obligations can be enforced in the application independent layer. If the obligations are application dependent, such as email the user's manager and charge some account with a certain amount of money, then the obligation enforcement code will need to be application dependent, and it may need to be called by the PEP (as in Figure 3).

A typical example of the BTG infrastructure is given in Figure 4. The user attempts to access a confidential record (step 1) which she is only allowed to access if the glass is broken. The user is denied access (because BTG=false), but the deny decision returns an application dependent obligation to display the Break the Glass

page and policy to the user[3]. The application displays the BTG policy to the user, which warns the user that if the glass is broken, certain actions will be taken (such as logging and notifying her manager). The user may then decide to break the glass, in which case this request will be granted, and obligations will be returned by the PDP to set the BTG state to true, and perform the various policy actions that the user was warned about. After the user is told that the glass has been broken, she may then access the confidential record.

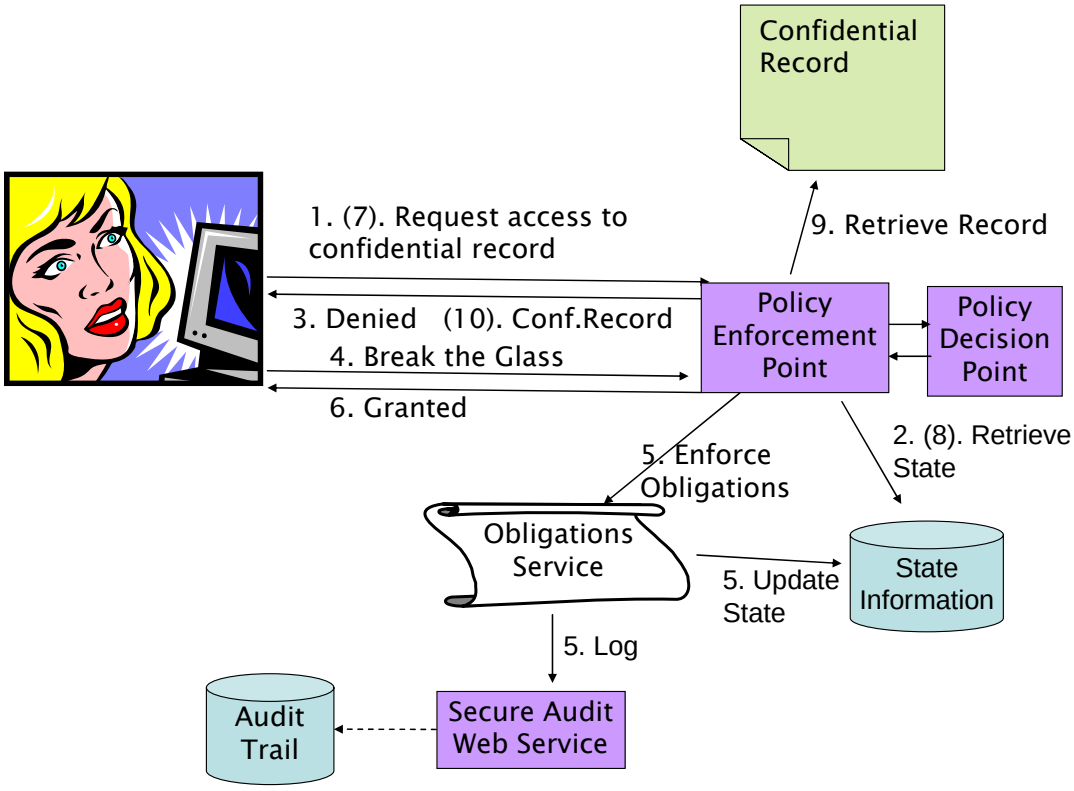


Figure 4. The Break the Glass Infrastructure

Implementing BTG policies in the TAS[3] trusted infrastructure requires the following components:

- policies that can return obligations on grant and deny. Obligations are required to be returned when the user asks to break the glass, and when the user asks to access the confidential resource to which access can only be granted if BTG is true.
- an AIPEP that can manage the BTG state variable. This requires the AIPEP to pick up the value of the BTG variable and pass it to the PDP every time an access request is made, and to call the obligation service to update the BTG state when this is returned. The granularity

[3] Note it is an application dependent matter whether to always display the BTG page and policy to class B and C users who are denied access to the protected resource, or to have an obligation returned for the class B users only, so that only they see the BTG page.

of this state variable is application dependent, which is controlled by the various attributes of the subject. It can be very fine grained e.g. each class B user has their own BTG state variable which they can individually set to true and only they will gain access to the protected resource; or it can be coarse grained e.g. one state variable for all class B users so that if any one of them breaks the glass they all get access to the confidential record. The implementation of the granularity of the state variable is described in [5].

3.2 Resetting the BTG state

A current unresolved issue, which is for further study, is how to reset the BTG state to false. There are a number of options for this. Firstly we could leave it to a human manager to decide when to reset the BTG state to false. This could be for example, after interviewing the subject who broke the glass, to satisfy himself that this was a legitimate use of the mechanism. The manager may also decide that the policy was in error and that the subject should not have had to BTG but rather should have had access to the resource all along, in which case the policy will need to be updated. Secondly the policy could contain a specific obligation that requires the BTG to be automatically re-set to false after a policy specified event occurs. The event could be the passage of a particular fixed period of time, or it could be once the user's current session has terminated, or other predetermined event. This mechanism will require the implementation of an event management system to handle events, their notifications and execution.

3.3 Adaptive Audit Controls

The requirement, from a BTG perspective, is to increase the level of auditing from its current level to an application dependent higher level, from the time the user decides to BTG until the BTG variable has been reset to false by the system. Auditing should then resume at its original level. We can implement the increase in the level of auditing by having an *event logging* obligation that is returned by the PDP at the same time as the obligation to set the BTG variable to true. This obligation will cause the application to increase its level of logging. Reducing the level of auditing once the BTG state is reset to false is more problematical, and is for further study. It relies in part on the resolution of how the BTG state is reset, as described in section 3.2. For example, if an event management system is operational, then when the user breaks the glass, two event logging obligations can be returned, one which causes the logging to be immediately increased, and one which will cause an event to be raised when the BTG state is set to false, this latter event causing the logging to be reset to a specific value.

The granularity of auditing is an application dependent issue. Whether the application will audit every action by every user at the same level of granularity, or individual actions by individual users at different levels of granularity, is still to be determined. Clearly the granularity of the application's logging capability will effect the content of the event logging obligations that are returned by the PDP. It could be that the granularity of the BTG variable should be tied to the granularity of the auditing capability. The precise design of adaptive audit controls is for further study.

4. Attribute Aggregation Infrastructure

One of the current limitations of Shibboleth, CardSpace, Higgins and similar systems is that the user can only select one identity provider, and consequently only a limited subset of his identity attributes. For many web based services this is not enough. Consider wanting to buy a book from Amazon, and in order to get a discount you need to provide proof of your IEEE membership, asserted by IEEE, as well as proof you have a credit card, asserted by Visa. It is currently not possible to do this with CardSpace or Shibboleth since the user can only select one identity card or one identity provider. We need a mechanism to allow a user to select (or aggregate) attributes from multiple identity providers in a single service session, without necessarily having the burden of authenticating to each identity provider during the session.

4.1. Conceptual Model

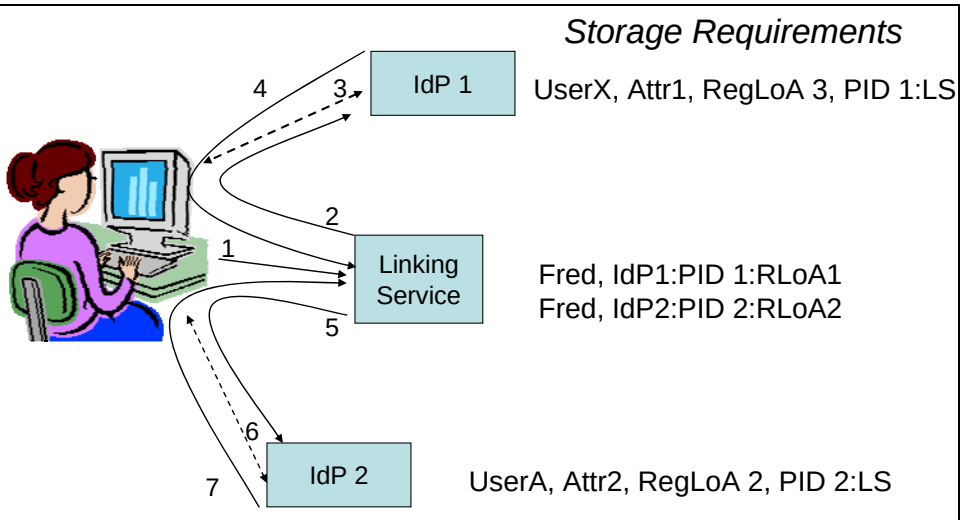
The TAS^3 conceptual model for attribute aggregation assumes that the user is the best (and probably only) person to know who are the authoritative sources for all of his identity attributes. This is a reasonable assumption to make, since most people know who issue their plastic cards, passports, health cards, driving licenses, group memberships etc. We also know that privacy protection is important from a requirements survey that we carried our prior to the design of the TAS^3 system [9]. We have therefore devised a new web service, called a *linking service*, whose purpose is to hold links between the user's various identity providers, and to do this without compromising the privacy of the user. Thus none of the user's (partial) identity providers know about any of the user's other ones. Furthermore, in order to fully protect the user's privacy, the linking service does not know who the user is, or what identity attributes they have. It only knows that some user has links with several different identity providers, and it holds these links on behalf of the user. When the user contacts a service provider for service provision, and they are redirected to their identity provider for authentication, the identity provider returns a pointer to the linking service in its response, which allows the service provider to aggregate the attributes from the various linked identity providers. The identity and service providers trust the linking service to hold these links securely, and to not divulge them to anyone except under the instructions of the user. The user is allowed to say which linked identity providers can be released to which service providers, through an identity provider link release policy (see sidebar).

4.1.1. Link Registration Phase

Here's how the link registration works. The user goes to the web page of his preferred linking service (there can be any number of these on the web). The linking service displays a list of all the identity providers with which it has already established trust relationships. The user selects one that they want to link to another one. The linking service redirects the user to the chosen identity provider, whereupon the user is asked to login and authenticate. The user authenticates using the identity provider's chosen method, by providing their identifier and authentication token. Upon successful authentication, the identity provider creates a random (but permanent) identifier for the user which is to be used solely with this linking service. The identity provider returns an authentication assertion to the

linking service, containing this permanent ID. This assertion effectively says "I have authenticated this user, and they are to be known by you as Permanent ID x (PIDx)." When the linking service receives this message it creates a new link entry for the user in its linking table, assigning the user its own local identifier, say Fred, then displays the list of identity providers again. The user selects another one, is redirected there, authenticates, and the second identity provider returns a different permanent identifier, say PIDy, to the linking service. The linking service adds this link entry to its linking table. The user can perform this identity provider linking as many times as they wish, and the linking service will create a new link table entry for this user each time, as in Table 1. The linking process is shown pictorially in Figure 5.

Each PID is regarded as a secret between the linking service and the issuing identity provider and therefore must be encrypted with the public key of the recipient when being transferred between the two.



1. User contacts her preferred linking service, and chooses one of her IDPs (IDP 1)
2. User is redirected to her chosen IDP
3. User authenticates to IDP, and IDP generates a PID for this user with this linking service (PID 1:LS)
4. User is redirected back to linking service with an authentication assertion that carries the Session LOA and PID. Linking service stores this in the user's linking table entry.
5. User chooses another of her IDPs and the whole process is repeated again

Figure 5. Establishing links between identity providers

4.1.2. Level of Assurance

Different identity providers will authenticate users in different ways, and to different strengths e.g. usernames and passwords are weaker than public key certificates and private keys. This is termed the Level of Authentication, or Level of Assurance (LoA). It is the assurance that a relying party can have that the user is really who they thinks he or she is. The assurance a relying party has, depends not only upon the electronic authentication method that was used, but also on the initial registration process that preceded this, for example, registering electronically over the web is much weaker than turning up in person with your passport. The US National Institute

of Science and Technology (NIST) has a recommendation for LoA which classifies it in four levels, with level 4 being the strongest and level 1 being the weakest [4]. Some service providers may wish to grant a user different permissions based on the LoA of their current session. For example, if the user authenticates with an LoA of 1, they can only read the resource, but with an LoA of 3 they can modify its contents. One of the limitations of the NIST recommendation is that the LoA is a compound metric dependent upon both the strength of the registration process and the strength of the electronic authentication method. We believe it is more useful if they are separate metrics, as described below.

Prior to any computer based authentication taking place, a user needs to register with a service, and provide various credentials to prove their identity. For example, before a new student is registered to use the University of Kent's computing services, they must first present their passport and existing qualifications, to prove they are entitled to register as a university student. We call this the *Registration LoA*. Different systems will require different registration documents and have different registration procedures, and will therefore have different Registration LoAs. After successful registration, the university allocates the student a login ID (their identifier) and associates various attributes with this in its database, e.g. degree course, student's name, date of birth, email address, department, tutor and so on. The university may offer different authentication mechanisms for student login, such as un/pw with Kerberos, un/pw with SSL, one time passwords via a mobile phone etc. Each of these mechanisms is assigned an *Authentication LoA*, but with one proviso. No Authentication LoA can be higher than the Registration LoA, since it is the latter that originally authenticated who the user was. When a user logs in for a session, they are assigned a *Session LoA* that equates to the Authentication LoA of the authentication mechanism they chose to use.

Returning now to the linking service, we have made provisions to include the LoA in our protocol messages. When the linking service redirects the user to an identity provider during the link registration phase, the user authenticates to the identity provider with their preferred authentication mechanism, and this has an associated Authentication LoA. The identity provider may return this as the current Session LoA to the linking service, along with the permanent identifier. The linking service stores this Session LoA as the Registration LoA of the user for this permanent identifier/identity provider tuple, as shown in Table 1.

Local User ID	PID	IDP	Registration LoA
Fred	A=12345	airmiles.com	1
Fred	EduPersonID=u23@kent.ac.uk	kent.ac.uk	2
Fred	PID=4567890	XYX.co.uk	1
Fred	UID=qwertyuiop	cardbank.com	3
Etc……			

TABLE 1. The Identity Provider Linking Table

4.1.3. Link Release Policy

The user is allowed to create an identity provider link release policy. This tells the linking service which linked identity providers should be released to which service providers. In the simplest case, the user might indicate that all linked identity providers can be released to all service providers. This will normally be the default

policy for each linking service (and is indicated by an * in each of the columns of the identity provider link release policy table, see Table 2). In the most complex case, the user will require a different set of linked identity providers to be used with each service provider. An example of such a policy for the user known locally to the linking service as Fred is shown in table 2. This policy indicates that books.co.uk can receive attributes from three identity providers (airmiles.com, kent.ac.uk and cardbank.com); cardbank.com can receive attributes from all linked identity providers; and any other service provider should only receive attributes from the permanent identity EduPersonID=u23@kent.ac.uk from kent.ac.uk. The reason that both the permanent identifier and identity provider are held in this table is because the user may have two different identities with one identity provider, and might wish to link these together in a service provider session.

Local User ID	SP	PID	IDP
Fred	books.co.uk	A=12345	airmiles.com
Fred	cardbank.com	*	*
Fred	books.co.uk	EduPersonID=u23@kent.ac.uk	kent.ac.uk
Fred	books.co.uk	UID=qwertyuiop	cardbank.com
Fred	*	EduPersonID=u23@kent.ac.uk	kent.ac.uk
Etc…			

TABLE 2. Identity Provider Link Release Policy Table

4.1.4. Service Provision Phase

When the user wishes to use a web service, they first contact the web site. The service provider does not know who the user is, so must therefore redirect the user to their identity provider for authentication. This is the IdP discovery phase, and various different designs have catered for this. In the Shibboleth model, the service provider may use a Where Are You From (WAYF) service and redirect the user to this. In CardSpace, the service provider returns the user to their CardSpace application whereupon the user picks a card which represents their chosen identity provider. In OpenID, the user's OpenID contains the name of their Identity Provider which allows the SP to redirect the user there. Liberty Alliance (LA) has a Discovery Service [10] which can be used. Either way, the user must next present his authentication credentials to the identity provider, either directly in Shibboleth, OpenID and LA, or indirectly in CardSpace, via an authentication dialogue.

The authentication dialogue needs to be enhanced when attribute aggregation is supported, by asking the user if they wish to use attribute aggregation with this service provider. In the simplest case this can be a tick box alongside the username/password screen. Or it could be a list of Linking Services which the IdP trusts and which the user has already linked with. With direct identity provider authentication, the identity provider is able to show this enhanced screen since it knows if it has already generated one or more permanent identifiers for this user with one or more linking services. With CardSpace's indirect dialogue it is more difficult. The CardSpace application could show this enhanced screen if the service provider says that it supports attribute aggregation, but in this case CardSpace does not actually know if the user has already set up links or not. (This could be achieved by the identity provider issuing a new card to the CardSpace application after it has

established a permanent identifier for this user with a linking service, but this is not a particularly user friendly solution.)

If the user chooses to perform attribute aggregation, the identity provider includes one or more *referrals* in its response to the service provider. A referral in effect says "you may find additional attributes for this user at this provider". A referral in this instance points to a linking service, and includes the permanent identifier of the user encrypted to the public key of the linking service. When the service provider receives the authentication assertion containing the user's identity attributes, if these are sufficient for the requested service, then access will be granted and no linked attributes are needed. If they are not sufficient, and the service provider supports attribute aggregation, it will follow the referral(s) by forwarding it(them) to the linking service(s) along with the authentication assertion (to prove that the user has been authenticated). It sets a Boolean in the request to the linking service either telling the latter to perform the aggregation, or saying it will perform the aggregation and referrals should be returned to it.

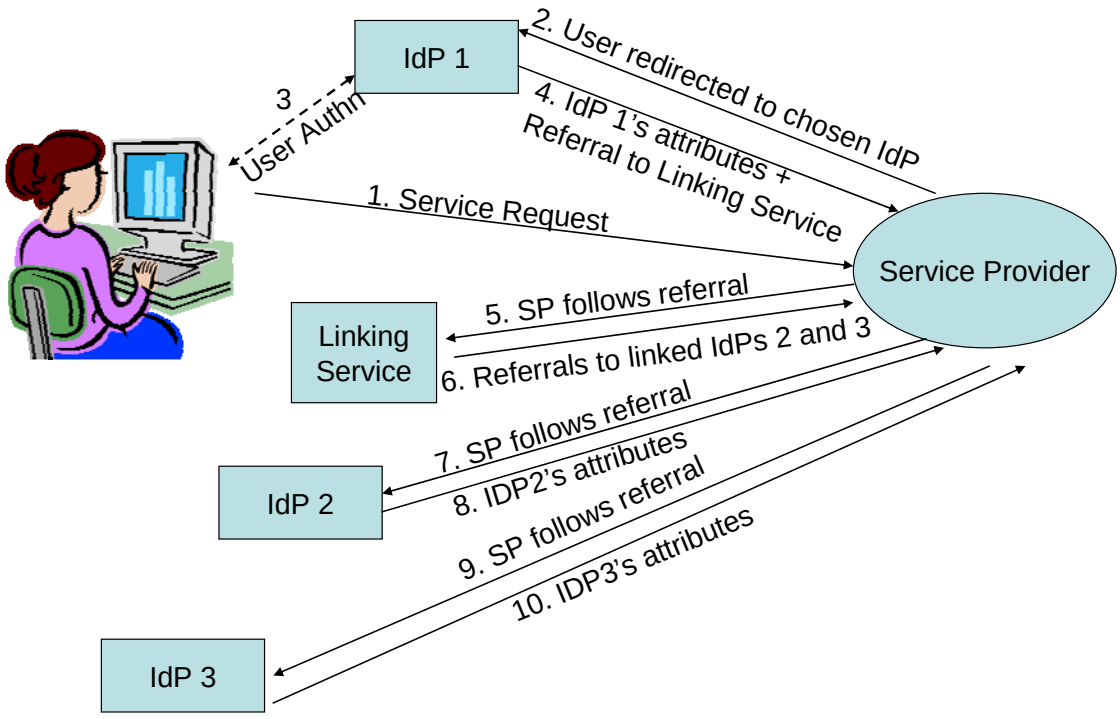


Figure 6. Attribute Aggregation by Service Provider

When the linking service receives the referral, it decrypts the permanent identifier and searches for this in its identity provider linking table (see Table 1). Once it has located the appropriate table entry, it retrieves the other table entries for the same user. Next it looks in its link release policy table (see Table 2) to see which of the linked identity providers can be sent to this service provider. If the service provider requested to perform the aggregation itself, then referrals to the allowed

identity providers are returned, with the permanent identifiers encrypted to their respective identity providers (see Figure 6). The service provider then acts on these referrals in the same way that it did with the original one. If the service provider requested the linking service to perform aggregation on its behalf, the linking service sends attribute query requests to the allowed identity providers (see Figure 7), forwarding the name of the service provider and the authentication assertion, so that the identity providers can encrypt their responses to the public key of the service provider and tie the attributes to the identifier found in the authentication assertion. Finally the identity providers digitally sign their responses. In this way service providers ultimately receive an authentication assertion and multiple attribute assertions, all digitally signed by their authoritative sources, and all containing the same random identifier that the original identity provider inserted into the authentication assertion. Although the service provider does not know any of the identifiers used to uniquely identify the user at each of the identity providers, nevertheless it can be assured that the user does possess all of the identity attributes in the various attribute assertions.

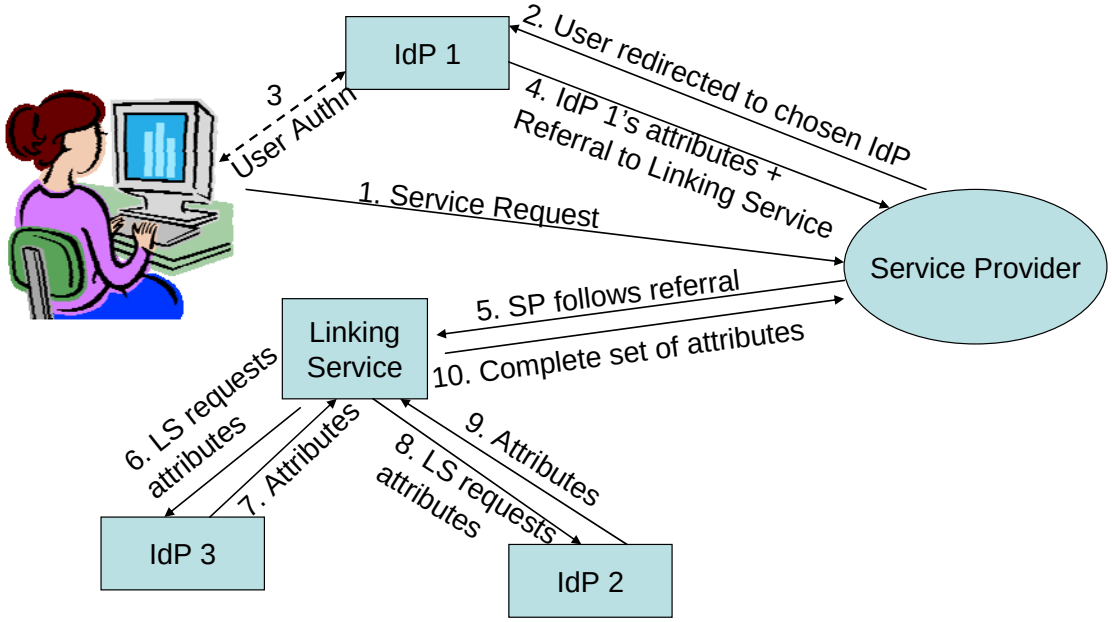


Figure 7. Attribute Aggregation by Linking Service

4.1.5. Using the LoA in Service Provision

The linking service may have stored Registration LoAs in its Identity Provider Linking Table during the user's link registration phase. Though not essential, they serve to

improve the performance of all subsequent user-service provider sessions. During a user's service session, the linking service will only utilise linked identity providers whose Registration LoAs are higher than or equal to the current Session LoA provided by the identity provider which authenticated the user's session. This prevents a user from creating links with low levels of assurance, and subsequently using them at higher Session LoAs, thereby pretending that the attributes have a high level of assurance. A user is allowed to create links at high Registration LoAs, and then subsequently use them on lower Session LoAs, since the service provider will know that the attributes can only be trusted up to the level of the current Session LoA.

If the linking service has stored the user's Registration LoA for a linked identity provider, and a subsequent user-service session is authenticated by a different identity provider at a lower Session LoA than this, the linking service is allowed to create a referral to the linked identity provider. The linked identity provider can then decide whether to return any attributes for the user at this low Session LoA or not. If however the subsequent user-session is authenticated by a different identity provider at a higher Session LoA than the Registration LoA, the linking service should not create a referral to the linked identity provider, since the linked identity provider should always refuse to return any attributes for the user in this high Session LoA. This is because its attributes have not been assured to such a high level and breaks the original proviso that no Authentication LoA can be higher than the original Registration LoA.

If the linking service has not stored the user's Registration LoA for a linked identity provider, then the linking service will need to create referrals to this linked identity provider for all subsequent user-service sessions, providing it is allowed by the link release policy, and the identity provider will need to decide whether to return the user's attributes or not.

4.2. Mapping the Conceptual Model to Standard Protocols

The conceptual model has been mapped to the Security Assertions Markup Language (SAML) v2 protocol [11] during the link registration phase, and to both Liberty Alliance and CardSpace web services protocols during the service provision phase. Our attribute aggregation model provides for the passing of the LoA between the various components, but this is currently not part of the SAMLv2 specification. However OASIS is currently working on a SAML profile of the NIST LoA recommendation, and when it is completed we will utilise this for passing the LoA between our components [12].

4.2.1. Link Registration Protocol

The link registration protocol uses standard SAML v2.0 authentication request/response messages [11] to request user authentication by a selected identity provider and return a persistent identifier to the linking service. Upon receipt of the permanent identifier in the SAML response, the linking service will either find an existing entry in the Identity Provider Linking Table for this permanent identifier/identity provider tuple, or a new entry will be created in the table. Either way, the user can then link additional identity provider accounts to this one.

In order to ensure that the identity provider always returns a persistent identifier to the linking service, the SAML authentication request is constrained in the following ways:

- the Format attribute of the <NameIDPolicy> element is set to“urn:oasis:names:tc:SAML:2.0:nameid-format:persistent”
- the allowCreate attribute of the <NameIDPolicy> element is set to true, which allows the identity provider to create a persistent identifier the first time around.

4.2.2. Service Provision Protocols

We have devised two possible protocol mappings for attribute aggregation using Liberty Alliance protocols, and one using CardSpace protocols. All three mappings encode referrals as Liberty Alliance ID-WSF Endpoint References (EPRs) according to the EPR generation rules defined in Section 4.2 of [13]. Each EPR points to a linking service where the service provider can find additional attributes for the user and the <sec:Token> of each EPR contains the encrypted permanent identifier of the user.

4.2.2.1. Service Provision using Liberty Alliance Protocols

Our original mapping used the Liberty Alliance ID-WSF Identity Mapping Service [13]. It required minor enhancements to the Liberty specifications. We took this mapping to the LA working group meeting in Stockholm in July 2008 for review and comment. The feedback we obtained was that the Liberty ID-WSF Discovery Service [10] might be better, since it would need fewer enhancements to the LA specifications, and open source code for the discovery service already existed, whereas none did for the identity mapping service. We then produced a mapping onto the discovery service and at the time of writing we are implementing this. After implementation we propose to take our results and minor enhancements back to the LA group for review and comment. The disadvantage of the discovery service mapping is that it requires two round trips between the SP or linking service and the IdP each time, the first trip being to discover the endpoint reference (EPR) of the attribute authority where the random identifier is now valid and the second to pick up the attribute assertions. In this case the linking service stores the discovery services of the various IdPs. The identity mapping service only requires one round trip each time since the identity mapping service is the one stored in the linking services tables.

The SAML authentication request message issued by the service provider, asks the identity provider to generate a random identifier for the user in the authentication response (by setting the Format attribute to “urn:oasis:names:tc:SAML:2.0:nameid-format:transient”) and to return both attributes and referrals (EPRs) in the response. The SAML response consists of a single SAML assertion which contains a SSO assertion containing three statements: an SSO authentication statement, an attribute statement containing the users attributes and an attribute statement containing the EPR(s) of the linking service(s). Once the service provider has received the SAML response it may attempt to access each of the EPR’s using the discovery service protocol mapping described below.

The discovery service is used for discovering web services. We can use this to discover the user’s various identity providers (and attribute authorities) that have been linked together at a linking service. After the service provider receives the initial referral(s) EPR(s) from the authenticating identity provider, it sends an ID-

WSF DiscoveryQuery [10] to each linking service. The DiscoveryQuery message contains the <sec:Token> copied from the referral EPR and the initial authentication assertion in the message's SOAP header [14]. The linking service decrypts the permanent identifier, retrieves the linked identity providers and if the service provider is performing aggregation, returns an ID-WSF QueryResponse. This contains referral EPRs to the discovery services of the user's linked identity providers. The service provider then sends a DiscoveryQuery message to each identity provider's discovery service, requesting the EPR of the attribute authority of the user. Alternatively, if the linking service is performing the aggregation, it sends the same message. The identity provider's discovery service locates the user's account by decrypting the permanent identifier, and then maps the random identifier from the authentication assertion into the user's account. The identity provider returns a QueryResponse containing the EPR of the attribute authority where the random identifier is now valid. The service provider (or linking service) sends a standard <samlp:AttributeQuery> to the attribute authority, using the random identifier, whereupon the attribute authority returns a standard <samlp:Response>, encrypted so that only the service provider can retrieve the attributes.

5. Multiple Policy Authorization Evaluation Infrastructure

The multiple policy authorization evaluation infrastructure introduces a new conceptual component called a Master PDP. The Master PDP is responsible for calling the multiple PDPs of the TAS3 infrastructure, obtaining their authorization decisions, and then resolving any conflicts between these decisions, before returning the overall authorization decision and any resulting obligations to the AIPEP.

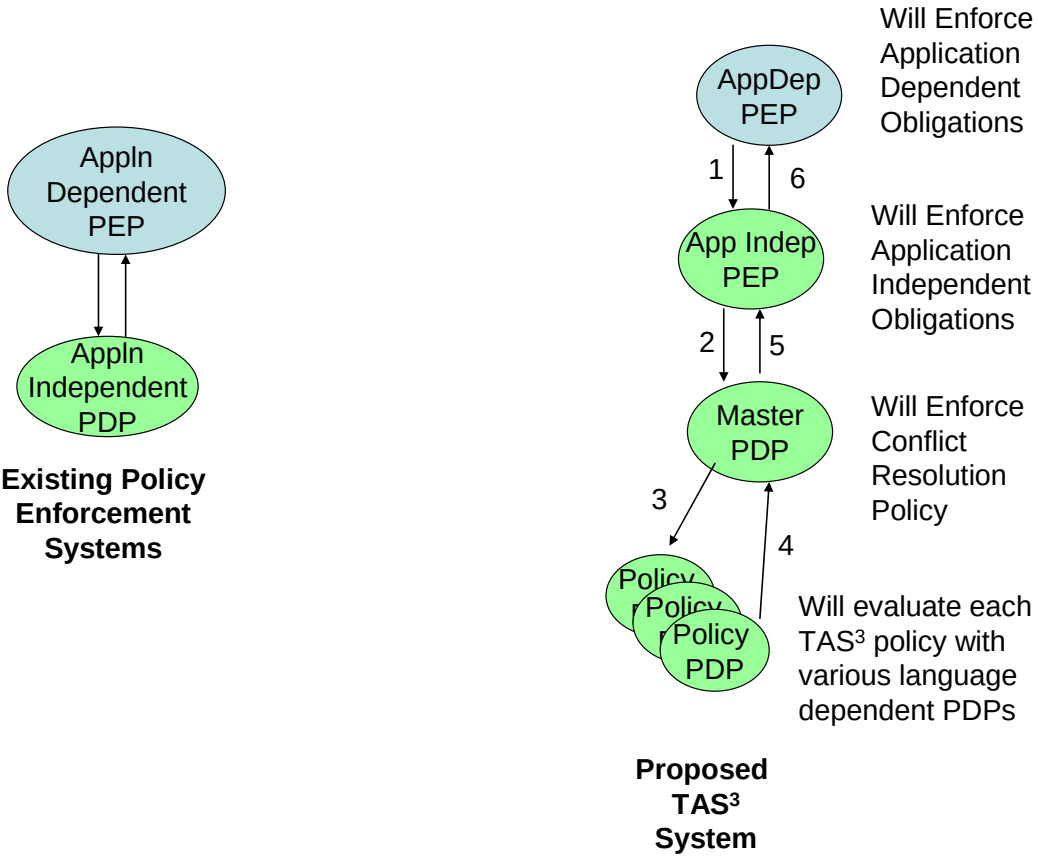


Figure 8. The Proposed TAS3 PEP-PDP Infrastructure

The Master PDP is configured with a Conflict Resolution Policy which enables it to determine which authorization decision from which policy/PDP combination should take precedence. Each of the policy PDPs will support the same interface, which in the first phase of TAS^3 will be the XACML request-response context [15]. This allows the Master PDP to call any number of PDPs, each configured with their own policy, which may be written in a different policy language to those of the other PDPs. For example, WP5 is building a behavioral trust engine which will return an authorization decision about whether the requester is trusted or not to perform the requested action. The policy language for specifying the behavioral trust rules is still to be

finalized, but the proposed design isolated this policy language from the rest of the authorization infrastructure, and the Master PDP will not be affected by any changes in this policy language as it evolves.

Consider an agency that wishes to retrieve some PII of a subject from a service provider in some country. The subject will have his own privacy protection policy written in some language e.g. P3P or XACML, which will be stored with the PII. The SP will have its own policy for who can access what information it is storing. The legislature for the country the data is held in may have its own privacy protection policy that the SP must enforce. Finally the agency will have a behavioral track record which indicates how trustworthy it is at enforcing privacy protection policies. The proposed TAS^3 design caters for this scenario by allowing all these policies and PDPs to be consulted before the agency is granted permission to retrieve the PII. A key component of this design is the conflict resolution policy that the Master PDP will enforce. Our current thinking is that the authoritative source for each element of PII (which might be as small as an individual attribute) should be the Source of Authority (SOA) for the conflict resolution policy that is enforced by the Master PDP for that element of PII. For example, for the favourite drink attribute, the user would be the authoritative source and therefore the author of the conflict resolution policy for this element, whereas for a subject's criminal record the legal system would be the authoritative source of the information and of the conflict resolution policy for this element (since clearly the user may wish to deny everyone access to his criminal record data.) The precise contents and syntax of the conflict resolution policy is still to be determined, and is for further study.

6. Dynamic Delegation of Credentials Infrastructure

Delegation of authority is an essential procedure in every modern business. A delegate is defined as “A person authorized to act as representative for another; a deputy or an agent” (www.dictionary.com). Without delegation of authority (DOA), managers would soon become overloaded. DOA allows tasks to be disseminated between employees in a controlled manner. A delegate may be appointed for months, day or minutes, for one task, a series of tasks, or all tasks associated with a role. DOA needs to be fast and efficient with a minimum of disruption to others. Delegators should not need permission from their superiors for each act of delegation they undertake, otherwise their superiors would soon become overburdened with delegation requests from subordinates. Instead, a delegation policy should be in place so that delegators know when they are empowered to delegate (i.e. what and to whom) and when they are not.

The recipient (or service provider) who is asked to perform a service for a delegate should be able to independently verify that the delegate has been properly authorized to act as a representative for the delegator, before granting the request. If the delegate has not been properly authorised, the delegate’s request should be declined. The recipient will therefore enforce the delegation policy of its organization and deny service requests from unauthorized delegates.

In a computing environment there is also a need for DOA. One computer process may need to delegate to another computer process. One person may need to delegate his privileges to another person in order to allow the later to undertake the computer based tasks of the former. Similarly in a service oriented world, computer services also need the ability to delegate tasks to other services, so that the latter can perform subtasks on the former’s behalf. Service providers need to be able to verify that each service requestor is properly authorized. If the service requestor has been dynamically delegated authority by another authorized entity, service providers need to be able to verify that this was done in accordance with their delegation policy.

6.1. Requirements for Web Services Delegation Of Authority

As stated above, the first requirement is for a general purpose delegation of authority service that can delegate from any type of entity to any other type of entity. (Requirement 1)

Secondly, we need to be able to independently name the delegator and the delegate. It might be acceptable in person to job delegation that the job takes a name subordinate to that of the person, but in person to person delegation and web service to web service delegation we should not have to make the delegate assume a principal name which is subordinate to that of the delegator. For the reason of prudent accountability, if nothing more, every principal should authenticate with its own identity, and not with that of another. So delegation should be from one named entity to another, where their names do not need to bear any relationship to each other. (Requirement 2)

In order to build a scalable authorization infrastructure, we need to move towards attribute or role based access controls, where a principal is assigned one or more attributes, and the holder of a given set of attributes is given certain access rights to certain resources. In this way we can give access rights to a whole group of principals e.g. to anyone with an IEEE membership attribute, or to any member of project X, or any web service of a specific type, without needing to list all the members individually as there might be many thousands of them. (Requirement 3)

The delegation scheme will benefit from a hierarchical model for roles and attributes so that delegators can delegate a subset of their roles/attributes. With hierarchical roles and attributes, a principal with a superior role (or attribute) inherits all the permissions of the subordinate roles (or attributes), and may delegate a subordinate role rather than the most superior role he holds. For example, a project manager may be superior to a team leader who is superior to a team member who is superior to an employee. Principals should to be able to delegate any of their roles and attributes to other principals, so that the delegate may perform on their behalf only those tasks that are enabled by the delegated attributes. For example, a project manager should be able to delegate the subordinate role of team member to an employee. (Requirement 4)

All organizations need to be able to control the amount of delegation that is possible, in order to stop “wrong” delegations from being performed. For example, a project manager should not be able to delegate his age or name attributes to anyone else, nor be able to delegate the team member role to one of his children. So we need to have a Delegation Policy, and an effective enforcement mechanism that will control both the delegation process itself (is this delegator allowed to delegate these attributes to this delegate?) and the verification process by the consuming web service (is this delegate properly authorized to access this service?). (Requirement 5)

We may want very fine grained delegation, in order to delegate a specific task rather than attributes or roles, because the latter usually confer permissions to perform a set of tasks. (Requirement 6)

Users must not be constrained to having a PKI key pair before they can delegate to another entity. Users should be able to authenticate and prove their identity without having to possess a public key certificate. (Requirement 7)

A delegator should be able to prematurely revoke an act of delegation, without the delegation lasting for its originally intended period of time. When delegation takes place, its effect should be instantaneous. There are many reasons why premature revocation may be needed e.g. the delegator returns early from vacation or sick leave and wishes to continue in his role himself, or the delegate proves to be untrustworthy or incompetent in the delegated role, or the delegate completes the delegated tasks earlier than anticipated and their privileges should now be removed etc. (Requirement 8)

Finally, we wish to make the whole DOA system web services compliant, so that it will integrate nicely with the service oriented architectures (SOA) web services world that is the subject of the TAS^3 project. (Requirement 9)

6.2. Design of a Delegation of Authority Web Service

We can utilise the existing PEP/PDP and ABAC models when creating a delegation of authority (DOA) web service – see Figure 9. The DOA web service will receive a delegation request from a delegator to delegate an attribute or attributes to a delegate, in the form of an attribute certificate (AC). An AC is a digitally signed

attribute assertion that states that the holder (the delegate) has been assigned this set of attributes by the issuer (the delegator). The delegator can be any web service, or a human being acting via a web services user interface. The delegate can be another web service or another human being. In this way we achieve the desired objective of person to person, service to service, person to service and service to person delegation of authority (Requirement 1). The target resource of the DOA web service, which acts as a conventional PEP, is the software that is able to issue the AC for the delegate, on behalf of the delegator. This *issue AC* software should create the attribute certificate in any standard format as required (e.g. either X.509 AC or a signed SAML attribute assertion). This *issue AC* software should have its own digital signing key pair for this task, so that future credential recipients can verify that the issued credential is authentic. Since most users do not have their own PKI key pairs they cannot issue their own ACs. This is why we require the DOA web service to sign the credential on the delegator's behalf. This solves Requirement 7.

The delegator's request will be intercepted by the DOA Web Service PEP, and passed to the PDP to ask if this user is allowed to delegate this/these particular attribute(s) to this delegate, according to the organisation's delegation policy (Requirement 5). The PDP (actually the PIP) either retrieves the delegator's current set of authorisation credentials or roles/attributes from the local repository, or they are sent by the user along with the delegation request. The PDP then consults the delegation policy to see if the requested delegation is allowed or not. If the policy allows the delegator and delegate to be independently named, then this solves Requirement 2. As a result of evaluating the policy, the PDP replies granted or denied to the PEP. If granted, the PEP will ask the *Issue AC* software to issue a delegated authorisation credential to the delegate on behalf of the delegator, and will then either publish this in the local credential repository or return it to the requestor, or both. The delegate will now be able to use the issued credential to gain access to the service that has been delegated to him, and may also be able to further delegate the embedded attribute to other delegates, if allowed by the delegation policy. If the local repository stores delegated attributes instead of credentials, the *Issue AC* software will still create the delegated attribute(s) for the delegate, but not sign them, and the delegated attribute(s) will be stored in the local repository. Subsequently the delegate will be able to ask the DOA web service to dynamically issue a new short lived credential for him, based on the attributes that are stored for him in the local repository.

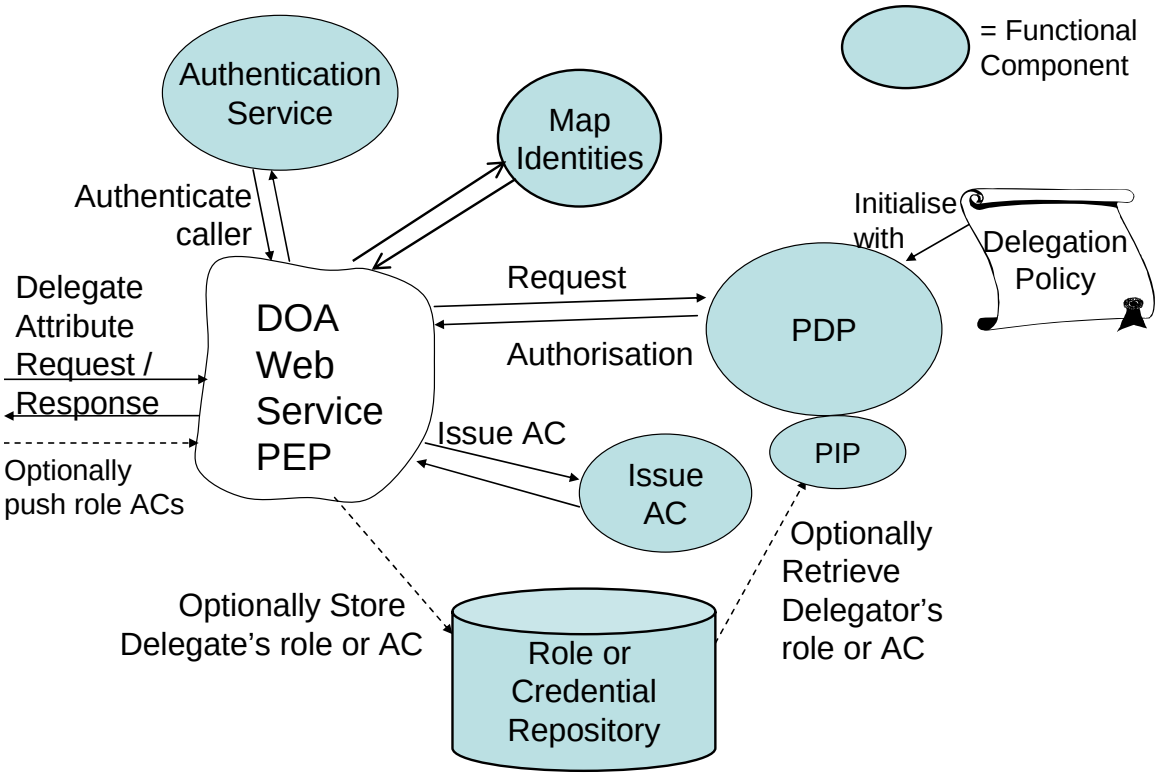


Figure 9. The Delegation of Authority Web Service Architecture

When a delegator makes a Delegate Attribute request to the DOA web service, the delegator is first authenticated to determine who he is. Delegator authentication can be by any suitable means, and can be via an internal authentication service or external FIM service as previously described. This model does not dictate any particular authentication scheme (Requirement 7). It is up to an implementation to determine the most appropriate authentication mechanism to use. That being said, digital signatures would be the most appropriate and secure mechanism for web service to web service authentication, but for authenticating a human user that is accessing the DOA web service via a web services user interface, a username and password stored in the local LDAP directory might be appropriate.

The next step is to optionally map the requestor's authenticated name into the authorisation name that is held in the authorisation credentials. This step is only needed if the two names are different, for example, when proxying is used[4] or when the authentication mechanism uses a different name form to that stored in the issued credentials[5]. Ideally this step should not be needed in the latter case, since

[4] For example a user may authenticate to a proxy and the proxy authenticates to the DOA web service, passing the user's name as a parameter. The mapping function would need to retrieve the user's name from the message passed by the proxy.

[5] For example, the authentication service uses usernames and passwords which are stored in the LDAP entries of the users, whilst the LDAP distinguished names are used in the authorisation credentials.

the authenticated name should be held in the authorisation credential. If the mapping is needed, how this is performed is not part of the model, but care will be needed since a security vulnerability will be introduced if the mapping is not made in a secure manner.

Once the PEP has the delegator's authorisation name, it asks the PDP if this user is allowed to delegate this/these particular attribute(s) to the delegate. If granted is returned, the PEP then asks the target resource (*Issue AC*) to issue the new authorisation credential to the delegate, on behalf of the delegator. It then publishes the new credential in the repository and/or returns it to the requestor. If the delegate wishes to further delegate this credential to someone else, then the delegate will now take on the role of delegator and access the DOA web service to request delegation of this/these attribute(s) to someone or something else. In this way, delegation can continue automatically from one user to another, providing of course that each delegation is in accordance with the organisation's delegation policy.

The model supports three different modes of operation, depending upon whether the repository stores credentials (ACs) or plain attributes/roles or whether the delegator pushes authorisation credentials to the service and receives delegated credentials in return. In all cases, delegation only takes places once, but credential issuing may take place zero, one or more times. When the repository stores credentials, they are only issued once by the DOA web service, they will typically have a relatively long lifetime (the period of the delegation), and they can be retrieved at will from the repository by the delegate or by web services that wish to validate the authority of the delegate to access their services. When the repository stores attributes/roles, the DOA web service can be called repeatedly by the delegate to issue typically short lived credentials to it based on the attributes/roles that have been delegated and stored in the repository. This service could also be used by other web services which the delegate is attempting to access, in order to retrieve the delegate's credentials. When the DOA web service is only issuing already delegated attributes, the delegator's name is not required, only the name of the delegate. But the requestor must be authenticated and authorised to ensure they are entitled to retrieve the delegate's short lived credentials. In both of these modes of operation the repository will need to record the validity period of the delegation and any policy conditions that are attached to it. If credentials are stored, this information is embedded in the issued credentials, if attributes are stored, separate fields will be needed in the repository to record it. When the repository stores attributes, it has to be strongly secured to prevent tampering with its contents and an attacker inserting false attributes. When the repository stores credentials, since the latter are digitally signed, it is not possible for an attacker to insert false credentials into the repository without first gaining access to the private signing key of the *Issue AC* service. Even if the repository is only weakly protected, the worst an attacker could do would be to remove a user's credentials – a denial of service attack. When the delegator presents his credentials during the request for delegation, and asks for the delegate's credentials to be returned to it, no repository is needed by the DOA web service. The credentials are stored external to the service, but in this case the DOA web service needs a Credential Validation Service (CVS) to validate the presented credentials. The CVS is described next.

6.3. Design of a Credential Validation Service

We propose a new conceptual component called a Credential Validation Service (CVS), whose purpose is to validate a set of credentials for a subject, issued by multiple dynamic attribute authorities from different domains, according to the local policy, and return a set of valid attributes. There are several reasons for making the CVS a separate component to the PDP. Firstly, its purpose is to perform a distinct function from the PDP. The purpose of the PDP is to answer the question "given this access control policy, and this subject (with this set of valid attributes), does it have the right to perform this action (with this set of attributes) on this target (with this set of attributes)" to which the answer is essentially a Boolean, Yes or No[6]. The purpose of the CVS on the other hand is to perform the following "given this credential validation policy, and this set of (possibly delegated) credentials, please return the set of valid attributes for this entity" to which the answer will be a subset of the attributes in the presented credentials, possibly mapped into locally known and trusted attributes. When architecting a solution there are several things we need to do. Firstly we need a trust model that will tell the CVS which credential issuers and policy issuers to trust. Secondly we need to define a credential validation policy that will control the trust evaluation of the credentials, including mapping the validated attributes into locally known attributes. Finally we need to define the functional components that comprise the CVS.

6.3.1. The Trust Model

The CVS needs to be provided with a trusted master credential validation policy[7]. We assume that this credential validation policy will be provided by the Policy Administration Point (PAP), which is the conceptual entity from the XACML specification that is responsible for creating policies [15]. If there is a trusted communications channel between the PAP and the CVS, then the policy can be provided to the CVS through this channel. If the channel is not trusted, or the policy is stored in an intermediate repository, then the policy should be digitally signed by a trusted policy author, and the CVS configured with the public key (or distinguished name if X.509 certificates are being used) of the policy author. In addition, if the PAP or repository, has several different credential validation policies available to it, that are designed to be used at different times and under different conditions, then the CVS needs to be told which policy to use. In this way the CVS can be assured of being configured with the correct credential validation policy. All other information about which sub policies, credential issuers and their respective policies to trust can be written into this master credential validation policy by the policy author.

In a distributed environment we will have many issuing authorities, each with their own issuing policies provided by their own PAPs. If the policy author decides that his CVS will abide by these issuing policies there needs to be a way of securely obtaining them. Possible ways are that the CVS could be given read access to the remote PAPs, or the remote issuing authorities could be given write access to the local PAP, or more realistically, the issuing policies can be bound to their issued

[6] XACML also supports other answers: indeterminate (meaning an error) and not applicable (meaning no applicable policy), but these are conceptually other forms of No.

[7] Note that whilst we refer to the policy in the singular, we acknowledge that it will contain multiple policy statements, and therefore may be regarded as a set of policies.

credentials and obtained dynamically during credential validation. Whichever way is used, the issuing policies should be digitally signed by their respective issuers so that the CVS can evaluate their authenticity. If the issuing policies are bound to the credentials, then a single signature over all the information will suffice.

The policy author may decide to completely ignore all the issuer's policies, or to use them in combination with his own credential validation policy, or to use them in place of his own policy. Thus this information (or policy combining rule) needs to be conveyed as part of the CVS's policy.

6.3.2. The Credential Validation Policy

The CVS's policy needs to comprise the following components:

- a list of trusted credential issuers. These are the issuers in the local and remote domains who are trusted to issue credentials that are valid in the local domain. They are the roots of trust. This list is needed so that the signatures on credentials and policies can be validated. The list could contain the raw public keys of the issuers or it could refer to them by their X.500 distinguished names or their X.509 public key certificates.
- the hierarchical relationships of the various sets of attributes. Some attributes, such as roles, form a natural hierarchy. Other attributes, such as file permissions might also form one e.g. *all* permissions is superior to *read, write* and *delete*; and *write* is superior to *append* and *delete*. When an attribute holder delegates a subordinate attribute to another entity, the credential validation service needs to understand the hierarchical relationship and whether the delegation is valid or not. For example, if a holder with a manager role delegates the administrator role to someone, is this a valid delegation or not? The relationship of manager to administrator in the attribute hierarchy will provide the answer to this question.
- a description (schema) of the valid delegation graph. The process of delegation forms a directed acyclic graph (DAG), with the initial PMI roots of trust as the sources of the graph. Intermediate nodes in the graph represent delegates who subsequently act as delegators and further delegate their attributes (or permissions) to others. Sink nodes represent delegates who have not further delegated their attributes (or permissions) to others. Edges in the graph represent the attributes or permissions that have been delegated from the delegator to the delegate. Successor edges must always represent the same or less attributes and permissions than the union of their predecessor edges, otherwise a delegator will have delegated more privileges than he himself possessed. The graph is acyclic because a delegator should not be able to delegate to herself or to a predecessor. Rationally, there is a reason for this – a delegate should never *need* to delegate to an entity that previously delegated directly or indirectly to it. But there is also a security reason for this. There is a potential security loophole if a delegator, who is allowed to delegate a privilege but not to assert it, does subsequently delegate it to herself, then she would be able to assert the delegated privilege. This CVS policy component describes how the CVS can determine if a chain of delegated credentials and/or policies falls within a trusted graph or not. This is obviously a complex policy component. One way of simplifying it, is to restrict the directed graph into being a delegation tree, in which there is only one source or PMI root node which holds all the attributes that can be delegated, and each act of delegation creates a separate delegate subordinate node. If a delegate receives attributes from two or more delegators

in separate acts of delegation, then these are represented as separate edges and nodes in the tree, without merging the delegate nodes together. Delegation trees significantly simplify the process of credential validation and credential revocation because each credential only has a single parent. Even then, there is no widely accepted standard way of describing delegation trees. One approach can be found in X.509 [17] and a different approach in [18]. The essential elements however should specify who is allowed to be in the tree (both as an issuer and/or a subject), what attributes they can validly have (assert) and delegate, and what constraints apply.

- any validity constraints on the various credentials (e.g. time constraints or target constraints). The CVS's policy may place its own constraints on credential validity regardless of those of the issuer.
- finally, we need a disjunctive/conjunctive directive (or policy combining rule) to say how to intersect the issuer's policy with the CVS's own policy. The options are: only the issuer's issuing and delegation policy should take effect, or only the CVS's policy should take effect, or both should take effect and valid credentials must conform to both policies.

Note that when dynamic delegation of authority is not being supported, the above policy can still be used in a simplified form where a delegation tree or graph reduces to a one level hierarchy, in which the root node(s) are the set of trusted issuers and the first level are the set of delegates who can be issued with credentials. In this case the CVS's policy now controls which trusted issuers are allowed to assign which attributes to which subjects, along with the various constraints and disjunctive/conjunctive directive.

An important requirement for multi-domain dynamic delegation is the ability to accept only part of an asserted credential. This means that the policy should be expressive enough to specify what is the maximum acceptable set of attributes that can be issued by one Issuer to a Subject, and the evaluation mechanism must be able to compute the intersection of this with those that the Subject's credential asserts. Our model is based on full independence of the issuing domain from the validating domains. In general it is impossible for a validating domain to fully accept an arbitrary set of credentials from an issuing domain, since the issuing and validating policies will not match. It is not always possible for the issuing domain to tell in advance in what context a subject's credentials will be used (unless new credentials are issued every time a subject requests access to a resource) so it is not possible to tell in advance what validation policy will be applied to them.

6.3.3. The CVS functional components

Figure 10 illustrates the architecture of the CVS function and the general flow of information and sequence of events. First of all the service is initialised by giving it the credential validation policy (step 0). Now the CVS can be queried for the valid attributes of an entity (step 1). Between the request for attributes and returning them (steps 1 and 6) the following events may occur a number of times, as necessary i.e. the CVS is capable of recursively calling itself as it determines the path in a delegation tree from a given node to a PMI root of trust. The Policy Enforcer requests credentials from a Credential Provider (step 2). When operating in credential pull mode, the credentials are dynamically pulled from one or more remote credential providers (these could be AA servers, LDAP repositories etc.). The actual attribute request protocol (e.g. SAML or LDAP) is handled by a Credential

Retriever module. When operating in credential push mode, the CVS client stores the already obtained credentials in a local credential provider repository and pushes the repository to the CVS, so that the CVS can operate in logically the same way for both push and pull modes. After credential retrieval, the Credential Retriever module passes the credentials to a decoding module (step 3). From here they undergo the first stage of validation – credential authentication (step 4). Because only the Credential Decoder is aware of the actual format of the credentials, it has to be responsible for authenticating the credentials using an appropriate Credential Authenticator module. Consequently, both the Credential Decoder and Credential Authenticator modules are encoding specific modules. For example, if the credentials are digitally signed X.509 attribute certificates, the Credential Authenticator uses the configured X.509 PKI to validate the signatures. If the credentials are XML signed SAML attribute assertions, then the Credential Authenticator uses the public key in the SAML assertion to validate the signature. The Credential Decoder subsequently discards all credentials that are deemed by the Authenticator module to be unauthentic – these are ones whose digital signatures are invalid, either cryptography or because the signer's certificate cannot be traced to a PKI root of trust, or because the signer's certificate has been revoked. Authentic credentials on the other hand are decoded and transformed into an implementation specific local format that the Policy Enforcer is able to handle (step 5).

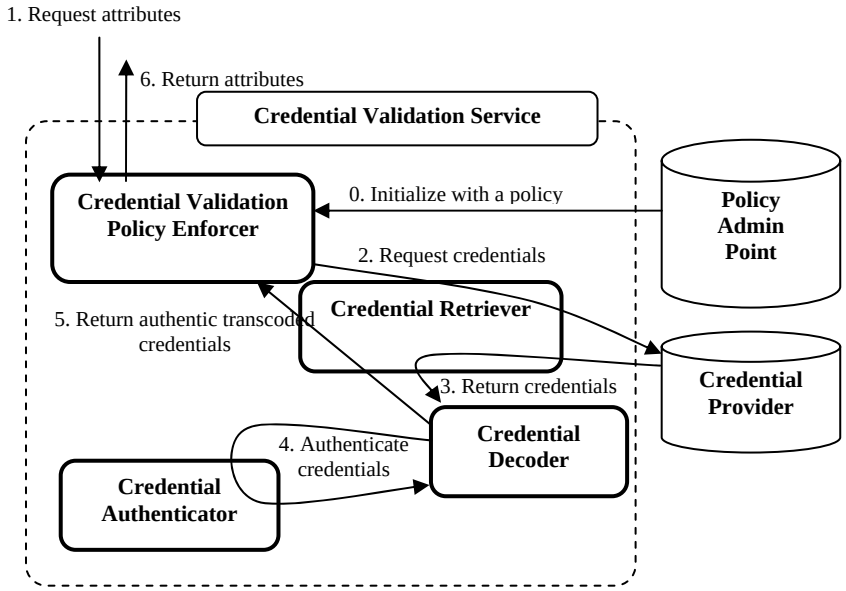


Figure 10. Data Flow Diagram for Credential Validation Service Architecture

The task of the Policy Enforcer is to decide if each authentic credential is valid (i.e. trusted) or not. It does this by referring to its Credential Validation policy to see if the credential has been issued by a PMI root of trust or not. If it has, it is valid. If it has not, the Policy Enforcer has to work its way up the delegation tree (or graph) from the current credential to its issuer, and from there to its issuer, recursively, until a PMI root of trust is located, or no further issuers can be found (in which case the credential is not trusted and is discarded). Consequently steps 2-5 are

recursively repeated until closure is reached. Even when the delegation graph has been simplified to a delegation tree, in the general case there will be multiple trees each with their own PMI root of trust, who each may have their own Issuing Policy, which may have been further restricted by their delegates, which may then need to be adhered to or not by the Policy Enforcer according to the CVS's policy. There are also issues of height first or breadth first upwards tree walking, or top-down vs. bottom-up tree walking. These are primarily implementation rather than conceptual issues, as they effect performance and quality of service, and so we will address these later when we describe our implementation of a CVS.

The proposed architecture makes sure that the CVS can:

- Retrieve credentials from a variety of physical resources
- Decode the credentials from a variety of encoding formats
- Authenticate and perform integrity checks specific to the credential encoding format

All this is necessary because realistically there is no way that all of these will fully match between truly independent issuing domains and the relying party.

7. Dynamic Management of Policies Infrastructure

In a web services environment, there are issuing domains that issue credentials to users and target domains that consume credentials. The authorization policy of the target domain decides whether an issued credential is to be trusted or not i.e. is valid or not, and whether it provides sufficient permissions or not to the accessed resource. In an attribute (or role) based authorisation policy, the permission-attribute assignment (PAA) rules form the access control policy. The user-role assignment (URA) rules form the credential validation policy. Thus, an authorisation policy includes an access control policy and a credential validation policy.

In this section, we propose a dynamic management of policies model which provides the following features for authorisation administration in a web services world:

- Administrative roles are defined which grant permission to dynamically update limited parts of the authorisation policy in the target domain, more specifically, to assign organizational level attributes to a subset of the privileges which grant access to a service's workflow resources (see Figure 11).
- Administrators are dynamically created by assigning these administrative roles to them. These roles can be dynamically delegated, and also dynamically revoked, thereby dynamically adding and removing administrators from the system.
- An administrator can dynamically assign a subset of the workflow permissions granted by the administrative role, to any organizational level user attributes (i.e. perform PAA). In addition, the administrator can provide the policy information for validating the user credentials that contain these attributes (i.e. URA validation).
- Collaborations between organisations are independent of each other, since an organisation's workflow privileges are independent of those of other organisations.
- Application-level (workflow) security infrastructures are separated from organisational level security infrastructures since workflow permissions are dynamically assigned to organizational level attributes.

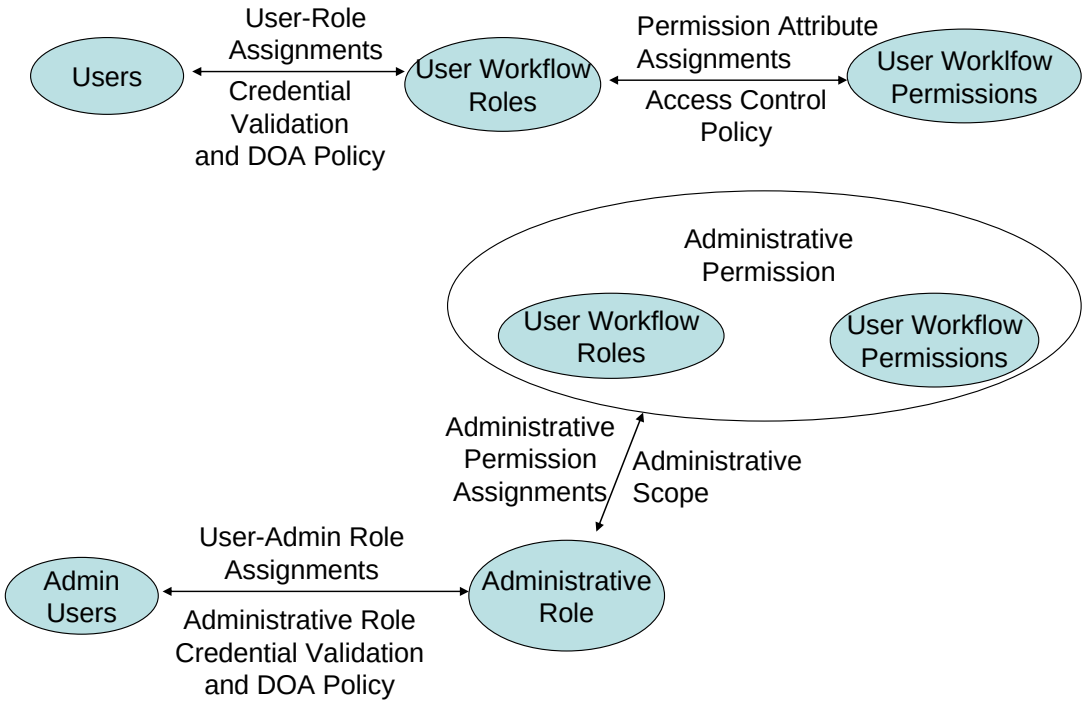


Figure 11. User Roles and Administrative Roles

By allowing authorization policies to be dynamically updated as above, our model allows the authorisation system of a target domain to dynamically *recognise* trusted administrators, to dynamically *recognise* the new attributes they are trusted to issue, and to dynamically *recognise* new users of the VO. The initial definition of the administrative roles means that the authorization system knows the limit of their administrative authority in assigning permissions to users. We call this model Recognition of Authority.

There are two approaches for assigning permissions in a local organisation to users in partner organisations. The first is to directly assign workflow permissions to remote user attributes and the second is to map remote user attributes into local workflow roles/attributes by attribute-role mapping. Both approaches can facilitate collaborations between organisations. In the attribute-role mapping approach, the permissions given to a remote attribute are the workflow permissions of the local role, which is fixed. Thus, this approach limits the granularity of delegation to that of the pre-defined local workflow roles (and their subordinate roles), whilst direct permission assignment allows each workflow permission to be delegated or assigned separately. On the other hand, by mapping remote user attributes to local workflow roles, the changes of participants in a workflow are confined to the modification of mappings from an organisation's attributes to the local workflow roles (it does not affect the workflow's specification) and changes to the specification of local workflow roles do not require modifications to the remote user attribute specifications. Thus,

this approach supports the separation of workflows from organisational changes. Since both approaches have their merits, our model is designed to support both approaches. When an administrative role is defined, its administrative permissions are defined as either an ability to assign a restricted set of workflow permissions to any user attributes, or an ability to map any user attributes into a restricted set of existing local workflow roles.

We identify two types of permission: a *normal permission* (or *workflow permission*) and an *administrative permission*. A workflow permission is a consent (for a user) to perform a particular workflow action on a particular resource under certain conditions. An administrative permission is a consent (for an administrator) to perform either PAA, or to perform role mappings to workflow roles.

When a set of workflow permissions is given to an attribute or role, we say that the role or attribute is a *workflow role*. When a set of administrative permissions is given to a role we say the role is an *administrative role*. Someone who holds an administrative role is called an administrator. The set of workflow permissions and workflow roles that an administrator can assign or map to new user attributes is called his *administrative scope*.

The recognition of authority management model for facilitating dynamic collaboration between organisations comprises the following steps:

1. The policy writer (SoA) of the target domain defines a set of administrative roles for the target domain, an administrative role credential validation policy, and the workflow permissions that are attached to these administrative roles (i.e. the administrative scope).
2. The SoA dynamically delegates these administrative roles to trusted people in remote domains with whom there is to be a collaboration, by issuing administrative role credentials to them.
3. To establish a collaboration, one of these administrators must update the SoA's authorisation policy by writing a collaboration policy. The collaboration policy includes an access control policy and/or a role mapping policy, and a user credential validation policy. The latter specifies validation rules for user credentials containing newly defined (organizational level) user attributes, whilst the former specifies either permission attribute assignments or role mappings for the newly defined user attributes. In this way, users who hold credentials containing these new attributes will gain access to the appropriate target workflow resources.
4. In order to ensure that no administrator can overstep his delegated authority, the authorisation system has to validate that the collaboration policy lies within the the administrative scope specified in 1. above. If it does, it is accepted, and its policy rules become dynamically incorporated into the SoA's policy. If it does not, it is rejected, and its policy rules will be ignored.
5. When a user from a collaborating domain wants to access a protected resource in the target domain, assuming the collaboration policy has been accepted, the authorisation system retrieves and validates the user's credentials/attributes against the now enlarged credential validation policy. Only valid attributes will then be used by the access control system to make access control decisions for the user's request against the now enlarged access control policy.

6. An administrator may dynamically delegate his administrative role to another person, providing the delegate falls within the scope of the administrative role credential validation policy set by the resource SoA (see Figure 11).

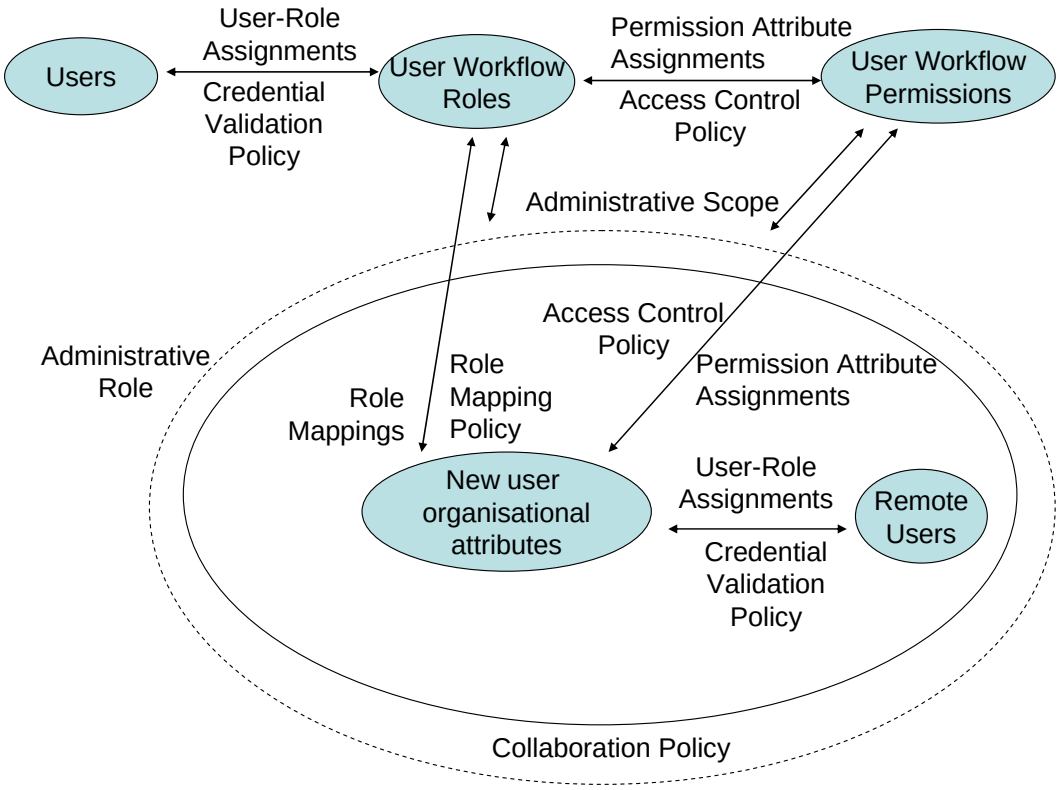
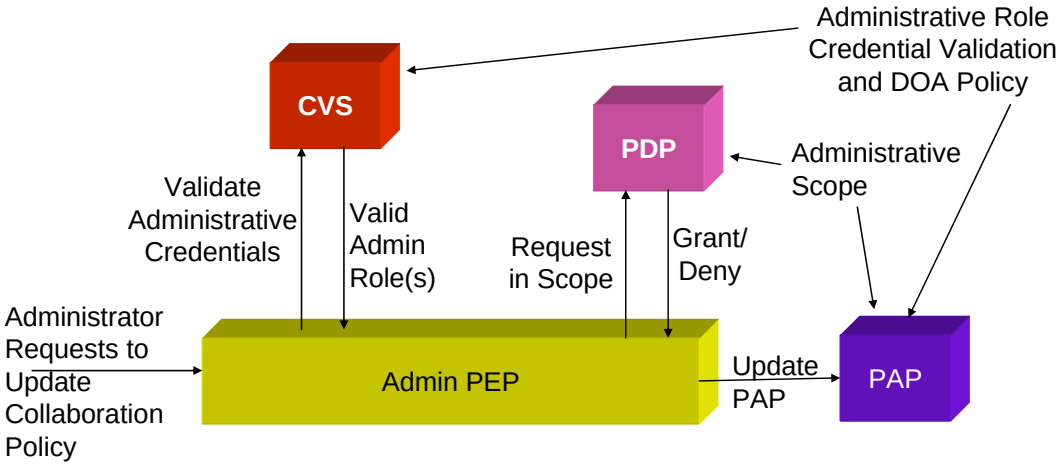


Figure 12. Collaboration Policies

7.1. Updating the Collaboration Authorisation Policy



CVS=Credential Validation Service
PDP= Policy Decision Point
PEP= Policy Enforcement Point
PAP= Policy Administration Point

Figure 13. Updating the Collaboration Authorisation Policy

The high level conceptual model for the dynamic updating of collaboration authorization policies is presented in Figure 13. The PAP holds the authorisation policy of the target domain. The system is initialized by the SoA writing an administrative role validation policy for the CVS and an administrative scope validation policy for the PDP. These policies are stored in the PAP. The CVS and PDP need to read these policies in order to validate the administrators' requests.

The SoA delegates the administrative roles to remote administrators and they can delegate further if necessary. The delegation web service described in section 6 can be used for this, or alternatively, if the administrators have their own key pairs, they may issue their own delegation attribute certificates directly to other administrators. These new administrators are now able to define their own collaboration policies within their administrative scope.

An administrator sends a request to add or update a collaboration policy to the Admin PEP. The request may contain the administrator's delegated role(s) as signed credentials. To make this policy take effect in the target domain, the Admin PEP requests the CVS module to validate the administrator's claimed administrative role(s) and to return his valid ones. The CVS is either pushed the administrator's administrative credentials, or alternatively it may pull them from a repository. It validates them based on its administrative role credential validation policy and returns the valid administrative roles to the Admin PEP. The Admin PEP now needs to know if the presented collaboration policy is within the administrative scope of the

validated roles of the administrator. In order to do this, the Admin PEP creates a request to the PDP to validate either the role-permission assignments or the attribute-role mappings (or both) within the collaboration policy. The subject, action and target of the request are: the set of valid administrative roles, the Map or Assign action, and the local workflow role(s) or permission(s) respectively. If the request is granted, the access control and/or role mapping and credential validation policies will be stored in the PAP. The Admin PEP now informs the CVS and PDP about the new policies that have been added to the system and the CVS and PDP read them in. Any implementation of the Admin PEP, CVS and PDP should be able to perform their tasks automatically without human intervention.

The collaboration policies for one collaboration should be independent from those of all other collaborations, regardless of who is responsible for administering the policies. The consequences of this when evaluating user access requests are that either there should be a separate authorisation system (PDP and CVS) and associated policies for each collaboration, or if one authorisation system (PDP and CVS) makes access control decisions for multiple collaborations, then the policies for each collaboration must be kept separate and not combined. One way of doing this would be to have a unique collaboration ID for each collaboration, and to identify which collaboration each policy applies to and to which collaboration each user access request refers.

The SoA or the administrators who have permission to define a collaboration policy can also revoke an existing collaboration policy. In order to do this, they send a collaboration policy to the PEP along with a revoke request. The Admin PEP queries the CVS to get the valid administrative roles (and thus administrative scope) of a requestor and then queries the PDP in order to confirm that the requestor can define this policy. If he can then the Admin PEP removes the various policies from the PAP and informs the CVS and PDP of the removal. When an administrator's administrative role is revoked, we do not propose to automatically revoke any collaboration agreements that he might have established, since this may not be appropriate. Instead the SoA or replacement administrator can always revoke a collaboration policy by the method just described.

8. Enforcement of Sticky Policies Infrastructure

When we introduce the concepts of sticky policies[8] [19] into policy based authorisation infrastructures, the authorisation decision that is returned from a PDP may contain a security policy (as an obligation) along with the grant or deny result. This security policy is the sticky policy that is intended to accompany the retrieved data. Obligations are used to return the sticky policy, as obligations should be enforced by the PEP prior to granting the user access. However these sticky policy obligations typically wont be fully enforceable by the local PEP. They will need to be transported to the security system (PEP/PDP) of one or more remote sites for processing and enforcement along with the transferred data to which the sticky policy applies. For example, an outgoing message containing personal identifying information (PII) may be allowed to leave the current system, providing that the user's privacy policy is attached to it and that this policy is enforced by the PEPs/PDPs of every receiving system; or an outgoing confidential message may be allowed to leave the current domain providing that its contents are deleted by the receiving system within 7 days of receipt. We thus have the situation where the outgoing message needs to be supplemented with security policy information that is to be enforced by the receiving system. How is this to be achieved?

In this section we present three different possible approaches to solving this problem, which we call the *encapsulating security layer* model, the *application protocol enhancement* model, and the *back channel* model. All three models require the introduction of a new component, the application independent PEP (AIPEP) to be an interface between the conventional application dependent PEP and the existing application independent PDP. The AIPEP acts like a PDP to the PEP and a PEP to the PDP. The functionality of the AIPEP is to process and enforce the sticky policies and associated obligations that apply to multiple nodes of a distributed application. In addition, in the encapsulating security layer model it transports the application messages (see Figure 15), whilst in the back channel model it transports the policies (see Figure 16). The functionality of the PDP remains the same as in today's systems in all models (and hence the PDP remains application independent) whilst the functionality of the PEP may have to be modified to fit the new requirements of the AIPEP.

[8] These are policies that should be firmly attached to data, should travel with the data messages throughout a distributed system and should be enforced by each data processing node in the system.

8.1. The Application Protocol Enhancement Model

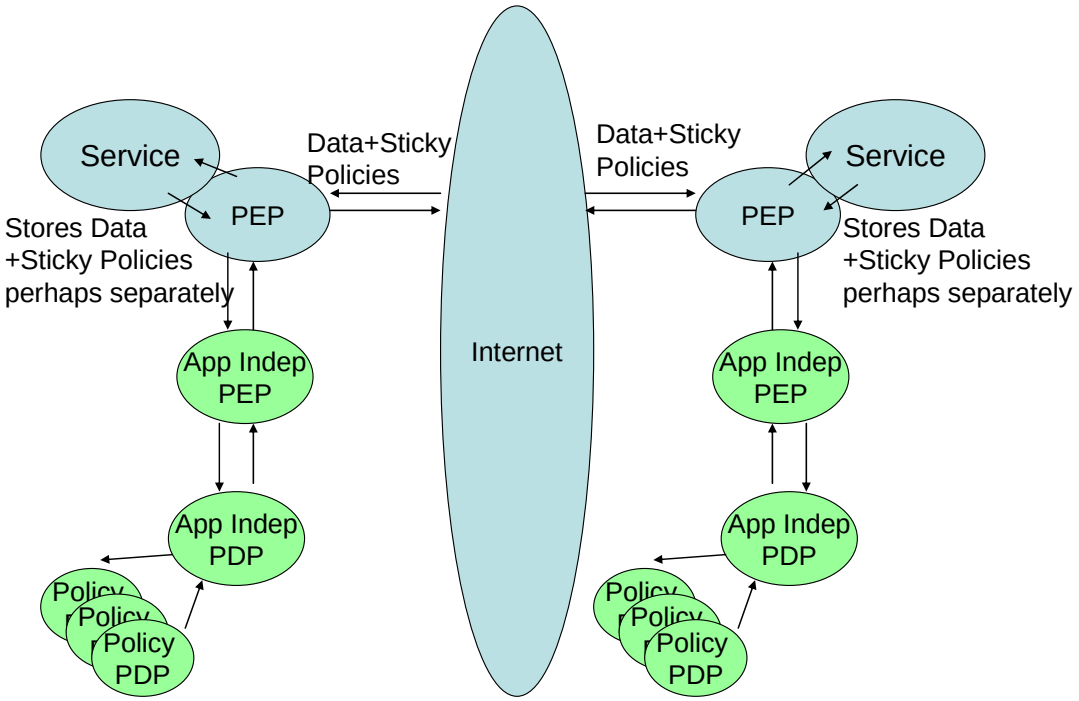


Figure 13. The Application Protocol Enhancement Model

In this model, the PEP supplements the existing application protocol with policy information. In a situation such as callout 1 or 3 in figure 2, the PEP receives and parses the outgoing message that is to be security enforced, segments and extracts relevant information from it (see Section 8.4), and passes this to the AIPEP for an authorisation decision. The AIPEP passes the request to the Master PDP. The Master PDP calls the relevant subordinate PDPs and returns the overall decision to the AIPEP. If this is Deny, this is relayed to the PEP whereupon this message segment should be discarded from the outgoing message and not transferred[9]. If this is Grant, the Master PDP optionally returns an obligation to the AIPEP saying what policy should accompany the outgoing message segment. The AIPEP passes this obligation to the PEP along with the authorisation decision, and the latter produces a policy packet that has to be attached to the outgoing message segment in an application dependent manner. The actual contents of this policy packet are transparent to the PEP, but should be internationally standardised so that all AIPEPs and PDPs can understand it. The PEP duly attaches this policy packet to the outgoing message segment, and may merge several segments together again before sending the message to the recipient system.

[9] A use case for this is where a set of patient records containing details of recent surgical operations are being transferred to a researcher for analysis, but the records include those of VIPs, which should be removed from the outgoing results.

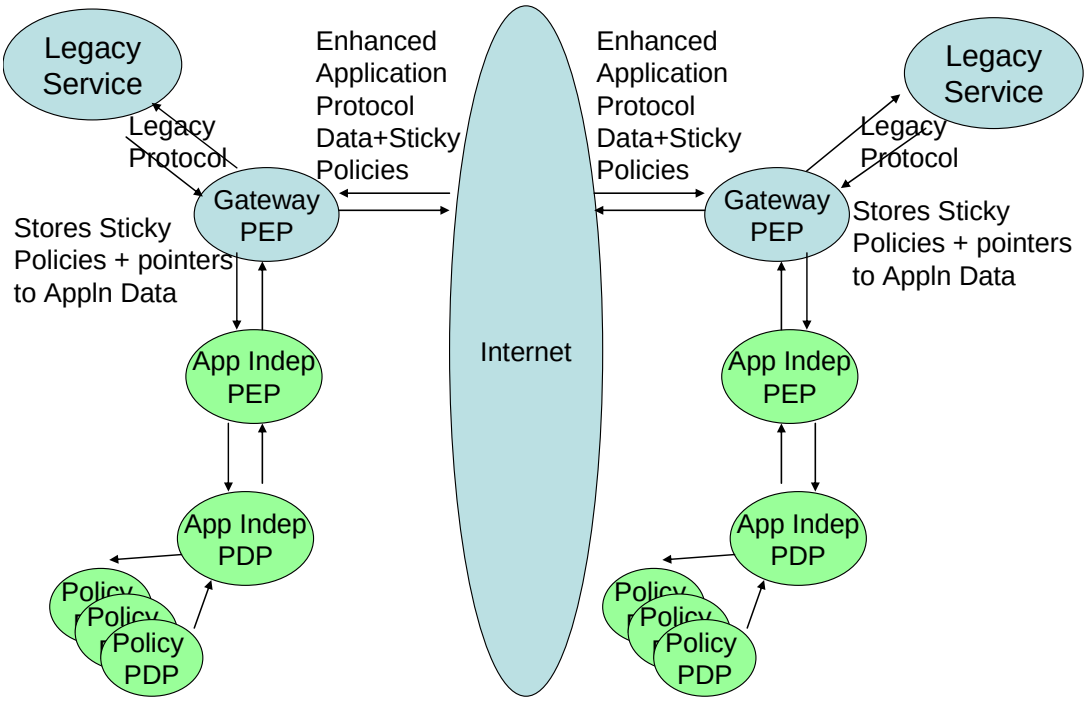


Figure 14. Using a Gateway for Legacy Applications

Some applications, such as S/MIME [20], are able to attach policies to data as they already have protocol fields that can be used for this. Other applications may not have such flexibility. In this case the PEPs could run in gateway machines and the communication between the application service and a PEP could be the existing legacy protocol, and the protocol between the PEP gateways could be an enhanced application protocol (see Figure 14).

When the message is received by the PEP at the receiving system, the PEP parses the incoming message, extracts the policy packets, message segments and relevant information for the PDP (see Section 8.4) and passes these to the AIPEP. The AIPEP parses this policy and processes it. Whilst the incoming policy processing is fully carried out by the AIPEP on message receipt, it may have knock on effects which will cause a similar policy packet to be attached to the same information when the latter is subsequently transferred elsewhere, or it may even deny the information from being transferred to other specific destinations. For example, if the policy packet is a user's privacy policy containing some blacklisted sites and the information is the user's PII, the AIPEP may update the PDP's policy to include a policy rule denying access to the blacklisted sites, and an obligation that says when this PII is granted permission to leave the local system the same sticky privacy policy should be attached to the data before it leaves the system.

When the AIPEP has finished processing the incoming policy, it calls the Master PDP for an authorisation decision on the incoming message segment. The Master PDP calls the subordinate PDPs, then makes the overall decision and if granted may

optionally return an obligation to the AIPEP. This obligation can contain any set of actions which are to be enforced locally by the AIPEP or PEP when it is processing the incoming message (as now). The encodings of the various obligations are such that the AIPEP knows which obligations it can process and enforce, and which it should return to the PEP for it to process. For example, in XACML [15], each obligation is given a unique URI. Note that if the Master PDP denies permission for the incoming message segment to be received, then the AIPEP will need to rollback the effects of any policy actions it has taken (such as updating the PDP's policy).

8.2 The Encapsulating Security Layer Model

In this model, the security layer is a protocol layer beneath the application layer. Each application layer protocol message segment is passed to the AIPEP, which wraps each message with its own security header and policy and then transfers the message to the AIPEP at the remote system, which strips off the security header, enforces the policy, and passes the message segment back to the application layer. This is similar to the SSL approach, only in this case the security layer is responsible for carrying policies between systems rather than MACs and encrypted messages.

In more detail the system works as follows. The PEP parses the outgoing message that is to be enforced, segments and extracts relevant information from it (see Section 8.4), and passes the message segment and the extracted information to the AIPEP for handling. The PEP must also pass the connection details of the recipient application system (i.e. the remote application endpoint) with the first message segment. The AIPEP calls the Master PDP - note that this is the same Master PDP as in the previous model - and receives an authorisation decision, which if denied, causes a deny to be returned to the PEP. The PEP can then decide whether to send the message minus the offending segment, or terminate the entire message. If granted, the Master PDP may optionally return an obligation, which is processed by the AIPEP to create a policy packet for attachment to the application message segment. In this model, the AIPEP can understand the standardised policy contents and therefore can potentially treat different outgoing messages in different ways e.g. encrypt some, sign some, use SSL etc, which is not something that the PEP in the previous model could do. The AIPEP sends the outgoing message to its peer AIPEP, and includes details of the recipient application system (i.e. application endpoint) with the first message. The AIPEP at the receiving system strips off the security header and policy, enforces the policy (as before), then calls the Master PDP (as before). The application message is finally passed to the PEP at the specified application endpoint.

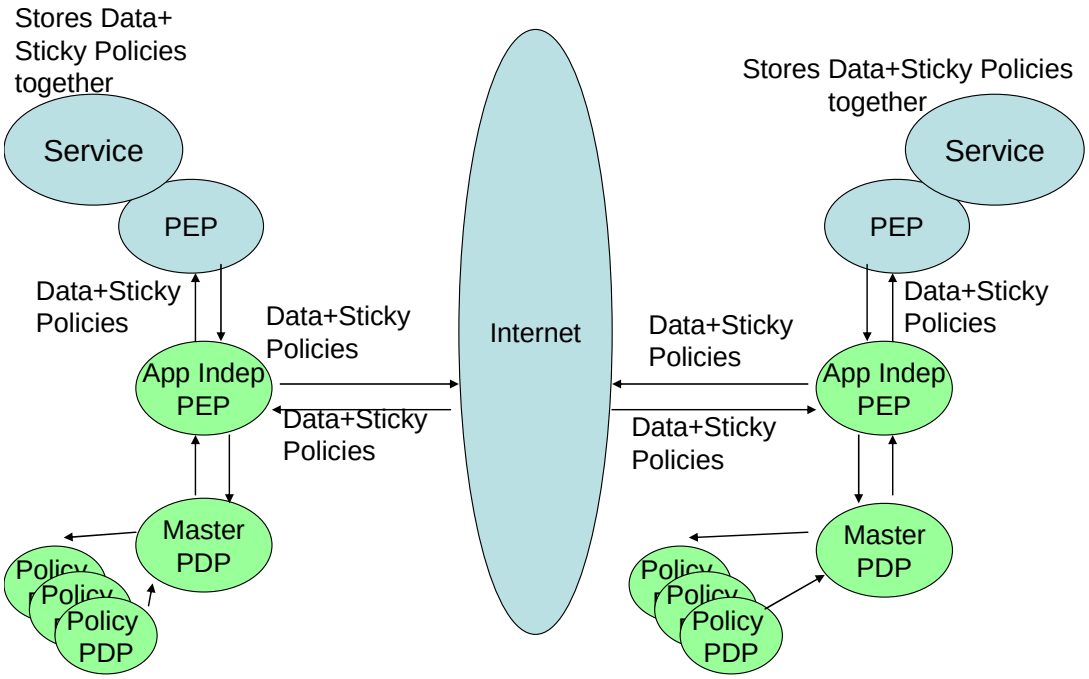


Figure 15. The Encapsulating Security Layer Model

Note that the AIPEPs will need to have their own standardised protocol and ports, so that they can talk to each other across the Internet. There will also need to be a mechanism for computing the endpoint information of an AIPEP given the application dependent endpoint information, as the latter is passed by the PEP on the sending side to its AIPEP.

8.3. The Back Channel Model

This model is least perturbing to existing distributed applications, since the AIPEP establishes a back channel with its peer AIPEPs in order to transfer the obligations and sticky policies that must accompany the application data.

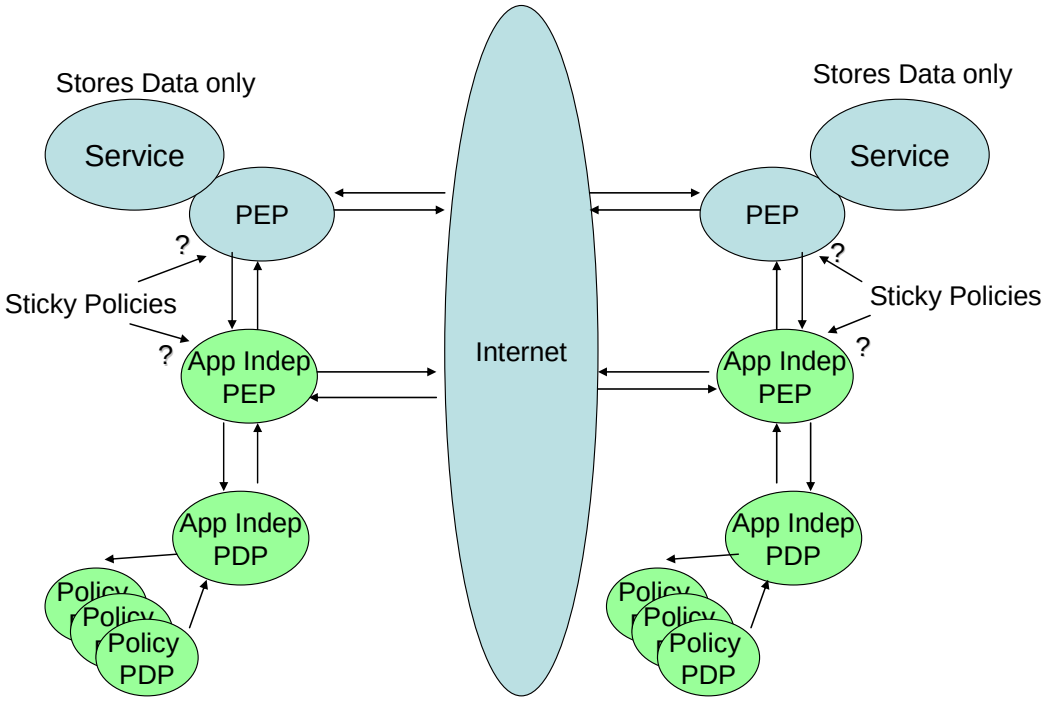


Figure 16. The Back Channel Model

The application works exactly as it does today, making a callout to the application independent infrastructure for a decision and enforcing the decision along with any local obligations that are returned. The application transfers outgoing application data to its peer without needing to modify the application protocol or data stream (except to remove any data blocks that are denied outgoing access). The AIPEP relays the decision request to the Master PDP and if an obligation to attach a sticky policy to the outgoing message is returned, the AIPEP needs to transfer this policy to its peer AIPEP before returning the grant decision to the PEP. There are two main issues here. Firstly, how does the sending AIPEP know who the receiving AIPEP should be? Secondly, how does the receiving AIPEP know which data block this policy applies to? The answer to the first question is that after trust negotiation has successfully completed (see Section 8.6) the trust negotiation module notifies the AIPEP with details about the remote trusted party. The answer to the second question is to attach the request context received from the PEP to the sticky policy, since the request context uniquely identifies the data block about which the decision has just been made (see section 8.4).

When the receiving PEP receives the incoming message and calls the AIPEP for a decision by passing it a request context, the applicable sticky policy should be ready and waiting for it, so that processing can continue as in the Application Protocol Enhancement model. Each AIPEP will need a temporary cache to store incoming sticky policies so that they can be linked to their data segments/request contexts when these subsequently arrive. There will be timing issues to consider, say if the

application data arrives before the sticky policy because the latter uses SSL whereas the application does not. These will need to be addressed during the implementation phase.

8.4. Functionality of The PEP

The standard information [2, 15] that the PDP (and hence the AIPEP) needs to be passed by the PEP is: details about the requesting user (i.e. subject attributes), details about the application destination (i.e. resource attributes), details about the requested action (i.e. action attributes) and any other relevant information (i.e. environmental attributes). In XACML, this is called a **request context**. Each application will typically store, format and transport this information in an application specific way, but the AIPEP needs it to be passed in a standard application independent way (for relaying to the PDP). The PEP is the only security entity that is capable of parsing an outgoing application message since it is the only security entity that understands the contents of the application specific message. So a primary function of the PEP is to parse each application message that needs an authorization decision, extract from it the information that the PDP requires (i.e. the request context), and format this into the standard format required by the AIPEP/PDP. In the TAS^3 project we will use the standard request context format specified in XACML. The precise information requirements of the PDP are dependent upon the application specific policy that is being used to make authorisation decisions. Consequently this information has to be determined in an application specific way since no two applications will require the same sorts of policies e.g. a mobile application might require authorisation decisions to be made based on the locations of the various mobile devices, whereas a printing application may have no such requirements but may instead require the time of day to be taken into consideration. Consequently, the precise contents of the information which the PEP has to extract from its messages to pass to the AIPEP/PDP have to be agreed by each application developer when the application is being built.

All the above models require the PEP to be responsible for parsing and logically segmenting outgoing messages into appropriate security blocks (and creating matching request contexts) so that each block can have a sticky policy applied to it. Whether the PEP passes the sticky policy to the AIPEP with the request context, or the AIPEP picks up the policy from its local storage, nevertheless it is always the responsibility of the PEP to provide the link between the policy and the security block/request context. A security block is an atomic unit of application data from a security perspective, which has a sticky policy attached to it (either physically or logically depending upon the model). A security block is defined as any application message, containing any arbitrary number of application elements, which has a common security requirement. Consequently, each security block may have a different security policy stuck to it. It is an application dependent matter to define what constitutes a security block. For example, the application may be transferring the names and addresses of a group of people in a single application message. In one application, each person may have the ability to set their own privacy policy for their own PII. At the application level, the group of names and addresses may be considered a single application level message for transfer to a remote site, but at the security level, each name and address may be considered to be a separate security block and therefore needs its own sticky policy. Thus it is the responsibility of the PEP to parse the outgoing message and to separate it into multiple security blocks,

and to call the AIPEP once for each block (i.e. name and address tuple in the message). In this way different privacy policies can be stuck to different name and address tuples. In another application a single corporate privacy policy may control access to the PII (names and addresses) of everyone in the database. In this case the entire message would be treated as a single security block and one privacy policy would be attached to the whole group of names and addresses. An application may transmit a large message with multiple elements as one security block, but the sticky policy might only refer to one particular element in the message. This is achieved by parsing the message, creating the request context from the one particular element, and then sticking the policy to the entire message. For example, a complete ePortfolio may be transmitted as a zip file accompanied with a sticky policy saying "The data from file A_identification_Pete.xml at XPATH location learnerinformation/identification/address must be used only for APEL purposes." Consequently it is the responsibility of the PEP to decide what constitutes a security block from the application's perspective and to act accordingly when calling the AIPEP. The result is that the AIPEP always receives the request context and either the sticky policy that applies to it or a reference to the policy.

8.5. Trust Negotiation

One issue that needs to be addressed by all three models is how does the sending system know if the receiving system can be trusted to obey the sticky policy that it receives? We propose that the well researched topic of trust negotiation [21] is used for this. Trust negotiation relies on trusted Attribute Authorities to issue credentials to components of a distributed application, in the form of signed attribute certificates or assertions, and during trust negotiation both the sender and recipient determine if the other party has the necessary credentials to participate in the interaction. We propose that trust negotiation is carried out by an application independent module during the process of service provider selection, prior to the transfer of the first application layer protocol message. In this respect, trust negotiation is independent of any of the models described here and of the application layer protocol, and is a necessary precursor of any application message and sticky policy transfer. Once trust negotiation has successfully completed, the trust negotiation module is in possession of all the necessary details about the remote party in order for the AIPEP or PEP to connect to its peer entity.

Note that in the encapsulating security layer model, it would be possible to build the trust negotiation functionality into the AIPEP rather than having it as a separate component. The peer AIPEPs would negotiate with each other to determine whether each is trusted or not, before the application data is transferred. If trust cannot be established by the AIPEP, the PEP would be informed that the application data cannot be transferred to the chosen application endpoint. The PEP would then be responsible for selecting another service provider/application endpoint and asking the AIPEP to try again. In the application protocol enhancement and back channel models, the PEP will need to call the trust negotiation module during service selection, prior to calling the AIPEP.

9. Event handling infrastructure & its application to adaptive audit controls

The TAS^3 infrastructure will incorporate an event handling infrastructure based on the publish/subscribe paradigm. Publish/subscribe is an asynchronous messaging passing infrastructure in which messages are grouped into classes. Unlike conventional messaging systems, in which senders send messages to specific receivers, in publish/subscribe messaging, senders (also known as publishers) send particular classes of message to receivers (also known as subscribers) who are interested in them. Consequently a message sender does not know how many recipients there might be for its messages; it all depends upon how many recipients have currently subscribed to receive that particular class of message. Subscribers express interest in one or more classes of message, and only receive messages that are of interest to them, without knowledge of what (if any) publishers there currently are.

Publish/subscribe messaging will be used by the TAS^3 infrastructure to send messages about specific security, privacy and trust related events. For example, if an administrator updates an authorization policy, a PDP can be sent an event message to inform it of the fact, so that it can read in the latest policy. The PDP will be a subscriber for messages of this class, and the policy management software will be a publisher of these messages. Publish/subscribe allows us to decouple system components, so that the components will work correctly regardless of whether there is another component sending or receiving messages or not.

A primary area in which we intend to use publish/subscribe is adaptive audit controls. Each TAS3 system component that sends log information to the secure audit web service (SAWS) [22] will subscribe for particular security/privacy/trust events, such as a security alert, or an occurrence of break the glass, and this will cause it to increase its level of logging that it sends to SAWS. Conversely, when an “all clear” or “glass re-set” message is transmitted, the component will reduce its level of logging to the minimum.

Researching into publish/subscribe mechanisms is not an objective of the TAS^3 project, and we do not propose to either design or spend significant resources developing our own publish/subscribe message passing infrastructure. We propose to use any suitable existing open source publish/subscribe messaging product that is available. If necessary we are prepared to enhance existing software to suit our requirements. It is for further study to determine what suitable products are available.

10. Authorization Ontology

10.1. Overview

This document describes the design of an ontology model for authorization policies.

The ontology model is based on the common ontology suggested in WP2. This is shown in Figure 17. There is a root called DOG (the same as owl:Thing) which represents the most general concept in the world. Then concepts are categorized into three top level classes: Activity, Entity and Descriptor. Activity represents anything that is happening or being done. An Entity is a thing with a distinct and independent existence. Descriptors are used to characterize an object or further describe another descriptor.

Two parts are included in the authorisation ontology model. The first part is the information representing the system to be governed by the policy; the second part is the policy rules that are used to get an authorization decision result for an access request to a specific system. All these classes are defined under three top level classes, and the full class hierarchy can be found in the ontology model section.

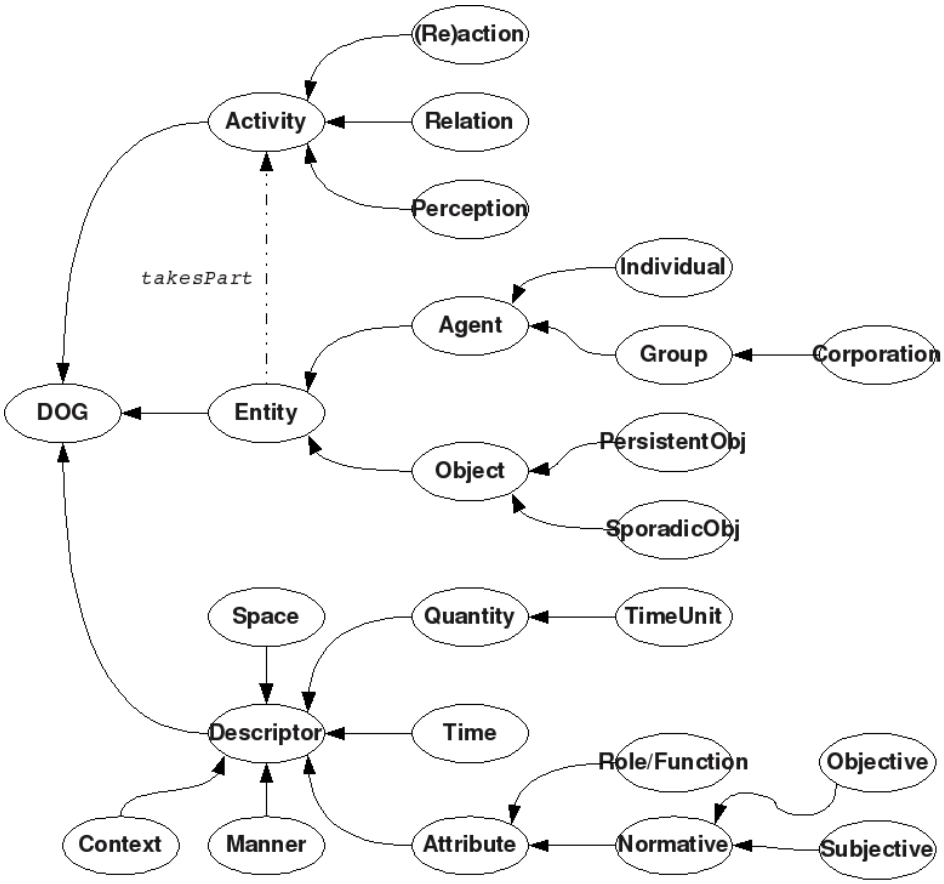


Figure 17. The Common Ontology

The major classes used to describe the system include Subject, Action, Resource, Environment and Attribute. The Action class represents operations (with parameters) that can be performed in a system. Resources are those valuable assets to be protected by a policy. Subjects are those who perform Actions. Environment is additional information about the system e.g. the current time, current state variables etc. Descriptors and Attributes are used to describe features of Subjects, Actions, Resources and Environment. An authorization result can be made based on the descriptors and attributes attached to Subjects, Actions, Resources and the Environment.

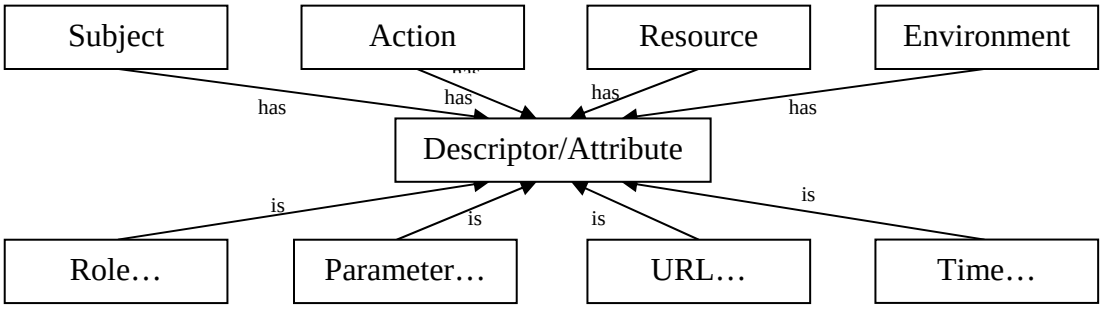


Figure 18. The System

The policy consists of a set of rules. Each rule is structured in the form of "if (Predicate) then (Result)", where Predicate is a statement that can be evaluated as either True or False by a PDP, and Result can be a Decision with optional Obligations.

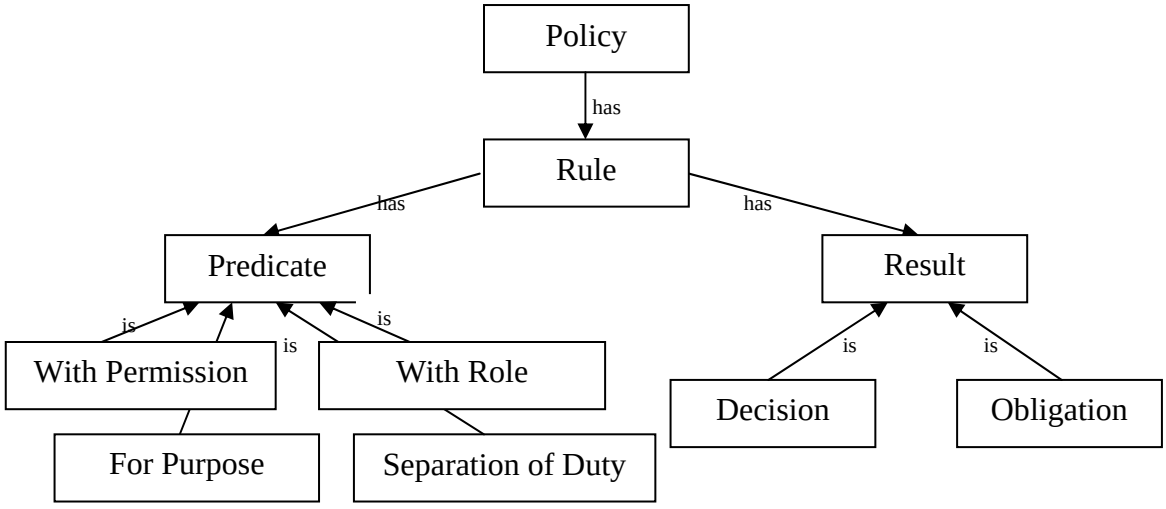


Figure 19. An Authorisation Policy

Predicates are assertions about the attributes of a system. They can be assertions about or constraints upon the subject attributes, action attributes, resource attributes or environment attributes. They can also be statements of

privacy, or constraints based on previous actions such as separation of duties (SoD). New predicates can be defined providing they can be evaluated by the PDP. A Predicate can include other Predicates as sub-predicates; a binary property of a Predicate indicates whether All sub-predicates or Any sub-predicate should be satisfied. A rule may include more than one predicate; only when all predicates are satisfied is the Result part of the rule returned by the PDP. There are two components of a result: Decision and Obligation(s). For an authorization result, there is one and only one Decision result, and any number of Obligations are optional.

A policy may contain more than one rule. Each rule has a priority, which is an integer ranging from 0 to 2^16; the higher the number, the higher the priority. There are two built-in rules with lowest priority 0, PermitAll and DenyAll. They are mutually exclusive and can be used to construct by-default-permit or by-default-deny policies.

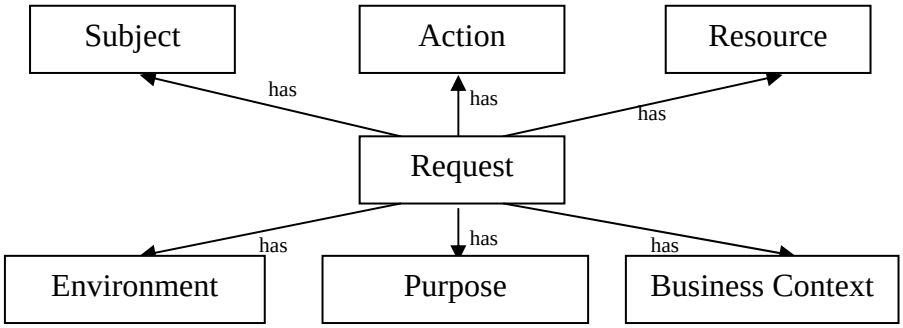


Figure 20. An Access Request

An access Request is modelled as a class in the ontology model. An access request may include Subject, Action, Resource, Environment, Purpose and Business Context. These objects and the descriptors/attributes associated with them will be used by the PDP to evaluate the predicates in the policy.

10.2. The Authorisation Ontology Model

In this section of the document, the authorisation ontology model will be described in an Object-Oriented style, rather than a Description-Logic style. This means that the model is structured in terms of classes, properties and instances. The hierarchy of all classes is shown in Figure 21. The description of each class is given below.

AccessRequest

AccessRequest is a Request by a Subject, who requests to perform an Action on a Resource for a Purpose. Other information that may be included in an AccessRequest can be the information of the Environment and the Context of the Action.

Properties:

Access_has_BusinessContext: Under what Context the Action is performed.
Access_has_Environment: The information of the Environment when Request is submitted
Access_has_Purpose: For what Purpose the Subject requests to access that Resource.

AccountObligation

AccountObligation specifies under which account the Action should be performed.

Properties:
underAccount: Under which account the Action should be performed. This could be a Subject or an Identity.

Action
Actions are operations allowed to be performed in a system.
Properties:
Action_has_Parameter: An action may have zero or more Parameters.
Action_on_Resource: This indicates on which resources an action can be performed on.

Activity
Imported from common ontologies defined by WP2.

Administration
Administration is a type of function like "do ACTION with ATTRIBUTE on USER/RESOURCE". An example is "ASSIGN the STAFF ROLE to ALICE". It can be associated with an Administrator, to state what privileges an Administrator has.
Properties:
onUser: The User that the AdministrationAction to be performed on.
performAdminAction: Defines which AdministrationAction to be performed in an Administration function.

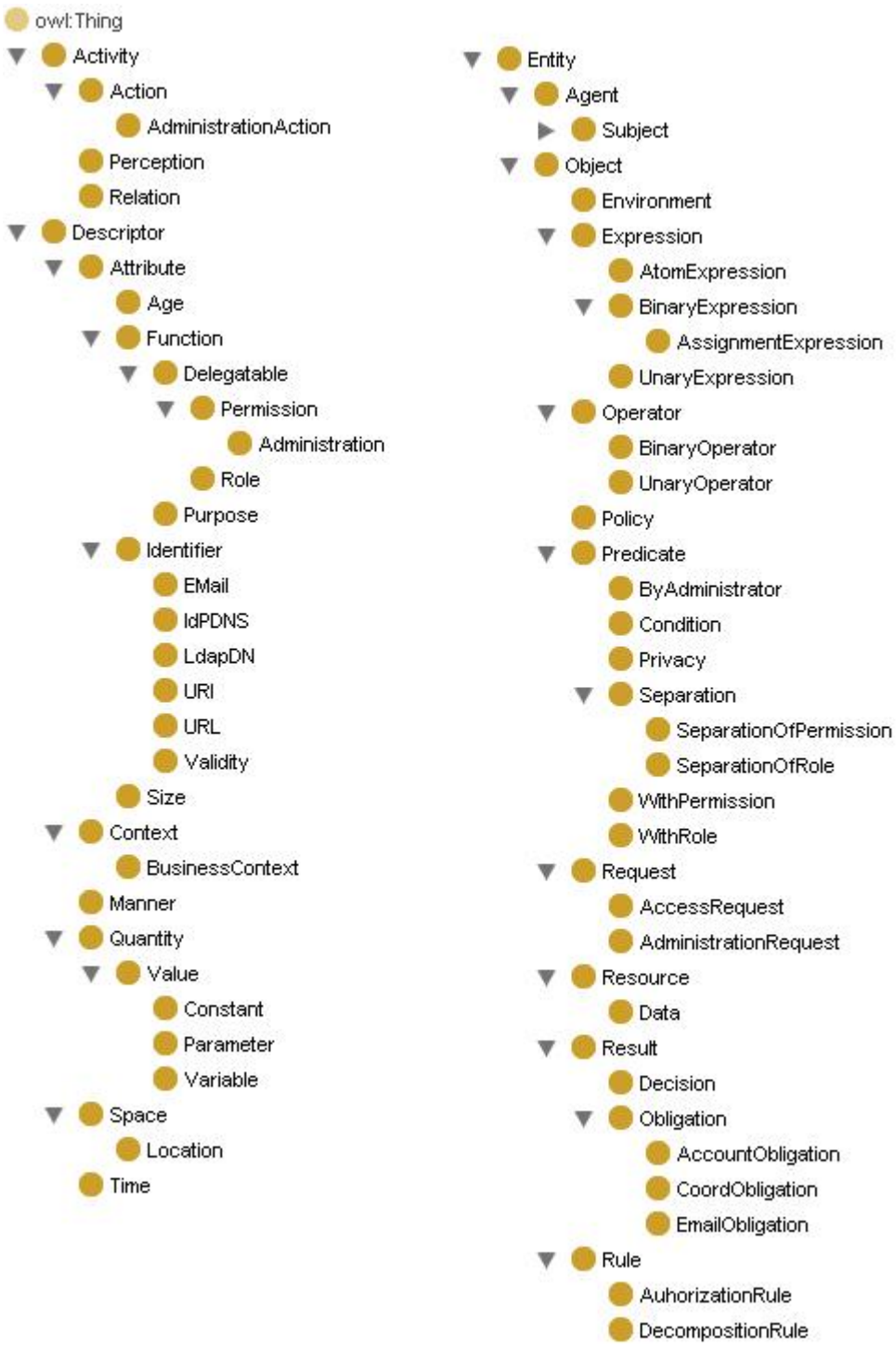


Figure 21. The Authorisation Ontology Class Hierarchy

AdministrationAction

AdministrationAction is a type of Action. It can be performed on Users or Resources, with Attributes as parameters.
Instances:
Delegate, Issue, Revoke

AdministrationRequest
AdministrationRequest is a Request to perform AdministrationActions on Users or Resources, such as issue or revoke an Attribute.

Administrator
Administrators are a kind of Subjects who can perform AdministrationActions in a system.
Properties:
Administrator_do_Administration: This defines Administration tasks associated with an Administrator.

Agent
Imported from common ontologies defined by WP2.

AssignmentExpression
AssignmentExpression is a type of Expression, which assign the value of right-hand side element to left-hand side element.

AtomExpression
AtomExpression is just an Attribute or a Value.

Attribute
Attributes are used to describe Entities and Activities in a system. They can be used to evaluate Predicates in rules.
Properties:
attributeValidity: An attribute may have a Validity period. Only within the period, the Attribute is valid.
issuedBy: This shows by which Subject this Attribute has been issued.

AuthorizationRule
Rules for Authorization. This kind of rules has one and only one Decision result, with optional Obligation result.
Properties:
Auhtorization_has_Decision: The Decision result of an Authorization rule.
Authroization_has_Obligation: Optional Obligations with the Decision result.
Instances:
PermitAll, DenyAll

BinaryExpression
A BinaryExpression has a BinaryOperator and two elements.

BinaryOperator
BinaryOperator is a type of Operator can take both LHS and RHS, such as Or.
Instances:

Assign, Divide, Sub, And, Equal, Or, Multiply, Add

BusinessContext
BusinessContext defines the boundary or a set of Actions performed in the system. Constrains like SoD (Separation of Duty) must be defined within a Context.
Properties:
contextString: A string used to identify a Context.
firstAction: The start Action of a business context.
lastAction: The end Action of a business context.

ByAdministrator
This Predicate states that a Request must be submitted by a certain Administrator.
Properties:
Assertion_on_Administrator: By which Administrator the administration tasks are taken by.

Condition
Condition is represented by boolean Expression, which tells when this Condition is True or False.
Properties:
conditionExpression: A boolean Expression of the Condition.

Constant
Constant is a value will not change.

Context
Imported from common ontologies defined by WP2.

CoordObligation
CoordObligation enables assigning an Expression to a Variable. For example, balance = balance - amount. This is represented by an AssignmentExpression.

Data
Data is a type of Resource, which can be further used by the requester. To protect Data from being misused, Privacy objects can be associated with Data to specify this Data can only be used for certain Purposes.
Properties:
dataPrivacy: Privacy constrains associated with a Data.

Decision
Decision is the Result of an Authorization rule; it could be Permit, Deny, NotApplicable or other type of Decision depending on the PDP implementation.
Instances:
Permit, Deny, NotApplicable

DecompositionRule
Decomposition rules are used to refine policies in high level to concrete policies which can be used for decision making.

Delegatable
This is used to represent a function which can be delegated to others. It only can be hold by a subject. When an attribute is delegated, its delegationDepth is deducted by one.
Properties:
delegationDepth: The depth an attribute can be delegated.

Descriptor
Imported from common ontologies defined by WP2.

EMail
An Email address.

EmailObligation
EmailObligation specifies an Email should be sent out in conjunction with the enforcement of a Decision.
Properties:
emailBody: The body of the Email to be sent out.
emailFrom: This specifies through which account the Email should be sent out.
emailSubject: The subject of the Email.
emailTo: To whom the Email should be sent.

Entity
Imported from common ontologies defined by WP2.

Environment
Environment includes the information, which is not part of the Request but is needed for making Decision. For example, current time of the system.
Properties:
Environment_has_Attribute: Attributes used to descripe an E nvironment.
Environment_has_Location: The Location information of an Environment.
Environment_has_Time: Current time of an environment.

Expression
General form of Expression used in a policy. Each Expression may have one operator, and one or two elements depending on the type of Operator. An expression can have just one element of an Attribute or a Value, in this case it is an AtomExpression. An Expression can also be nested as an element in other Expressions.
Properties:
expressionType: The value type of the Expression.
lhs: The first or left-hand side element of an Expression.
operator: The operator of an Expression.
rhs: The second or right-hand side element of an Expression.

Function
Functions represent combinations of Actions and Resources. Depending on entities a function is used to descript, it could be a permission, a role or a purpose. A function can include other functions, to form a function hierarchy.
Properties:

Function_comprises_Function: A function may comprise other functions to form a function Hierarchy.
onTarget: The target of the Action in the definition of a Function.
performAction: The Action to be performed in the definition of a Function.

Identifier
Identifier is used to identify a Subject or a Resource.
Properties:
identifierString: The string value of the identifier.

IdPDNS
The DNS name of a Shibboleth Identity Provider (IdP) server.

LdapDN
A DN (Distinguished Name) in a LDAP directory.

Manner
Imported from common ontologies defined by WP2.

Object
Imported from common ontologies defined by WP2.

Obligation
Obligation is an operation that should be performed by the PEP in conjunction with the enforcement of an authorization Decision.
Properties:
attributeAssignment: An AssignmentExpression representing the Variable and the value to be assigned to it.
fulfillOn: This specifies the Obligation should be performed in conjunction with which Decision. For example, an AccountObligation should be fulfilled on a Permit decision.
temporalType: TemporalType specifies at which stage of the enforcement of a Decision, the Obligation should be performed. There are three possible values of temporalType: Before, With or After. For example, the temporalType of an AccountObligation should be with.

Operator
Operator is used to construct expressions.

Parameter
Parameters of Actions can be used as Values in Expressions.
Properties:
parameterName: The name of a Parameter.

Perception
Imported from common ontologies defined by WP2.

Permission
A Permission attribute represents a subject is allowed to perform an Action on a certain Resource.

Policy
A policy consists of a set of Rules. When a Request comes to a PDP, the Predicate parts of these Rules are tested with the information from the Request. The Rules will be checked in the order of priority. If the Predicate part of a Rule is satisfied, then the Result part of that Rule will be returned by the PDP.
Properties:
Policy_has_Rule: Rules included in a Policy.
policyName: The name of a Policy.

Predicate
A Predicate is a statement can be evaluated by a PDP, with result of either True or False. The way of defining and evaluating a statement can be customized/extended according to the implementation of a PDP. A Predicate can be included in other ones as a sub Predicate.
Properties:
Assertion_on_Thing: This is a general form of assertion of a certain thing. It should be specialized in each type of Predicate.
satisfyAllOrAny: This indicates how sub Predicates should be satisfied, it can be either satisfyAll or satisfyAny.
subPredicate: Sub Predicates are those Predicates to be satisfied, besides the statements in the current Predicates.

Privacy
Stands for Privacy constrains on Data resource.
Properties:
forPurpose: For what reason a Data resource can be used.

Purpose
Purpose represents the reason of a Request to Data.

Quantity
Imported from common ontologies defined by WP2.

Relation
Imported from common ontologies defined by WP2.

Request
A Request to be answered by a PDP. A Request contains variant information which can be used by a PDP to evaluate Predicates. The PDP will check the Rules of a Policy with the information contained within the Request, and return the Results of the matching rule.
Properties:
Request_has_Target: On which target the Subject requests to perform Action on.
Request_has_Action: The Action that the Subject requests to perform on the Resource.
Request_has_Subject: The Subject who submits the Request.

Resource
Resources are those entities where Actions can be applied on. A set of Attributes can be associated with Resources. These Attributes can be used to construct Policies protecting the Resources.
Properties:
Resource_has_Attribute: Attributes associated with a Resource.
Resource_has_Identity: The Identity of a Resource.

Result
Result is the consequence of a Rule if the Predicate part of it is True. For an Authorization rule, it must have one Decision as its Result, together with Optional Obligations. For other kind of rules, such as reasoning rules like DecompositionRule, the result could be another Predicate which can be further used in other Rules.

Role
Role is a type of Attribute. In RBAC model, a Role is associated with a set of Permission. A user with a certain Role is allowed to perform Actions on Resources defined by the Permissions associated with that Role. Like some other Attributes, Role can be issued to a User by an Administrator of the system.
Properties:
Role_has_Permission: Permissions associated with a Role.
superiorTo: If Role A is superior to Role B, then Role A has all Permissions associated with Role B. This relationship is transitive.

Rule
Rule is the basic element in a Policy. Each rule has two parts, Predicates and Results. The Predciates can be evaluated by a PDP, and if the result is True, then the Results are returned by PDP.
Properties:
Rule_has_Predicate: The Predicate part of a rule. If there are more than one Predicates, all of them should be satisfied before the Results are returned. If a Rule needs "any predicate, then sub Predicates with "SatisfyAny" switch should be used.
Rule_has_Result: The Results part of a Rule. This part is returned by PDP if all Predicates are satisifed.
rulePriority: The priority of a Rule. It is an integer number ranging from 0 (the lowest priority) to 2^16 (the highest priority).

Separation
This is an abstract class representing Separation of Duties (SoD).
Properties:
inContext: In which Context the SoD constrains apply.
maxCardinality: The maximum cardinality of mutually exclusive objects a user may have in single Context.

SeparationOfPermission
This kind of SoD constrains state that in a certain Context, a User must not have more than maxmum number of Permissions from defined mutually exclusive Permissions.
Properties:

mutuallyExclusivePermission: Mutually exclusive Permissions in a Context.

SeparationOfRole
This kind of SoD constrains state that in a certain Context, a User must not have more than maxmum number of Roles from defined mutually exclusive Roles.
Properties:
mutuallyExclusiveRole: Mutually exclusive Roles in a Context.

Service
Services are parts of a system. They can perform actions with or with our user interaction. In some scenarios, Services can also be used to issue/revoke Attributes to other Subjects.

Space
Imported from common ontologies defined by WP2.

Subject
Subjects are entities who perform actions in a system. A set of Attributes can be associated with Subjects. These Attributes can be used to support evaluating Predicates in a Policy.
Properties:
Subject_has_Attribute: Attributes associated with a Subject.
Subject_has_Function: Subjects are associated with a set of Functions, in forms of Permission, Administration or Role.
Subject_has_Identity: The Identity of a Subject.

UnaryExpression
An UnaryExpression has an UnaryOperator and a single element.

UnaryOperator
UnaryOperator is a type of Operator can take only one element, such as Not.
Instances:
Not

URI
URI (Uniform Resource Identifier).

URL
URL (Uniform Resource Location).

User
Users are Subjects can perform Actions in a system.
Properties:
User_has_Role: User can have Role attribute, which is associated with a set of Permissions.

Validity
Validity is a time period, within which an Attribute or Predicate is valid.
Properties:

validateFrom: The start time of the Validity period.
validateTo: The end time of the Validity period.

Value
Values are used with operators to construct Expressions.

Variable
Variable's value can come from an Expression or properties of other objects.
Properties:
variableName: The name of a Variable.

WithPermission
This predicate states that a Permission must be associated with the Subject of the Request.
Properties:
Assertion_on_Permission: Assert that the request has the Permission to carry out the Action on the Resource.

WithRole
This predicate states that a set of Roles are required to satisfy this predicate. If the User in the Request has the Role asserted by this Predicate or a Role superior to it, then the Precate is true. If there are more than one Role asserted in this Predicate, then user need to have all these roles at the same time.
Properties:
Assertion_on_Role: Asserts that the User has a Role.

11. Conclusions

This document represents the first phase of the design of an identity management, authentication and authorisation infrastructure for the TAS[3] project. This design has been created during the first year of the project (based in part on the background knowledge and experience of the participants), and in parallel with this design, implementation of some of the functionality of the IdMAA has already been undertaken e.g. demonstration of Break the Glass policies. Implementation of others features are at various stages of development, and yet others has not yet been started. We expect that implementation experience and more detailed technical designs will further feed into this high level design document and will improve it. Integration of the IdMAA infrastructure with the application demonstrators and components from the other work packages will further inform the design. As such there is still significant further work to do in the coming years, and the design will be improved and modified in the light of further implementation experience, integration with the output of the other work packages, and piloting in the application demonstrators.

Limitations in the current design that we are currently aware of are:

i) we have not identified a suitable subscribe/publish messaging system nor designed exactly how it will fit into the TAS[3] infrastructure
ii) we are not sure exactly how the policy conflict resolution mechanism will work in the Master PDP
iii) we have not designed the way that sticky policies will be bound to their data
iv) we are not sure how complex it will be to build application dependent PEPs, but we know that we should keep this as simple as possible in order to minimise the work of application developers
v) we have not designed a language for specifying parameterised obligations
vi) we are aware that some obligations might require two phase commit type interfaces and that integrating this with applications, especially legacy ones, could be very problematic

We are sure that there will also be other limitations that we have not yet documented or are aware of.

We have undertaken the following standardisation activity during the first year of the TAS3 project:

i) We have participated in the Open Grid Forum. The author has chaired the OGSA Authorisation Working Group, and been the editor or co-editor of 4 OGF working documents that have progressed to final call during the year [23, 24, 25, 26]
ii) Various consortium members are members of the OASIS security technical committee and the Liberty Alliance project, and have attended various LA meetings throughout the year. In particular, David Chadwick and Sampo Kellomaki attended the LA meeting in Stockholm in July 2008 to present the design for attribute aggregation described in this document, and the mapping of it onto the LA Identity Mapping Protocol.
iii) David Chadwick is the UK BSI representative to ITU-T X.509 standards meetings. This year the ITU-T group completed the 2009 edition of X.509 and have now started work on the next version of X.500 which will include

protocols for password management, which is an important component of identity management.

12. Glossary

Attribute Authority (AA) – an authoritative source of subject attributes

Break the Glass (BTG) – a term used to describe an access control policy that allows users who would not normally have access to a resource, to gain access themselves by "breaking the glass" in the full knowledge that they will have to answer for their actions later to their management

Credential Issuing Service (CIS) – the service of issuing a digitally signed attribute assertions provided by an authoritative source of subject attributes

Credential Validation Service (CVS) – the service of validating digitally signed attribute assertions and determining which are trusted and which are not.

Federated Identity Management (FIM) – The communal services provided by a group of organisations which have set up trust relationships between themselves, so that they can send each other digitally signed attribute assertions about their users' identities in order to grant each others' users access to their resources.

Identity management, authentication and authorization infrastructure (IdMAA)

Identity Provider (IdP) **–** an authoritative source of subject attributes (i.e. an AA) that is also capable of authenticating subjects prior to issuing credentials.

Level of Assurance (LoA) – a metric which is used to measure the confidence (or assurance) that a relying party can have, that an authenticated user is really who they say they are. One scale, devised by the US National Institute of Science and Technology, ranges from 1 to 4, with 4 being the highest.

Policy Decision Point (PDP) – the (application independent) part of an access control system that can answer access control requests with a granted or denied decision.

Policy Enforcement Point (PEP) – the (application dependent) part of an access control system that is responsible for enforcing the decisions returned by the PDP.

Personal Identifying Information (PII) – personal information that can be used to identify someone

Privilege Management Infrastructure (PMI) – a highly scalable infrastructure, based on digitally signed attribute assertions, which allows subjects to be authorised to use the resources of relying parties based on their mutual trust in Attribute Authorities. A component of FIM.

Public Key Infrastructure (PKI) – a highly scalable infrastructure, based on public key cryptography, which allows subjects to authenticate to relying parties based on

their mutual trust in Public Key Certification Authorities (a type of TTP). A component of FIM.

Separation of Duties (SoD) – a security procedure whereby a high risk task is split into at least two sub-tasks which have to be carried out by different people.

Single Log Off (SLO) – the converse of SSO, whereby a user is simultaneously logged out of all the services that he is currently logged into via SSO.

Single Sign On (SSO) – the process whereby a user can sequentially gain access to a number of computer services by only providing his login credentials once to the first service he contacts.

Trust Management (TM) – the process of the management of trust between entities. Trust management may rely on many different factors such as the way an entity behaves (behavioural trust), the assertions of trusted third parties, or its performance against a set of trust metrics.

Trust Negotiation (TN) – the process whereby two entities negotiate a trusting relationship between themselves, by sharing their credentials that were issued to them by TTPs that both of them trust.

Trusted Third Party (TTP) – an entity that is trusted by other entities, usually so that the latter may be introduced to each other in order to establish trusted relationships between themselves.

Service Provider (SP) – an entity which offers some kind of electronic service to users.

X.509 A joint standard by the ITU-T, ISO and IEC which describes both PKI and PMI. X.509 public key certificates are ubiquitously used on the web for SSL/TLS communications with web servers.

13. References

[1] Trusted Architecture for Securely Shared Services, “Annex I - “Description of Work””. 29/11/2007
[2] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 “Security Frameworks for open systems: Access control framework”
[3] ANSI. “Information technology - Role Based Access Control”. ANSI INCITS 359-2004
[4] William E. Burr, Donna F. Dodson, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, Emad A. Nabbus. “Electronic Authentication Guideline”, NIST Special Publication NIST Special Publication 800-63-1, Feb 2008
[5] David W Chadwick, Linying Su, Romain Laborde. “Coordinating Access Control in Grid Services”. Concurrency and Computation: Practice and Experience, Volume 20, Issue 9, Pages 1071-1094, 25 June 2008.
[6] OASIS. "XACML v3.0 Obligation Families Version 1.0" Working draft 3. 28 December 2007
[7] R. L. "Bob" Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. “Federated Security: The Shibboleth Approach”. Educause Quarterly. Volume 27, Number 4, 2004
[8] Arun Nanda. “Identity Selector Interoperability Profile v1.0” Microsoft Corporation, April 2007. see http://download.microsoft.com/download/1/1/a/11ac6505-e4c0-4e05-987c-6f1d31855cd2/Identity-Selector-Interop-Profile-v1.pdf
[9] David Chadwick, George Inman, Nate Klingenstein. “Authorisation using Attributes from Multiple Authorities – A Study of Requirements”. Presented at HCSIT Summit - ePortfolio International Conference,16-19 October 2007, Maastricht, The Netherlands.
[10] Hodges, J. Cahill, C.(Editors). “Liberty ID-WSF Discovery Service Specification V2.0”. Liberty Alliance Project
[11] OASIS. “Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0”, OASIS Standard, 15 March 2005
[12] OASIS. "Level of Assurance Authentication Context Profiles for SAML 2.0". Working Draft 1. 1 July 2008
[13] Hodges, J. Aarts, R. Madsen, P. and Cantor, S.(Editors). “Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification v2.0”. Liberty Alliance Project.
[14] Hodges, J. Kemp, J. Aarts, R. Whitehead, G. Madsen, P. “Liberty ID-WSF SOAP Binding Specification v2.0” Liberty Alliance Project.
[15] OASIS “eXtensible Access Control Markup Language (XACML) Version 2.0” OASIS Standard, 1 Feb 2005
[16] D.W.Chadwick. "Dynamic Delegation of Authority in Web Services" in "Securing Web Services: Practical Usage of Standards and Specifications". Edited by Dr Panayiotis Periorellis, Newcastle University. Idea Group Inc. 2008. pp111-137 ISBN 978-1-59904-639-6. Information about the book can be found at http://www.igi-global.com/reference/details.asp?id=6976
[17] ISO 9594-8/ITU-T Rec. X.509 (2001) “The Directory: Public-key and attribute certificate frameworks”

[18] O. Bandmann, M. Dam, and B. Sadighi Firozabadi."Constrained delegation". In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages131-140, Oakland, CA, May 2002. IEEE Computer Society Press.
[19] Mont, M.C.; Pearson, S.; Bramhall, P. "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services". Proc 14th Int Workshop on Database and Expert Systems Applications, 1-5 Sept. 2003. Page(s): 377 – 382
[20] B. Ramsdell et al. "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification". RFC 3851. July 2004
[21] Bertino, E., Ferrari, E., Squicciarini, A.: Trust Negotiations: Concepts, Systems and Languages. IEEE Computer, pp. 27-34, 2004.
[22] Wensheng Xu, David Chadwick, Sassa Otenko. "A PKI based secure audit web service". IASTED Communications, Network and Information Security CNIS, November 14 - November 16, 2005, Phoenix, USA
[23] V. Venturi, T. Scavo, D.W. Chadwick, "Use of SAML to retrieve Authorization Credentials", 7 April 2008.
[24] David Chadwick, Linying Su. "Use of WS-TRUST and SAML to access a Credential Validation Service". OGF Authz WG Draft, 1 June 2008
[25] David Chadwick, Linying Su, Romain Laborde. "Use of XACML Request Context to access a PDP", OGF Authz WG Draft, 31 March 2008
[26] David Chadwick. "Functional Components of Grid Service Provider Authorisation Service Middleware". 6 April 2008

Document Control

Amendment History

Version	Date	Comments
0.1	13 Dec 2008	Initial version by David Chadwick and Lei Lei Shi
0.2	16 Dec 2008	Commented version from Marc Santos
0.3	22 Dec 2008	Updated ontology model from Lei Lei Shi
1.0	31 Dec 2008	Included all comments from internal reviews, and added missing sections such as Glossary and Exec Summary