Chadwick, David W., Otenko, Sassa and Nguyen, Tuan Anh (2009) ***Adding Support to XACML for Multi-Domain User to User Dynamic Delegation of Authority.*** **International Journal of Information Security, 8 (2). pp. 182-196.**

Adding Support to XACML for Multi-Domain User to User Dynamic Delegation of Authority

David W Chadwick[1], Sassa Otenko[2] and Tuan Anh Nguyen[1]

[1]University of Kent, Computing Laboratory, Canterbury, Kent, CT2 7NF

[2]Oracle Corporation UK Ltd., Oracle Parkway, Thames Valley Park, Reading, Berkshire. RG6 1RA.

[1]Tel: +447796447184

[1]Fax: +44 1227 762 811

Email: d.w.chadwick@kent.ac.uk, sassa.nf@gmail.com, tn32@kent.ac.uk

[1]URL: http://www.cs.kent.ac.uk/people/staff/dwc8/index.html

Authors Biographies

David Chadwick is Professor of Information Systems Security at the University of Kent, and the leader of the Information Systems Security Research Group. He is a member of IEEE and ACM. He has published widely, with over 120 publications in international journals, conferences and workshops, including 5 books and 12 chapters in books. He specializes in Trust Management, Public Key Infrastructures and Privilege Management Infrastructures. Current research topics include: policy based authorisation, privacy protection, identity management, the management of trust, the delegation and recognition of authority between domains and autonomic security.

Dr Alexander (Sassa) Otenko has a specialist degree in Applied Mathematics and Computing from Sumy State University in the

Ukraine, where he was the top student in his year. He also has a PhD from Salford University in the UK. Sassa moved to the University of Kent with David Chadwick in 2005, where he was appointed as a research associate. He helped to tutor Tuan as he studied dynamic delegation of authority for his PhD. Sassa was the main architect and Java programmer for the PERMIS privilege management software which he developed whilst studying for his PhD under Professor Chadwick. He subsequently completed implementing the SAML interface to PERMIS, and integrating PERMIS into Globus Toolkit.

Tuan Anh Nguyen has a Master of Electronics and Telecommunications degree from the Hanoi University of Technology, Vietnam. He is studying for a PhD in dynamic delegation of authority at the University of Kent, under the supervision of Professor Chadwick.

Abstract. We describe adding support for dynamic delegation of authority between users in multiple administrative domains, to the XACML model for authorisation decision making. Delegation of authority is enacted via the issuing of credentials from one user to another, and follows the role based access control model. We present the problems and requirements that such a delegation model demands, the policy elements that are necessary to control the delegation chains and a description of the architected solution. We propose a new conceptual entity called the Credential Validation Service (CVS) to work alongside the XACML PDP. We describe our implementation of the CVS and present performance measurements for validating delegated chains of credentials.

Keywords*. XACML, RBAC, Delegation of Authority, Credential Validation Service.*

1 Introduction

XACML [8] is an OASIS standard for writing access control policy languages in XML. Many people are starting to experiment with it in their applications e.g. [11, 12, 25]. XACML was designed as a general purpose access control policy language for protecting resources from being accessed by subjects who are identified by their attributes. Some of XACML's benefits include: a flexible attribute based authorisation model, where access control decisions can be made based on the attributes of the subject, the action, the target and the environment; a comprehensive way of specifying conditions, so that arbitrarily complex conditions can be specified; and the support for obligations. An obligation is an action that should be performed by the application's policy enforcement point (PEP) when enforcing the access control decision made by the policy decision point (PDP). XACML policies can be built that support role based access controls (RBAC) [5], in which users are assigned to roles, and roles are given permissions to access resources. An RBAC profile for XACML has been published [21]. This requires two XACML PDPs to be constructed. One, the access control PDP, determines if the holder of a set of roles can be granted access to a resource. The other, the role assignment PDP, can only be called by the Role Authority to determine if a particular user is allowed to hold a particular role. There are a number of deficiencies in this profile. Firstly the role assignment PDP has no knowledge about who is a Role Authority and who is not, and therefore it cannot make decisions about who can assign roles. Similarly it cannot support static or dynamic delegation of authority because in these cases the Role Authority can change. Consequently this function has to be performed by the application dependent PEP. We would like it to be application independent. The XACMLv3 working draft [18] which has been undergoing development for several years, is concerned with the administration and delegation of policies, rather than the delegation of attributes from one user to another. Therefore it is complementary to the work described here.

A delegate is defined as "A person authorized to act as representative for another; a deputy or an agent" [1]. 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. Delegation of authority is thus the act of one user with a privilege, the delegator, giving it to another user (the delegate), in accordance with some delegation policy. Static delegation of authority is when the delegation policy contains the complete list of delegators. No new delegators may be created without updating the delegation policy. Dynamic delegation of authority is when the delegation policy only contains the initial set of delegators (the trusted root delegators) and further delegators can be created dynamically through the act of delegation.

In the context of role based access controls there are two aspects to delegation of authority. One is the delegation of a role from one user to another. This is delegation of the user-role assignments. The other is the delegation to assign permissions to roles from one user to another. This is delegation of the role-permission assignments. The later is sometimes referred to as delegation of policy administration. In this paper we are only concerned with the former acts of delegation, where one user delegates his role to another user, so that the latter can carry out the functions of the former. This is the usual form of delegation in organisations. Delegation of policy administration on the other hand is a specialised function usually only carried out by security administrators, and we will not cover that in this paper.

When one user delegates a role to another user, who then delegates to another user, recursively, a directed acyclic graph (DAG) is created, starting from the root user who has the role initially and is the source of the DAG, to the users at the sink nodes of the DAG who end up with the authority to assert the delegated role, but cannot delegate it further themselves. Intermediate nodes are users with permission to delegate, but may or may not be able to assert the role themselves (according to the delegation policy). We differentiate between static and dynamic delegation of authority as follows. Static delegation of authority is when intermediate nodes are not allowed in the DAG and all the delegators are configured as root nodes in the software (or policy) prior to users accessing resources i.e. the depth of the delegation DAG is known from the start to be one, and no intermediate nodes can be created. Dynamic delegation of authority is when only the root user nodes and delegation policy are configured into the software prior to user access, and users may dynamically delegate authority to other users as and when they wish. In this case the delegation DAG is created dynamically as one user delegates to another, and new leaf and intermediate nodes are created spontaneously.

A responsive authorisation infrastructure that can cater for rapidly changing dynamic environments should be able to validate the roles given to any of the users in a dynamically created delegation DAG, even though the actual DAG is not known when the authorisation policy is written and fed into the PDP. This requires the authorisation policy to be supplemented with a delegation policy that will state how the delegation DAG is to be constrained. As long as a user's role falls within the scope of the delegation DAG then it is considered valid, if it falls outside the DAG, and thus outside the delegation policy, it is not. The purpose of the current research was to add dynamic delegation of authority to an authorisation infrastructure that contains an XACMLv2 access control PDP (or in fact any PDP that bases its access control decisions on the attributes of subjects), without changing the XACMLv2 PDP or its policy[1].

We assume that permissions are assigned to roles, or attributes in the more general case, and that the attributes are assigned to the users. An important point to clarify at the outset is the difference between an attribute and a credential (i.e. authorisation credential). An attribute is a property of an object[2]; a credential is a *statement* or *assertion* about an attribute. In particular, a credential must state: what the attribute is, who the attribute belongs to, who says so (i.e. who is the credential issuer), and if there are any policy constraints on its validity. When attributes of an entity do not exist as part of the entity, they are often stored or transferred as separate stand alone credentials. In this paper we are concerned with dynamic delegation of authority from one user to another by the use of credentials. One important feature of a credential is that it requires validation before the user can be attributed with the asserted property.

The rest of this paper is structured as follows. Section 2 describes the problems that need to be addressed when creating an infrastructure to

[1] Note that this research originally started whilst XACMLv2 was still under construction, when it was known that XACMLv2 would not support dynamic delegation of authority. This was one of the reasons for not proposing changes to XACMLv2. Work is currently underway to add administrative policy delegation to XACML v3 [18], but this is complementary to the work described here.

[2] Dictionary.com defines an attribute as “A quality or characteristic inherent in or ascribed to someone or something”

support dynamic delegation of authority between multiple domains, and this leads to various requirements being placed on any proposed solution. Section 3 describes the new conceptual credential validation service (CVS) that is proposed to resolve the problems and requirements described in Section 2. Section 4 briefly describes the XACMLv2 infrastructure. Section 5 discusses how the CVS could conceptually be incorporated into the XACML infrastructure. Section 6 describes our implementation of a CVS, and provides some performance measurements of its operation. Section 7 concludes, and looks at possible future work in this area.

2 Problem and Requirement Statements

The underlying model used for dynamic delegation of authority in multiple domains is an enhancement of the basic XACMLv2 model (see section 5). In this enhanced model a user (subject) is dynamically given a set of attributes by one or more dynamically created attribute authorities (AAs) in one or more domains, and these attributes, in the form of credentials, are presented (pushed) to or obtained (pulled) by a new functional component of the authorisation infrastructure which we call the Credential Validation Service (CVS). The CVS validates the user’s credentials and returns the valid attributes directly or indirectly to the PDP. The PDP then makes its access control decisions based on its policy, the validated set of subject attributes, the target and environmental attributes and the parameters of the user’s request. Below are a set of issues and requirements that need to be addressed by the new functional component in such a model.

1. **Valid vs. Authentic Credentials.** The first thing to recognise is the difference between an *authentic* credential and a *valid* credential. An *authentic* credential, from the perspective of authorisation decision making, is one that has been received exactly as it was originally issued by the AA. It has not been tampered with or modified. Its digital signature, if present, is intact and validates as trustworthy meaning that the AA's signing key has not been compromised, i.e. his public key (certificate) is still valid. This means that the public key certificate has been issued by a known trusted root CA or one of its subordinates, and has not been revoked since then. A *valid* credential on the other hand is one that is trusted by the CVS's policy for authorisation decision making. In order to clarify the difference, an example might be an employee certificate issued by the University of Kent. This credential is authentic, since it has been issued by the University of Kent's AA. The credential is also valid for accessing staff resources at the University of Kent. However, the credential is not valid if used via eduroam to access staff resources at the University of London (yet it remains authentic).
2. **Credential validity is determined by the target domain**. The above discussion leads onto the second problem that needs to be addressed in any solution, and this is that there are potentially *multiple domains* within an authorisation infrastructure. There are issuing domains, which issue credentials, and target domains that consume credentials. The CVS is part of the target domain, and as such it must use the policy of the target domain to decide whether a

credential is to be trusted or not i.e. is valid or not. So the validity of an authorisation credential is ultimately determined by the (writer of the) CVS policy. A valid credential is a credential that is trusted by the consumer of the credential.

3. **Multiple trusted credential issuers.** In any system of any significant size, there will be multiple credential issuers. Some of these will be trusted by the target domain, others will not be. Thus the system must be capable of differentiating between trusted and untrusted issuers, and of dynamically obtaining this information from somewhere. (In point 4 below we propose to use roots of trust.) Different target domains in the same system may trust different issuers, and therefore the CVSs must be capable of being flexibly configured via their policies to say which issuers are trusted and which are not. For example, in the physical world of shopping with credit cards, there are several issuers such as Amex and Visa. Some shopkeepers accept (trust) both issuers, others only trust one of them. It is their (the target domain's) decision which card issuers to trust.
4. **Identifying roots of trust.** Point 3 above leads us to conclude that the CVS must be configured, in an out of band trusted way, with at least one authorization (or Privilege Management Infrastructure – PMI) root of trust and it is from these PMI roots of trust that all credentials must be validated in order to be trusted. A *PMI root of trust* must be a single entity identified directly or indirectly by its

public key[3], since this key will be used to validate the signed credentials that are received. Note that it is not possible to refer to a PMI root of trust through its set of assigned attributes, e.g. anyone with a project manager attribute and company X attribute, since these attributes may identify several candidate roots, and may be issued by several attribute authorities, in which case it wont be known who to trust. This implies that a higher authority is the real PMI root of trust, the one who issues the set of attributes that can be trusted.

5. **The role of the Issuer's policy.** Most issuers will have an Issuing Policy that states its rules for issuing credentials and places constraints on the use of the issued credentials. This policy will include any delegation policy to say which delegates are allowed to act as delegators and delegate which credentials to which users. Consequently there will be constraints on which credentials are deemed to be valid for which purposes within and without the issuing domain. However, the target domain may choose to ignore these constraints and trust (treat as valid) credentials which the issuer deems to be invalid. A well known example in the physical world concerns supermarkets who issue their own discount coupons. These coupons state quite clearly that they are only valid for use in supermarkets owned by the issuer. However, it is often

[3] When an X.509 conformant PKI is used which already has its own configured CA root public keys, the globally unique name of the subject in the PKI certificate can be used to refer to the authorization root of trust, instead of the public key in the certificate, in which case the subject will be trusted regardless of which public/private key pair it is currently using.

the case that a different brand of supermarket will accept these discount coupons as a way of enticing the other supermarkets' customers to come and shop in their own supermarket. Thus the CVS must have a way of either conforming to or overriding the issuer's policy. If a target domain chooses to ignore the issuer's policy, then it is liable for any losses incurred by this. The issuer cannot be held responsible for targets that ignore its Issuing Policy.

6. **Obtaining the Issuing Policy.** In a multi-domain system, the target domain may not be aware of the issuing domain's Issuing Policy, unless it is explicitly placed into the issued credentials. If the complete Issuing Policy is not explicitly placed in the issued credentials, but the target domain still wishes to enforce it and only treat as valid those credentials that the issuer says are valid, then the target's CVS will need to infer or be configured with the issuer's Issuing Policy. For example, in SPKI [7], a credential is marked as being infinity delegatable or not, and does not contain any other details of the Issuing Policy, such as who is entitled to be delegated the permission. Thus unless a delegatable credential explicitly contains restrictions, or out of band means are used to transfer them, the target CVS will infer than anyone is entitled to be delegated this credential.
7. **Pulling credentials.** The CVS may not have all the credentials it needs in order to validate the credential(s) presented by the user, e.g. if only the credential of a sink node in a delegation DAG is presented, but none of the intermediate node credentials are presented. In the most extreme case the user may not present any

credentials at all, for example, when a user logs into a portal and the portal displays only the services this user is allowed to see, the portal has, unknown to the user, retrieved the user's credential(s) from a repository in order to determine which services to display. There is thus a strong requirement for the CVS to be able to pull credentials before the PDP can make access control decisions.

8. **Discovering credential locations**. The user's credentials may be stored and/or issued in a variety of places, for example, each AA may store the attributes or the credentials it issues in its own repository. One could always mandate that the user collects together the credentials he wants to use, before attempting to gain access to a resource e.g. as in the VOMS model [13]. Alternatively the user could send references to the locations of the credentials rather than the credentials themselves. Both of these models have their merits, but they are not always very user friendly. In fact, in some cases, the user may not be aware what credentials have actually been issued to him or where they are stored – he might only know what services he is allowed to access, as in the portal example given above. Thus, in addition to being pushed credentials the CVS must also be capable of contacting different repositories/AAs in order to pull the user's credentials prior to making its access control decision.
9. **Multiple user identities.** If the user is known by different identities to the different AAs, then there must be a way for the user to use these mixed credentials in the same session. The GridShib project currently uses a mapping table to convert between X.509

PKI identities and Shibboleth identity provider identities [14]. But a more flexible approach is needed, in which the user may determine which set of credentials are to be used in a given session and the CVS can prove the user's right to assert each one. We propose one solution to this in [20].

10. **Multiple credential formats.** Following on from above, the user's credentials may be created in different formats and stored in different repositories, and therefore presented to the CVS in different ways, e.g. as signed SAML assertions [2], as X.509 attribute certificates [3], as Shibboleth encoded attributes [4] etc. The CVS needs to be able to decode and handle credentials in different formats. In [25] the authors propose a Credential Conversion System (CCS) to handle this problem, by the CCS re-issuing credentials in the local format. However this changes the trust model by introducing trust in the CCS rather than trust in the credential issuer. We prefer to keep to the original trust model where credentials are issued by their authoritative sources.
11. **Hierarchies of attributes.** The attributes may form some sort of hierarchy, for example in accordance with the ANSI RBAC specification [5], in which the superior attributes (or roles) inherit the permissions of the subordinate roles. The CVS needs to be aware of this hierarchy when validating the credentials. For example, if a superior role holder delegates a subordinate role to another user, then the CVS needs to know if this delegation is valid or not, given that the attributes are different. Furthermore some of the attributes known to the CVS won't form a hierarchy. Therefore

the CVS needs to be able to cater for multiple disjoint attribute hierarchies.

12. **Constraining credential validity**. Only part of an authentic credential might be valid in a target domain. For example, a credential might contain multiple attributes but the target domain only trusts the issuer to issue a subset of the enclosed attributes.
13. **Understanding and differentiating attributes**. If each credential issuer puts its own proprietary attributes into its credentials, no other VO partner will be able to understand and use them. But for inter domain authorisation, remotely issued attributes need to be understood by the receiving domain. There are two possible solutions to this problem: standard attributes and attribute mappings both of which are described below.

The function of attribute mappings is to map external (unknown) attributes into internally known ones. With this approach the unique combination of issuer, attribute type and attribute value are mapped into a locally understood attribute value or role, and the remote users with these credentials inherit the permissions that are granted to the local attribute value or role. Of course, this requires the mapping function to be configured with knowledge of these unknown attributes so that correct mappings can be performed, but this is a tractable problem.

The alternative approach to attribute mappings is to define a common set of standard attributes. This allows the attributes that are issued by every issuing domain to be understood by each receiving domain. This is a feasible approach for federations. Perhaps the most extreme example of this are credit cards such as Visa. Visa cards (a plastic credential) are issued by hundreds of different issuing banks, but all contain the same Visa logo (attribute) and all are treated as being equal by the relying parties, regardless of the issuer. The US academic community also adopted this approach some years ago, by defining the EDU person schema [6], which is a collection of standard attribute types, and in some cases (for the affiliation attributes) standard values as well. However, in the general case it will never be possible to define standard sets of values for all attributes that will provide sufficiently fine grained access control both for and between all organisations in a VO. Locally defined values of standard attributes will become the differentiating factor for fine grained control. In order to prevent clashes of attribute values, Internet 2 defined a standard encoding format for values to provide them with global uniqueness so that no two organisations should issue the same value for attributes which are inherently different. However, not all organisations may conform to these uniqueness rules, and there is no way of enforcing it. Thus standard attributes on their own can never fully resolve the problem of understanding and differentiating between attributes. For example, suppose most organisations in the world issue a standard Project Manager attribute to their project managers. In a VO between organisations A and B, the CVS policy for organisation B might only trust the Project Manager attributes issued by itself, and not those issued by organisation A (or by C or D or any other organisation). Or alternatively it might wish to downgrade those issued by organisation A and treat them as being equivalent to a guest user attribute. Or it might decide to trust the project managers from A as being equal to its own project managers. Thus some form of attribute mapping may still be required even when standard attributes are used.

In conclusion, the CVS's policy needs to be flexible enough to cater for all the above requirements, including the ability to perform attribute mappings.

3 Architecting a Solution

Given the problem statements and various requirements from above, one can see that all these new functional requirements cannot be met by existing PDPs. Consequently, we have proposed a new conceptual component called the Credential Validation Service (CVS), whose purpose is to perform the new functionality. In essence the purpose of the CVS is to validate a set of credentials for a subject, issued in different formats by multiple dynamic attribute authorities from different domains, according to the local policy, and return a set of valid attributes. How this conceptual component is merged into the XACML infrastructure will be described later. There are several reasons for making the CVS a separate component to the XACML 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[4]. 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. Secondly, the XACML language is incapable of specifying credential chains and therefore handling delegated credentials in a request context. This is because subjects and attribute issuers are identified differently in the language (subjects are identified by any attributes of any data type whilst the attribute issuer is an optional string), hence it is not possible to chain delegated credentials together.

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.

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

3.1 The Trust Model

The CVS needs to be provided with a trusted master credential validation policy[5]. 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. 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

[5] 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.

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 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 (see section 2 point 5), 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.

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. All attributes need to be included in this hierarchy; both externally defined ones and internally understood ones. 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*. Other attributes usually will not e.g. the names of organisations. 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.

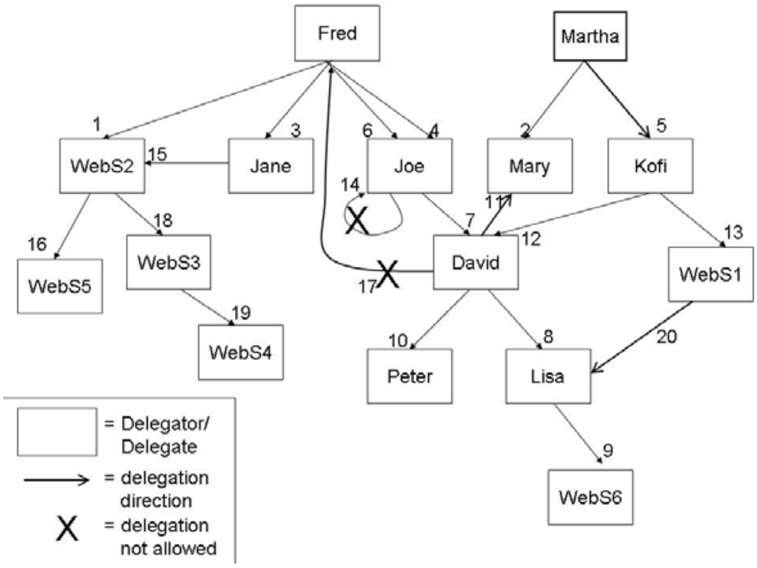

Figure 1. An example Delegation Directed Acyclic Graph

- 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 (see Figure 1). 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 (e.g. edges 14 and 17 in Figure 1). 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, or set of trees, in which there is only one source or PMI root node for each tree which holds the set of attributes that it can delegate, 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, such as edges 7 and 12 in Figure 1, then these are represented as separate edges and nodes in the tree, without merging the delegate nodes together. Figure 2 shows how the DAG of Figure1 might be simplified into two delegation trees. 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 [3] and a different approach in [9]. The essential elements however should specify who is allowed to be in each tree (both as an issuer and/or a subject), what attributes they can validly have (assert) and delegate, and what constraints apply.

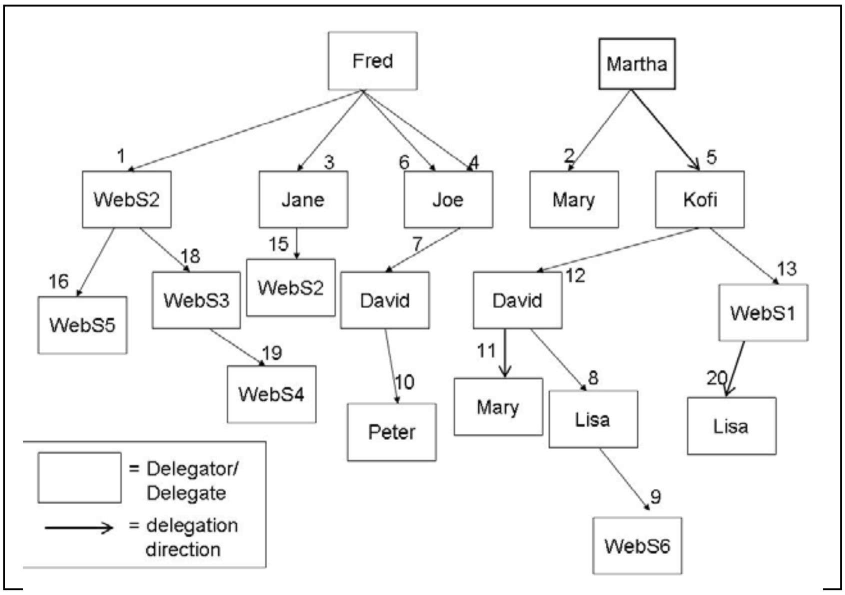

Figure 2. An example set of Delegation Trees

- 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. Consider for example time constraints. An issuer gives each issued credential a validity period, which may range from fairly short (e.g. minutes) to very long (e.g. several years). The primary reason for issuing short lived certificates (for other than intrinsically short lived permissions) is so that they do not need to be revoked, and therefore the relying party does not need to consult revocation lists, white lists, or OCSP servers etc. In the case of relatively long lived credentials, the CVS policy author may have his own opinion about which credentials to trust, from a chronological perspective, and therefore may wish to place his own additional time constraints on remotely issued credentials. For example, a plumber may have a "certified plumber" credential, which is valid for 10 years from the date of issue. He may be required to pass a competence test every ten years to prove that he is conversant with the latest technology developments and quality standards before the credential is renewed. However, in the target domain, the CVS policy author may decide that he does not want to accept anyone with a credential that is newer than one year old, due to insufficient experience on the job, or is more than 8 years old, due to doubts about competencies with the latest technologies. Consequently the CVS must be told what the local constraints are on credential validity.
- the attribute mapping policy that maps externally defined attributes into ones known by the local PDP and used in its access control rules.

- 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 to support static delegation of authority. In this case the delegation trees reduce to one level hierarchies, in which the root nodes are the (static) set of trusted issuers and the first level nodes 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, but the subjects are not allowed to delegate further.

XACMLv2 [8] or its RBAC profile [21] are not suitable instruments to express Credential Validation Policies. As the RBAC profile states "*The policies specified in this profile do not answer the question "What set of roles does subject X have?" That question must be handled by a Role Enablement Authority, and not directly by an XACML PDP*". The current working draft of XACMLv3 [18] is not suitable either. 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. The approaches used by XACML can only state that an asserted set of attributes or policies is fully accepted, or fully rejected. In [18] the delegation is deemed to be valid if the issuer of the delegated policy could have performed the request that the policy grants to the delegatee. We think this is a serious deficiency, which lies at the core of the XACML policy evaluation process. We think it is a limitation on an independent issuing domain to have to take into account all the policies that the validating domains support, so that only fully acceptable sets of credentials or policies can be issued to its subjects. 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.

Having identified this problem, we propose a solution that uses a non-XACML based credential validation policy first, and an XACML policy next for access control decision making that uses the delegated attributes that have been validated by the CVS's policy.

3.2.1 Formal Credential Validation Policy

We define a Credential Validation Policy as an unordered set of tuples <S, I, C, E>, where S is a set of Subjects to whom any Issuer from set I can assign at most a set of Credentials C, but only if any of the conditions in set E holds true:

$$CVP = \{<S, I, C, E>\}$$

We define the Credential Validation process as a process of obtaining a subset of valid credentials V, given an asserted set of credentials c, issued by issuer i to the subject s, if condition e holds true at the time of evaluation:

$$V = \{ c \cap C \mid c \cap C \neq \emptyset, s \subseteq S, i \subseteq I, e \subseteq E, <S, I, C, E> \subseteq CVP \}$$

Note that in XACML the only possible evaluation of a Credential Validation process is:

$$V = \{ c \mid c \subseteq C, s \subseteq S, i \subseteq I, e \subseteq E, <S, I, C, E> \subseteq CVP \}$$

Further, we define a dynamic delegation process as a process of obtaining a set R of Credential Validation rules for intermediate issuers, i.e. the issuers on the path from the policy writer to the end user, where the intermediate issuer s is issued a set of Credentials c by a higher level issuer i, subject to condition e and a constraint on subject domain d:

$$R_s = \{ <d \cap S \backslash s, s, c \cap C, e> \mid c \cap C \neq \emptyset, s \subseteq S, i \subseteq I, e \subseteq E,$$

$$<S, I, C, E> \subseteq CVP \cup R_i \}$$

Thus the issuer i can allow the issuer s to delegate a subset of his own permissions to a subset of his own set of subjects, subject to the condition e being stricter than that imposed on i. Note the recursive nature of the process - the tuple <S, I, C, E> must belong to the CVP or to the set of valid rules for issuer i. Note also that loops in the delegation are prohibited by excluding the holder of the rule from the set of possible subjects. XACML currently lacks the expressiveness for deriving new Credential Validation rules given the set of existing rules and valid credentials.

3.3 The CVS functional components

Figure 3 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

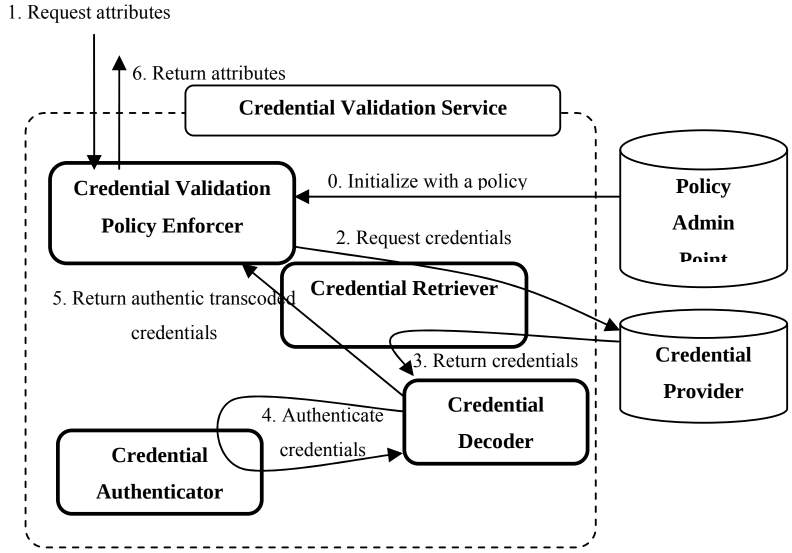

Figure 3. Data Flow Diagram for Credential Validation Service Architecture

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).

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 set of delegation trees, 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 them further in Section 6 where 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 validating domain.

4 The XACML Model

Figure 4 shows the overall conceptual set of interactions, as described in XACMLv2 [8]. The PDP is initially loaded with the XACML policy prior to any user's requests being received (step 1). The user's access request is intercepted by the PEP (step 2), is authenticated, and any pushed credentials are validated and the attributes extracted (note that this is not within the scope of the XACML standard). The request and user attributes (in local format) are forwarded to the context handler (step 3), which may ask the PIP for additional attributes (steps 6 to 8) before passing the request to the PDP (step 4). If the PDP determines from the policy that additional attributes are still needed, it may ask the context handler for them (step 5). Optionally the context handler may also forward resource content (step 9) along with the additional attributes (step 10) to the PDP. The PDP makes a decision and returns it via the context handler (step 11) to the PEP (step 12). If the decision contains optional obligations they will be enforced by the obligations service (step 12).

As can be seen from Figure 4, XACMLv2 currently has nothing to say about credentials or how they are validated.

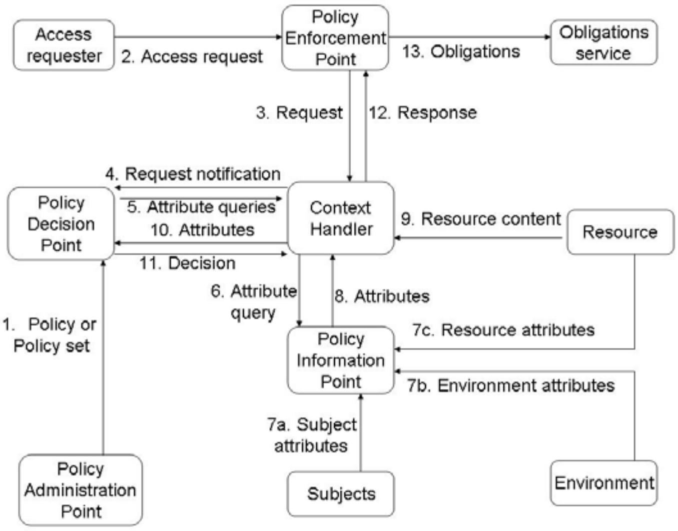

Figure 4. Data Flow Diagram for XACML Architecture

5 Incorporating the CVS into XACML

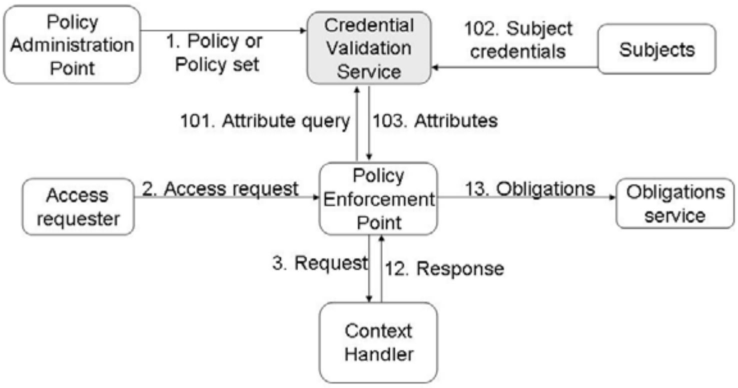

Figure 5. Incorporating the CVS by directly calling it from the PEP

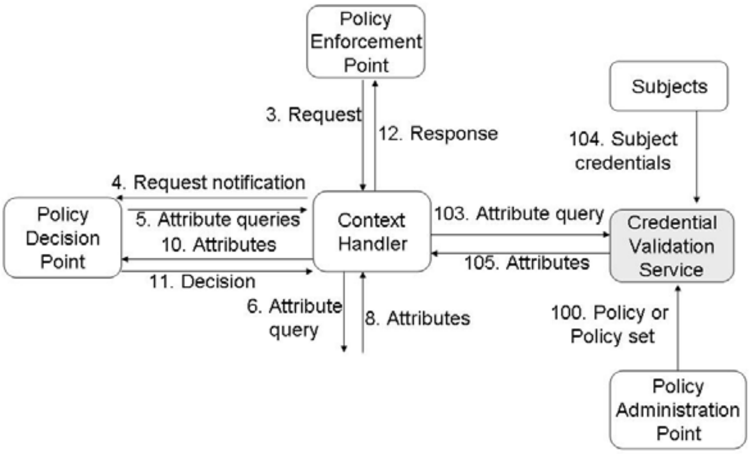

Figure 6. Incorporating the CVS as an additional module called by the XACML Context handler

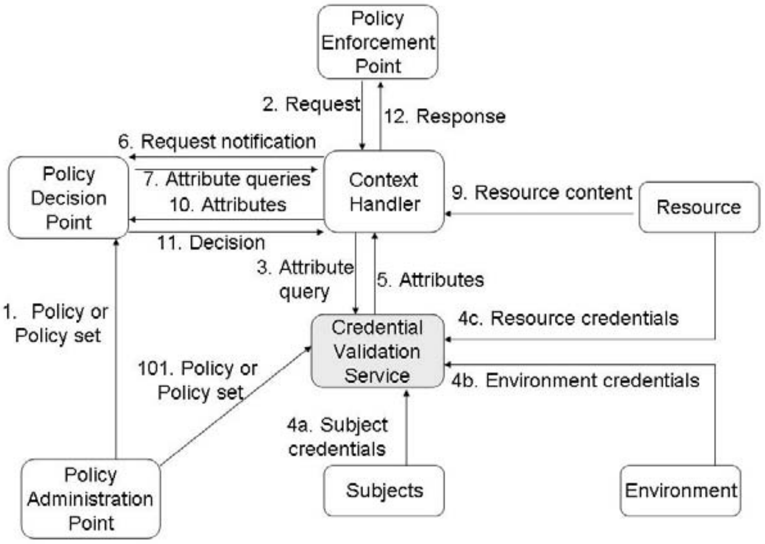

Figure 7. Incorporating the CVS as a replacement for the XACML PIP module

Figures 5, 6 and 7 show the three possible ways in which the CVS could be incorporated into the XACML model. The CVS could be an additional component called by either the PEP (step 101 in Figure 5) or the context handler (step 103 in Figure 6), or it could completely replace the PIP (step 6 in Figure 7).

The primary advantage of having the CVS called by the PEP (Figure 5), is that existing XACMLv2 implementations do not need to change. Furthermore some PEPs, such as Globus Toolkitv4 [27], already implement this model. When GT4 is pushed a complete set of credentials, it makes a call-out to an external plug-in to process them before calling the PDP. When called by the PEP, the CVS is either passed a complete bag of credentials (push mode), or it fetches all the credentials it needs (pull mode), or a mixture of the two modes occurs and the CVS pulls the additional attributes that it needs to supplement those pushed by the PEP. In all cases the CVS returns the valid set of subject attributes to the PEP, which the latter can then pass to the existing XACML context handler. The primary disadvantage of this model is that each application will need to be modified in order to utilise the CVS, since the PEP is an application dependent component of the authorisation infrastructure. Note that this model, when operating in push mode only, with no credential retrievals, is similar to that being proposed by the WS-Trust specification, in which the Security Token Service (STS) operates as a token validation service [10]. However, the STS has no equivalent functionality of the CVS operating in credential pull mode.

The advantage of having the CVS called by the context handler is that many existing applications i.e. the PEPs, will not need to change. If subject credentials can be packaged like subject attributes and hence relayed transparently from the access requestor to the context handler, or the CVS operates in pull mode, then a PEP can call an enhanced context handler without needing to be modified. The only change that will be needed is to the context handler component of an XACML implementation. Support for multiple autonomous domains that each support delegation of authority can be added to applications without either the application logic or the context handler interface needing to change. Only a new credential validation policy is needed. Credentials that were previously invalid (because they had been delegated) would now become valid, once the appropriate policy is added to the PAP.

The advantage of completely replacing the PIP by the CVS, is that we have the opportunity of using digitally signed credentials for constructing target attributes and environmental attributes as well as subject attributes. For example, time may be obtained from a secure time stamping authority as a digitally signed credential (step 4b in Figure 7), and validated according to the CVS's policy. The disadvantage of the last two approaches is that incorporating the CVS inside the policy evaluator introduces transforms to the request context that are invisible to the PEP.

At the current time we do not know which approach will eventually be favored. Consequently in our implementation we support two modes of operation: the CVS called directly by the PEP and the CVS called by the XACML context handler. We have also specified two Open Grid Forum profiles for the protocol interactions between a PEP and an enhanced context handler/PDP [22] and a PEP and a CVS [23] so that credentials can be transparently processed by XACML applications. Note that the mode of operation does not affect the implementation of the CVS, which is explained in the next section.

6 Implementing the CVS

There are a number of challenges involved in building a fully functional CVS that is flexible enough to support the multiple requirements outlined in section 2. Firstly we need to fully specify the Credential Validation Policy, including the rules for constructing delegation graphs (or multiple trees). Then we have to engineer the policy enforcer with an appropriate algorithm that can efficiently navigate the delegation graph (or a tree) and determine whether a subject’s credentials are valid or not. In our implementation we have chosen to constrain the delegation DAG into a set of delegation trees, with each tree having a single PMI root of trust. The output from the CVS is a set of valid attributes encoded in XACML format ready for passing to the PDP.

6.1 Credential Validation Policy

Comment [DWC1]: Add credential decoding

We have implemented our CVS policy in XML, according to the schema shown in Appendix 1. Most components of the policy are relatively straightforward to define, apart from the delegation trees. We have specified the list of trusted credential issuers (PMI roots of trust) by using either their subject distinguished names (DNs) or their subjectAltName Uniform Resource Identifiers (URIs) from their X.509 public key certificates. Only the latter subjectAltName is supported since this is the naming scheme used by all entities on the world wide web. We chose to use DNs or URLs rather than public keys for two reasons. Firstly, they are easier for policy writers to understand and handle, and secondly it makes the policy independent of the current key pair that happens to be in use by a trusted issuer. The authorisation policy is therefore independent of the underlying PKI.

Multiple disjoint attribute hierarchies are supported. Each attribute hierarchy is specified by listing superior-subordinate attribute value pairs. This allows any arbitrary partial order to be created, since there is no limit to the number of times a particular attribute value can occur as either a superior or a subordinate value in one hierarchy (subject to the restriction that loops are not created). Attributes and attribute values can be independent of any hierarchy if so wished, so that permission inheritance does not have to be supported if it is not required.

Delegation trees have each been defined as a name space (a delegation domain), a delegation depth and a root of trust. Anyone in the delegation domain who is given a credential by the designated root of trust may delegate it to anyone else in the same domain, who in turn may delegate it to anyone else in the same domain until the delegation depth is reached. X.500/LDAP distinguished names or HTTP URLs are used to define the delegation domains. A base DN or URL is used to specify the root node of the delegation domain, and the domain may be refined by defining included and excluded subtrees so that any arbitrary subtree may be constructed. All delegates must belong to the refined domain otherwise the delegation is not valid. Since we already refer to the credential issuers (roots of trust) by their LDAP DNs or URLs, it was natural to refer to the delegates in a delegation tree by their DNs or URLs as well. In this way we can easily link delegation chains together by matching the issuer in one certificate with the subject in the next certificate in the chain. We recognise that a more flexible approach to defining delegation trees is by referring to delegates by their attributes rather than their DNs or URLs, as for example as used by Bandmann et al [9]. Their delegation tree model allows a policy writer to specify delegation trees such as “anyone with a head of department attribute may delegate a project manager attribute to any member of staff in the department”. This is a future planned enhancement to our work. It introduces a level of indirection and complexity whereby one has to retrieve a delegate or delegator’s credentials, extract their attributes from this, see if they have an attribute that matches the one in the delegation rule, and then validate that this attribute was correctly assigned or delegated in the credential according to its governing rule. This adds a level of complexity that our current model does not have, since in our current model we simply need to match on the delegate or delegator’s name.

One obvious constraint that we place on our delegation trees is that the same attribute value (or one of its subordinate values in the role hierarchy) must be propagated down any given tree from the root of trust, and either new unrelated attributes that are not in the same role hierarchy, or superior values from the same role hierarchy, cannot be introduced in the middle of a delegation tree. This is to ensure that a delegator can only delegate his existing permissions or a subset of them, and not an unrelated set or superset. A new delegation tree would need to be specified for the delegation of an unrelated or superior attribute.

Trusted issuers and delegation domains are defined separately in the policy and then linked together with the attributes that each issuer is trusted to issue, along with any additional time/validity constraints that are placed on the issued credentials. (The constraints have not been shown in the schema.) The reason for doing this is improved flexibility, since one trusted issuer may be the root of several delegation trees, and one delegation domain may have several roots of trust.

In our current implementation we do not pass the full Issuing Policy along with the issued credential, we only pass the tree *depth* integer, since this was already an X.509 standard extension. Therefore the CVS does not know what the issuer's intended delegation tree is. We have assumed that the credential issuing software at the issuing site will enforce the Issuing Policy and so only credentials that conform to the Issuing Policy will be issued. However, the CVS policy writer is able to specify his own delegation domain for the received credentials and this may be more restrictive than that of the issuing domain, or the same as or less restrictive than it. So ultimately the owner of the resource will control the delegation tree that is deemed to be valid at the target site. In order to ensure that the Issuing Policy is enforced at the target site the issuer's delegation tree should be configured into the CVS's policy. This assumes that the structure of the issuer's delegation tree is the same as that of our CVS policy, which will not always be the case in independent domains using different models and software implementations. A future planned enhancement is to carry the complete Issuing Policy in each issued credential, and to allow the CVS's policy writer to enforce it, or overwrite it with his own policy, or force conformance to both. In this way a more sophisticated delegation tree can be adhered to. This of course will depend upon there being a standardised format for the transfer of Issuing Policies in credentials, which currently there is not for either SAML attribute assertions or X.509 or SPKI certificates.

6.2 Delegation Tree Navigation

Given a subject's credential, the CVS needs to create a path between it and a root of trust, or if no path can be found, conclude that the credential cannot be trusted. There are two alternative conceptual ways of creating this path, either top-down, also known as backwards [3, 17] (i.e. start at a root of trust and work down the delegation tree to all the leaves until the subject's credentials are found) or bottom-up, also known as forwards (i.e. start with the subject's credential and work up the delegation tree until you arrive at its root of trust). Neither approach is without its difficulties. Either way can fail if all the credentials are not pushed to the CVS. If the CVS has to pull credentials from the issuers or their repositories, then all the credentials have to be held consistently – either all with their subjects or all with their issuers, otherwise the CVS will not be able to efficiently locate them. In our implementation all credentials are held with their subjects, typically in their LDAP directory entries, or more recently, in files linked to their DNs held in WebDAV repositories [24]. As Li et al point out [17], building an authorisation credential chain is more difficult in general than building an X.509 public key certificate chain, because in the latter one merely has to follow the subject/issuer chain in a tree, whereas in the former, a DAG rather than a tree may be encountered. Graphs may arise for example when a superior delegates some permissions in a single credential that have been derived from two of more credentials that he possesses, or when attribute mappings occur between different authorities. Our CVS implementation is currently limited to supporting delegation trees rather than DAGs, and so it will not follow multiple superior credentials from a single subordinate one as these are forbidden. Delegations are also restricted to occurring in a single subject domain, and therefore attribute mappings will not occur. But even for the simpler PKI certificate chains, which our credential chains conform to, there is no best direction for validating them. SPKI uses the forwards chaining approach [15]. As Elley et al describe in [16], in the X.509 model it all depends upon the PKI trust model and the number of policy related certificate extensions that are present to aid in filtering out untrusted certificates, whether backwards or forwards chaining is preferrable. Given that our delegation tree is more similar to a PKI tree, and that we do not have the policy controls to filter the top-down (backwards) approach, and furthermore, we support multiple roots of trust so in general would not know where to start, then the top-down method is not appropriate.

There are two ways of performing bottom-up (forwards) validation, either height first in which the immediately superior credential only is obtained, recursively until the root is reached, or breadth first in which all the credentials of the immediate superior are obtained, and then all the credentials of their issuers are obtained recursively until the root or roots are reached. The latter approach may seem counter-intuitive, and certainly is not sensible to perform in real time in a large scale system, however a variant of it may be necessary in certain cases, i.e. when DAGs are supported, or when a superior possesses multiple identical credentials issued by different authorities. Furthermore, given that in our federation model described in section 2 (point 7) we allow a user to simply authenticate to a gateway and for the system to determine what the user is authorised to do (the credential pull model), the first step of the credential validation process is to fetch all the credentials of the user. This is performed by the Credential Retriever in Figure 3. Thus if the CVS recursively calls itself, the breadth first approach would be the default credential retrieval method. Thus we have added a retrieval directive to the credential validation method, which is set to breadth first for the initial call to the CVS, and then to height first for subsequent recursive calls that the CVS makes to itself.

In order to efficiently solve the problem of finding credentials, we add a pointer in each issued credential that points to the location of the issuer’s credential(s) which are superior to this one in the delegation tree. This pointer is the AuthorityInformationAccess extension defined in [19]. Although this pointer is not essential in limited systems that have a way of locating all the credential repositories, in the general case it is needed.

In order to ensure that a delegator does not overstep his or her authority, after retrieving the attribute(s) from the delegator’s credential we need to check that one of them is superior or equal to all the attributes in the delegate’s credential in the attribute hierarchy. If it is not superior to all of the delegate’s attributes in the attribute hierarchy, the delegator has exceeded his authority and the delegate's credential is discarded and processing stops.

In the case of relatively long lived credentials, revocation is clearly an issue. When a credential has been revoked, then all the credentials in the branch of the tree for which the revoked credential is the root, are also considered to be revoked. The CVS retrieves the revocation information about credentials when determining their validity. If any credential between the requestor’s credential and the root of trust has been revoked, then the requestor’s credential is considered to be invalid, and processing stops. We have also implemented a novel scheme for revoking credentials which uses the web as a finite state machine to indicate the revocation status of each credential [24]. This scheme inherently supports instant revocation and can be more efficient than using CRLs.

Finally, as a means of enhanced performance, we envisage that a background task could be run when the system is idle, that works its way down all the delegation trees from the roots of trust, in a breadth first search for credentials, validates them against the CVS’s policy, and caches the valid attributes for each user for a configuration period of time that is approximately equal to the period of CRL issuance. Then when a user attempts to access a resource, the CVS will be able to give much faster responses because the high level branches of the delegation tree will have already been validated.

6.3 Performance Measurements

The performance measurements were primarily conducted on a Linux machine (Intel Pentium(R) D 2.8GHz and 1GB memory) with Globus Tool kit 4.0.0 and MySQL installed. The application provided distributed access to a grid enabled MySQL database. Users with a particular role were granted access to the database. Since Globus Toolkit already supports java call outs to PIPs and PDPs, it was a relatively straightforward task to implement the PEP direct-call-to CVS model (Figure 5), with the CVS acting as a GT4 PIP.

Initially we configured the CVS to operate in attribute pull mode, retrieving X.509 attribute certificates from an LDAP server in which all the users' credentials were stored in their LDAP entries. The LDAP server ran on a different PC to the database service. They were connected via a high speed LAN. We tested attribute certificate chains from lengths 1 to 5, with 1 representing a credential issued directly by the PMI root of trust. The complete chain was PMI_root→AA1→AA2→AA3→AA4→AA5. No revocation lists were issued or processed.

Each set of performance measurements was carried out 100 times and the average and standard deviations were computed. Each set of 100 results contained several spurious results (between 1 and 6). According to Shewhart [28], when a process is in control, approximately 1% of the measurements will be greater or less than three times the standard deviation, and approximately 5% will be outside two times the standard deviation. Usually it was the very first one or two results in the set and then several random other ones. We believe that these spurious results are due to either java initialization or java garbage collection kicking in at random intervals. We removed these from the figures presented in the tables below.

Table 1. CVS pulling all credentials from LDAP Server (ms)

Delegation Chain length	Average time with signature verification(ms)	STD DEV	Average time without signature verification (ms)	STD DEV
1	9.44	2.30	5.44	1.75
2	17.08	3.68	9.49	2.62
3	24.77	4.81	13.31	3.28
4	31.96	5.60	16.80	3.82
5	39.92	6.69	20.25	4.66

The second column shows the average time for verifying a chain of attribute certificates. These figures are a combination of several tasks, repeated for each AC in the chain, namely: retrieve the AC from LDAP, verify its digital signature, and validate its asserted attribute(s) against the CVS's delegation policy. The figures show that as the AC chain length increases, the time taken to validate the subject's attribute increases by approximately 7.6ms for each additional AC in the chain between the subject's AC and the PMI root of trust. If we subtract this figure from all the results, for each AC in the chain, we find that the fixed CVS overhead time for marshalling arguments and producing a response is somewhere between 1.48 and 1.91ms.

In order to determine the time taken to pull an AC from LDAP, we configured the CVS to work in attribute push mode, providing the complete set of credentials to the CVS with each request. The results are shown in Table 2 column 2. To determine the time taken for signature verification we switched off the cryptography function for both the push and pull modes. These results are presented in the 4th columns of Tables 1 and 2.

Table 2. All credentials pushed to CVS (ms)

Delegation Chain length	Average time with signature verification (ms)	STD DEV	Average time without signature verification (ms)	STD DEV
1	6.16	1.55	3.91	1.45
2	10.67	2.46	6.47	1.96
3	15.74	3.34	8.92	2.64
4	19.78	3.59	11.10	3.27
5	24.70	4.24	13.91	3.66

An analysis of columns 2 and 4 in each table reveals that the time taken to verify a signature on an attribute certificate pulled from LDAP is approximately 3.86±0.1ms, whereas the same task when the attribute certificate is pushed to the CVS is just 2.2±0.1ms. The reason for this apparent disparity is that the public key certificate of the attribute certificate signer also has to be pulled from LDAP in the pull model, but since this was pushed to the CVS with the attribute certificates in the push model, there was no additional retrieval time in Table 2. We can therefore calculate that it takes approximately 1.66ms to retrieve a public key certificate from LDAP. When we compare the 4th columns of both tables we find that the average time to retrieve an attribute certificate from LDAP is 1.44ms (values range from 1.27 to 1.53ms) so this figure is close to that computed for retrieving a public key certificate. When we compare the 2nd columns of both tables we find that the additional time required to pull from LDAP when signatures are verified is 3.12ms on average (values range from 3.0 to 3.28). This is approximately the sum of the times we have just computed for retrieving an attribute certificate and public key certificate from LDAP. We don't believe there is any inherent difference in the time taken to retrieve an attribute or public key certificate, and the difference in the figures is within the error of measurement. To conclude, we calculate that it took approximately 1.55 ms to retrieve a certificate from LDAP and 2.2ms to verify the signature on an attribute certificate. We can see from table 2 that is takes approximately 3.91ms to parse the attribute certificate and verify it against the CVS's policy. When we add onto this the time for signature verification (2.2ms) and two retrievals from

LDAP (3.12ms) we get a total time of 9.23ms which is well within the standard deviation of the overall time for retrieving and validating an attribute certificate pulled from LDAP (9.44ms).

7 Conclusions and Future Work

Providing XACML with support for dynamic delegation of authority that is enacted via the issuing of credentials from one user to another, is a non-trivial task to model and engineer. In this paper we have presented the problems and requirements that such a model demands, and have architected a solution based on the XACML conceptual and data flow models. We have also presented at a conceptual level the policy elements that are necessary to support this model of dynamic delegation of authority. Given that these policy elements are significantly different to those of the existing XACMLv2 policy, and that the functionality required to evaluate this policy is significantly different to that of the existing XACML PDP, we have proposed a new conceptual entity called the Credential Validation Service, to work alongside the PDP in the authorisation decision making. The advantages of this approach are several. Firstly the XACML policy and PDP do not need to change, and support for dynamic delegation of authority can be phased in gradually. The exact syntax and semantics of the new policy elements can be standardised with time, based on implementation experience and user requirements. We have presented our first attempt at defining and implementing such a policy, and now have an efficient implementation that supports dynamic delegation of authority. A live demonstration is available at https://sec.cs.kent.ac.uk/dis.html.

Future work will look at supporting more sophisticated delegation trees and schema, and enforcing (or ignoring) Issuing Policies in target domains by passing the full policy embedded in the issued credentials. We also plan to support the delegation of role-permission assignments according to the design presented in [26] and incorporate additional policy elements in the delegation trees, such as attribute mappings of the kind described in [17].

Acknowledgments. We would like to thank the UK JISC for supporting this work under the research project entitled "Dynamic Virtual Organisations in e-Science Education (DyVOSE)".

References

1. See http://dictionary.reference.com/search?q=delegate
2. OASIS. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", 15 January 2005
3. ISO 9594-8/ITU-T Rec. X.509 (2001) The Directory: Public-key and attribute certificate frameworks
4. Scott Cantor. "Shibboleth Architecture, Protocols and Profiles, Working Draft 02, 22 September 2004, see http://shibboleth.internet2.edu/
5. ANSI. "Information technology - Role Based Access Control". ANSI INCITS 359-2004
6. Internet2 Middleware Architecture Committee for Education, Directory Working Group (MACE-Dir) "EduPerson Object Class Specification (200604)", 14 April 2006. Available from http://www.nmi-edit.org/eduPerson/internet2-mace-dir-eduperson-200604.html

7. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. “SPKI Certificate Theory”. RFC 2693, Sept 1999.

8. “OASIS eXtensible Access Control Markup Language (XACML)” v2.0, 6 Dec 2004, available from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

9. 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.

10. Paul Madsen. “WS-Trust: Interoperable Security for Web Services”. June 2003. Available from http://webservices.xml.com/pub/a/ws/2003/06/24/ws-trust.html

11. Markus Lorch , Seth Proctor , Rebekah Lepro , Dennis Kafura , Sumit Shah. “First experiences using XACML for access control in distributed systems”. Proceedings of the 2003 ACM workshop on XML security, October 31-31, 2003, Fairfax, Virginia

12. Wolfgang Hommel. “Using XACML for Privacy Control in SAML-based Identity Federations”. In 9th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2005), Springer, Salzburg, Austria, September 2005

13. Alfieri, R., Cecchini, R., Ciaschini, V., Dell'Agnello, L., Frohner, A., Lorentey, K., Spataro, F., “From gridmap-file to VOMS: managing authorization in a Grid environment”. Future Generation Computer Systems. Vol. 21, no. 4, pp. 549-558. Apr. 2005

14. Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Kate Keahey. “Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy”. Presented at NIST PKI Workshop, April 2006. Available from http://middleware.internet2.edu/pki06/proceedings/welch-idfederation.pdf

15. Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, Ronald L. Rivest. “Certificate chain discovery in SPKI/SDSI”. Journal of Computer Security, Issue: Volume 9, Number 4 / 2001, Pages: 285 - 322

16. Y. Elley, A. Anderson, S. Hanna, S. Mullan, R. Perlman and S. Proctor, “Building certificate paths: Forward vs. reverse”. *Proceedings of the 2001 Network and Distributed System Security Symposium (NDSS’01)*, Internet Society, February 2001, pp. 153–160.

17. Ninghui Li, William H. Winsborough, John C. Mitchell. “Distributed credential chain discovery in trust management”.Journal of Computer Security 11 (2003) pp 35–86

18. XACML v3.0 administration policy Working Draft 16 February 2007. http://www.oasis-open.org/committees/documents.php?wg abbrev=xacml.
19. Housley, R., Ford, W., Polk, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280, April 2002
20. David Chadwick. “Authorisation using Attributes from Multiple Authorities” in Proceedings of WET-ICE 2006, June 2006, Manchester, UK
21. OASIS. “Core and hierarchical role based access control (RBAC) profile of XACML v2.0”. February 2005
22. David W Chadwick, Linying Su, Romain Laborde. “Use of XACML Request Context to Obtain an Authorisation Decision”, Open Grid Forum Working Draft, 31 October 2007. Available from https://forge.gridforum.org/sf/go/doc14907?nav=1
23. David W Chadwick, Linying Su. “Use of WS-TRUST and SAML to access a CVS”. Open Grid Forum Working Draft, 31 October 2007. Available from https://forge.gridforum.org/sf/go/doc14908?nav=1
24. D.W.Chadwick, S. Anthony. “Using WebDAV for Improved Certificate Revocation and Publication”. In LCNS 4582, “Public Key Infrastructure. Proc of 4th European PKI Workshop, June, 2007, Palma de Mallorca, Spain. pp 265-279
25. G López, Ó Cánovas, AF Gómez-Skarmeta. “Use of XACML policies for a Network Access Control Service”. Applied Public Key Infrastructure. J. Zhou et al. Eds. IOS Press, 2005. Proc 4th International Workshop for Applied PKI, IWAP 05. Singapore. September 2005.
26. Tuan-Anh Nguyen, David Chadwick, Bassem Nasser. “Recognition of Authority in Virtual Organisations”. LCNS 4657, Trust, Privacy & Security in Digital Business eds Costas Lambrinoudakis, Gunther Pernul, A Min Tjoa. Sept. 2007, pp 3-13
27. Globus Toolkit Homepage is http://globus.org/toolkit/
28. Details of the Shewhart Control Chart can be found at http://www.itl.nist.gov/div898/handbook/mpc/section2/mpc221.htm

Appendix 1: CVS Policy Schema

```
<?xml version="1.0" >
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:permis="http://sec.cs.kent.ac.uk/permis" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="CVSPolicy" type="permis:CVSPolicyType"/>
   <xs:complexType name="CVSPolicyType" >
            <xs:sequence>
                        <xs:element name="TrustedIssuers" type="permis:TrustedIssuersType" />
                        <xs:element name="AttributeHierarchies"
type="permis:AttributeHierarchiesType" />
                         <xs:element name="Domains" type="permis:DomainsType"/>
                        <xs:element name="AttributeAssignments"
type="permis:AttributeAssignmentsType" />
            </xs:sequence>
           <xs:attribute name="CVSPolicyID" use="required" type="xs:anyURI"/>
   </xs:complexType>
<!-- -->
<xs:complexType name="TrustedIssuersType">
           <xs:sequence>
           <xs:element name="TrustedIssuer" maxOccurs="unbounded"
type="permis:TrustedIssuerType"/>
           </xs:sequence>
</xs:complexType>
<!-- -->
<xs:complexType name="TrustedIssuerType">
           <xs:attribute name="TrustedIssuer" use="required" type="xs:anyURI"/>
  <!-- Only LDAP and HTTP URLs are currently allowed for issuers -->
           <xs:attribute name="TID" use="required" type="xs:ID"/
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeHierachiesType">
           <xs:sequence>
```

```
        <xs:element name="AttributeHierarchy" maxOccurs="unbounded"
type="permis:AttributeHierarchyType"  />
        </xs:sequence>
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeHierachyType">
        <xs:sequence>
        <xs:element name="Superior" type="permis:SuperiorValueType"
maxOccurs="unbounded" >
        <xs:sequence>
        <xs:attribute name="AttributeOID" use="required" type="xs:anyURI"/
        <!-- Must be encoded according to SAML LDAP Profile e.g. urn:oid:1.2.3.4 -->
        <xs:attribute name="FriendlyName" use="required" type="xs:ID"/
</xs:complexType>
<!-- -->
 <xs:complexType name="SuperiorValueType">
        <xs:sequence>
        <xs:element name="Subordinate" type="permis:SubordinateValueType"
minOccurs="0" >
        <xs:sequence>
        <xs:attribute name="Value" use="required" type="xs:ID" / >
 </xs:complexType>
<!-- -->
<xs:complexType name="SubordinateValueType">
        <xs:attribute name="Value" use="required" type="xs:IDREF"/
</xs:complexType>
<!-- -->
<xs:complexType name="DomainsType">
        <xs:sequence>
     <xs:element name="Domain" maxOccurs="unbounded" type="permis:DomainType" />
        </xs:sequence>
</xs:complexType>
<!-- -->
  <xs:complexType name="DomainType">
        <xs:sequence>
```

```
            <xs:element name="RootNode" type="permis:RootNodeType"
maxOccurs="unbounded"
            </xs:sequence>
            <xs:attribute name="DomainID" use="required" type="xs:ID"/ </xs:complexType>
<!-- -->
 <xs:complexType name="RootNodeType">
            <xs:sequence>
            <!-- the excluded nodes must be immediately subordinate to the root node.
            Only LDAP and HTTP URLs are currently allowed for nodes -->
            <xs:element name="ExcludedNode" type=" xs:anyURI " minOccurs="0"
maxOccurs="unbounded"
            </xs:sequence>
            <xs:attribute name="Name" type="xs:anyURI" use="required"/>
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeAssignmentsType">
            <xs:sequence>
            <xs:element name="AttributeAssignment" maxOccurs="unbounded"
type="permis:AttributeAssignmentType"/>
            </xs:sequence>
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeAssignmentType" >
             <xs:sequence>
            <xs:element name="Attribute" type="permis:AttributeType" minOccurs="0"
maxOccurs="unbounded" />
             </xs:sequence>
            <xs:attribute name="AAID" use="required" type="xs:ID"/>
            <xs:attribute name="TI" use="required" type="xs:IDREF"/
            <xs:attribute name="DomainID" use="required" type="xs:IDREF"/>
            <xs:attribute name="DelegationDepth" use="optional"
type="xs:nonNegativeInteger"/>
   </xs:complexType>
<!-- -->
<xs:complexType name="AttributeType">
```

```
        <xs:sequence>
        <xs:element name="AttributeValue" type="permis:SubordinateValueType"
minOccurs="0" >
        <xs:sequence>
        <xs:attribute name="FriendlyName" use="optional" type="xs:IDREF"/
</xs:complexType>
<!-- -->
</xs:schema>
```