# Coordinating Access Control in Grid Services

David W Chadwick, Linying Su, Romain Laborde[1]
Computing Laboratory, University of Kent, Canterbury CT2 7NF, UK
[1]Institut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, 118 Route de Narbonne, F-31062, TOULOUSE CEDEX 9, France

**ABSTRACT**
We describe how to control the cumulative use of distributed grid resources by using coordination aware policy decision points (coordinated PDPs) and an SQL database to hold "coordination" data. When access to a resource is granted, obligations in the security policy ensure that the coordination database is updated. The coordination database is a normal grid service, thereby providing distributed access to the coordinated PDPs. Access to the databases is secured by the Grid Security Infrastructure (GSI) and its own PDP, so that only authorized users (the coordinated PDPs) can access it. A coordinated PDP is imbedded into the Globus Toolkitv4 authorization chain as a custom PDP so that any grid service can be protected by a security policy that provides a coordination capability. Each coordinated PDP uses the services of an uncoordinated PDP to make its access control decisions, so that any existing stateless PDP can be supplemented with a coordination capability. We provide performance results for the coordinated PDPs and compare these with two stateless PDPs. Virtually the entire performance penalty of using coordinated PDPs is accounted for by the heavy costs of using GSI to secure the communications between the coordinated PDPs and the coordination database.

**Keywords: PDP, coordinated access control, grid security**

## 1. INTRODUCTION

Automated Teller Machines (ATMs) have the capability to coordinate the withdrawal of money on a daily basis from any cash point in the world. This is achieved by standardization of the protocols within and between the banks, direct access to the user's account and the transactions that he has made, and the ability to write information to the security token (the bank card) that the user carries around with him. Providing a similar capability for grid jobs, for example, to limit the amount of storage or cpu that a user may request per day or per job from any location on the grid, is not so easy. The grid job will almost certainly run on different machines under different administrative control, will probably run under different account names on each machine, and the access control mechanism of one machine is typically unable to communicate with those of the other machines. The security token that is often passed from machine to machine is the proxy certificate [1] but this is not used by the policy decision points (PDPs) to communicate with each other, and is not under their direct control (unlike the bank card inserted into an ATM). Consequently the design of a policy based coordinated access control system presents a number of challenges.

The lack of communication between the PDPs of distributed applications can be addressed today by sidestepping the issue and using a centralised PDP with a common policy that is used by all the grid resources (see Figure 1). Such a system has been available for several years to Grid applications that use Globus Toolkit (GT) from v3.3 onwards. GT is capable of making an external authorization callout using the GGF SAML Authorisation protocol specification [3] and several PDPs such as PERMIS [2] and PRIMA [9] have implemented this protocol. This sort of access control infrastructure allows a common policy to be used by all the resources of a grid but since most PDPs today are stateless, they are still unable to coordinate their access control decisions across multiple access requests. A further disadvantage of this configuration is that the central PDP is a bottleneck to performance because every request needs to be diverted to it.

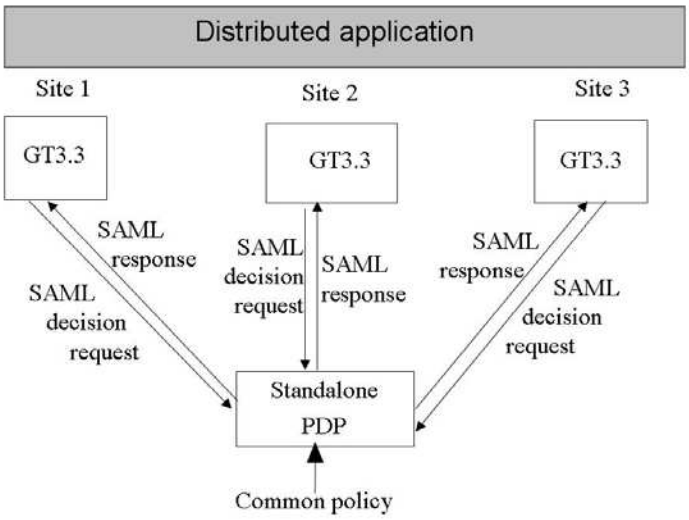


**Figure 1. Use of a common policy in today's distributed grid applications**

From performance and trust perspectives, it is preferable for each site to have its own PDP under its direct control and for the common policy to be distributed to the administrators so that they can review it and load it into their PDPs. But this approach still lacks coordination throughout the distributed application. Furthermore, the common policy will contain superfluous information for each of the PDPs, since it must cater for access requests to all of the resources in the grid.

To counteract these problems we have taken a multi-pronged approach. Firstly, in order to remove the superfluous information from each PDP's policy, we have designed and built a policy refinement engine that will decompose a common grid-wide policy and create resource specific policies for each resource PDP. We do not propose to describe the policy refinement process in this paper, but interested readers can consult [4]. Secondly, in order to address the coordination problem, the common policy (and consequently each refined resource specific policy) contains conditions and obligations that allow coordination to take place between multiple access requests. Conditions are placed on granting access to a user's request that depend upon previous access control decisions e.g. a project member is allowed access to memory only if less than 20GB memory has already been requested [for this job / project / day etc]. Obligations are then placed on the policy enforcement point (PEP) to record the resources that have been authorized for consumption. Coordination between the PDPs takes place by successively retrieving and updating this state information that is stored in an external coordination database. We chose to implement the system by using an external coordination database rather than storing the state information in each PDP for several reasons. Firstly most of today's PDPs, including ones with standard policies such as XACML [6], are stateless, so they will not need to be modified in order to support our secure coordination service. Secondly, if the state information was stored in each PDP, then each PDP would need to communicate with every other PDP, which would become increasingly difficult to engineer and manage as the number of PDPs grows. The periodic exchange of this state information has been suggested by another researcher [10] but we will comment more on this in Section 6 (Related Research). Thirdly, by using database technology to store the state information we benefit from its data persistency properties and the extensive research into fast and efficient data handling, searching, locking, distribution etc. Finally, by making the coordination database a grid service, we get the benefits of distributed access and secure communications between the PEP/PDP and the database service by using the existing Grid Security Infrastructure (GSI) [5].

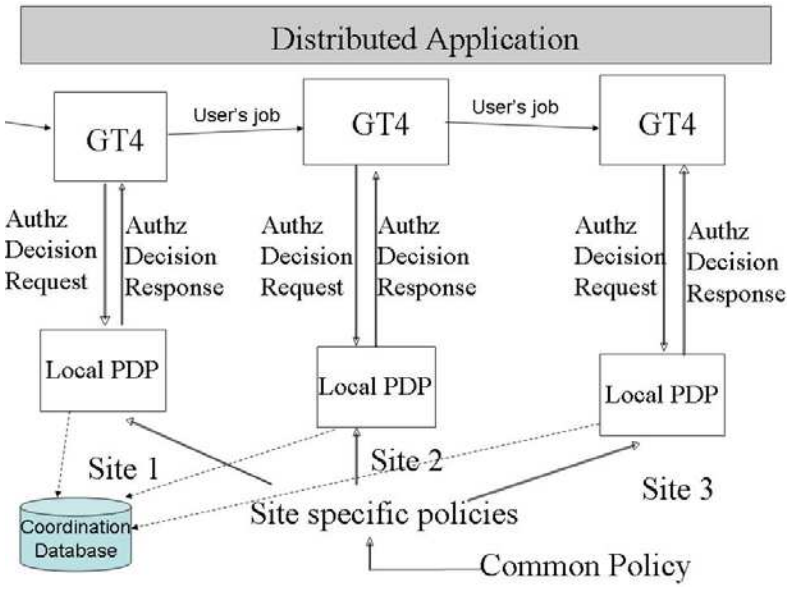


**Figure 2. Coordinated decision making in distributed applications**

A simplified picture of our overall design is shown in Figure 2. Each PDP loads its own resource specific policy, which may have optionally been created by refinement of a common grid wide policy. When coordinated access control decisions need to be made, each PDP communicates indirectly with all the other PDPs in the grid via the coordination database, to ensure that all the policy conditions are obeyed. This architecture is designed to have optimal performance for the decision making, since local uncoordinated decisions wont need to interact with the coordination database, whilst only coordinated ones will need to.

The rest of this paper is structured as follows. Section 2 describes the coordination policy, with its coordination attributes and obligations. Section 3 describes the coordination database grid service which holds the coordination data. Section 4 describes the coordinated PDP and how this is currently integrated into GT4. Section 5 presents the results of our performance measurements of coordinated and uncoordinated decision making and comments upon them. Section 6 compares our work with that of other researchers. Finally Section 7 concludes by looking at the limitations of the current implementation, and indicating where future development is still needed.

## 2. COORDINATION POLICIES

In our model, a PDP is considered to be stateless and makes its access control decisions against the current policy in isolation to all past, present and future access control decisions. This is in direct contradiction to the ISO Access Control Framework [11], which has the concept of *retained access control decision information (ADI)* that is stored internally in the PDP and is used to ensure that current decisions can be made according to the results of previous decisions. Obviously retained ADI is often needed, as many real world policies depend upon it, such as the ATM example given earlier. However, retained ADI stored internally in one PDP cannot be used by another PDP, without it being shared between them. The ISO model does not consider a distributed set of PDPs that need to share retained ADI, but we have to in a grid context. Consequently in our model we store the retained ADI externally to the PDPs, and pass it to the PDPs along with the other access control decision information when they need to make an authorization decision, so that the correct distributed and coordinated access control decisions can be made. We call the retained ADI *coordination data* in this paper.

A PDP is given access control decision information in the form of attributes about the subject, the resource, the requested action and the environment. This set of attributes is known as the request

context in the XACML specification [6]. The policy may place constraints on any of the request context attributes i.e. on the subject (e.g. only subjects who are students), the resource (e.g. only resources of type printer), the action (e.g. only print a maximum of 10 pages), and the environmental (e.g. only between 9am and 5pm) attributes. If any access control decision will affect future access control decision, then coordination between the access control decisions is required, which is why coordination data is needed. For example, if the policy states that students can only print up to 10 pages per day, then if a student asks to print 5 pages this will be allowed but the granted action must be remembered so that the next time the same student makes a request on the same day he will only be allowed to print up to 5 pages. This type of policy constraint on subject, action and/or environmental attributes will always require coordination between access control decisions, regardless of whether the system has a single centralised PDP or multiple distributed PDPs. The use of a policy to specify the constraints, and a coordination database to hold the coordination data, makes the coordination process independent of the number of PDPs involved in the grid access control decision making.

### 2.1 Coordination Attributes

We model the coordination data as a fifth type of request context attribute, which we term the coordination attributes. Coordination attributes are conceptually the same as any other type of request context attribute (resource, subject, action or environment) but in this case they are attributes of the coordination object, rather than the resource, subject, action or environment objects. The coordination object is conceptually a repository storing the coordination data that is necessary to allow coordination to take place between all of the access control decisions in a distributed system. The semantics of the coordination attributes are known to the coordination object but not to the PDPs, since the latter do not know the semantics of any of the attributes of the request context (environment, subject etc.). The PDPs only know that the request context attributes hold values that need to be compared with those in the access control policy to see if the access control conditions are fulfilled.

The coordination object is considered to be persistent and stateful, in much the same way that the environment object stores the environmental attributes in a persistent way. In this way the PDPs remain stateless. A significant difference between the environmental and coordination attributes is that the access control process only needs to read the former, whereas it needs to read and update the latter. Furthermore, a coordination attribute is related to attributes of a subject, resource, action, or the environment, and can be indexed on any combination of them, because policy constraints can be placed on any combination of them.

We specify a coordination attribute as follows:

Att[SubDim, ResDim, ActDim, EnvDim](C)

where Att is the name of a coordination attribute belonging to the coordination object C, and [SubDim, ResDim, ActDim, EnvDim] are optional multiple dimensions of the coordination attribute. SubDim, ResDim, ActDim and EnvDim denote the subject, resource, action and environment dimensions of the coordination attribute, respectively. Every attribute in SubDim (ResDim, ActDim or EnvDim), if any, come from the request context.

Examples of coordination attributes and related policies are:

- usage(C) means the coordination attribute called *usage* has a single value that is used by all subjects accessing all resources over all actions and environments, for example, when a policy states that the sum total of all resources used by everyone in any way can only be up to a certain amount and then all access must cease.
- usage[username(S), type(R)](C) means the coordination attribute called *usage* has a different value per subject per resource, where each subject is identified by their username attribute

and each resource by its type, for example, when a policy states that each user can use each type of resource up to a certain predefined amount.

Consider the following policy with a coordination constraint: *users, identified by their userIDs, cannot use more than 3GB of storage each throughout the grid*. This could be expressed mathematically as type(R)=storage∧type(A)=use∧amount(A) + alreadyUsed[userID(S)](C) ≤ 3.

Most stateless PDPs should now be capable of evaluating this type of policy, providing the request context contains the value of the alreadyUsed[userID(S)] coordination attribute. The fact that the coordination attribute contains embedded encoding in the form of [userID(S)] should be transparent to the PDP, since the names of attributes have no semantic meaning to the PDP. They are simply strings which need to be compared when matching attribute names. Providing the name of the coordination attribute is identical in the policy and in the request context (including the embedded encodings) the PDP should be able to compare the attribute names by matching the strings. Once the names are seen to be identical, all the PDP needs to do is to compare the value of the coordination attribute presented in the request context with the corresponding value in a policy constraint according to the mathematical function in the constraint (<, >, ≤ etc.).

### 2.2 Obligations

The next thing we need to ensure is that the coordination attributes are updated once the user has been granted access to the resource. This is achieved by the use of appropriate obligations in the coordination policy. Obligations are actions placed on the PEP that have to be obeyed if/when the user is given or denied the requested access to a resource. In the XACML standard [6] an obligation is defined as a set of attribute assignments, for example, assign balance[id(S)](C) + amount(A) to balance[id(S)](C). Since it is the PEP that enforces the grant or deny decisions of the PDP, and is responsible for enforcing all obligations, it seems appropriate that the PEP should also be the entity that updates the coordination object. We have extended the XACML concept of obligations by adding an optional directive to them which tells the PEP when to carry out the obligated actions. We have called this the Chronicle directive and it can take one of three values: Before, After or With. Chronicle=After indicates that the obligation should be enacted only after the user's access request has been enforced. Chronicle=Before indicates that the obligation should take place before the user's request is enforced. Chronicle=With indicates that the obligation and the user's request should be enforced as an atomic action. It is up to the coordination policy writer to determine which Chronicle value to use. Note that XACML does not have a Chronicle parameter, since it implicitly assumes the semantics of Chronicle=With. We believe that this is insufficient in practice, but there are important implications in the use of the alternative Chronicle values.

Chronicle=Before means that the coordination attribute will be updated before the user's request is processed. Therefore if the user's request subsequently fails for some unexpected reason e.g. the ATM machine jams and cannot dispense any money, the user may be prevented from performing the same action again at a later time. This is because the coordination attribute has already recorded the obligation prior to the action taking place. Similarly Chronicle=After means that the user is allowed to perform his action before the coordination attributes are updated. If anything goes wrong with the subsequent coordination attribute update, the obligation will not be recorded and the user may be allowed to perform the same request again, in contravention of the policy. However Chronicle=After might be the option that some policy writers prefer e.g. banks might prefer ATM withdrawals to occasionally allow a customer to withdraw over his daily limit than to risk upsetting him by occasionally not allowing him to withdraw his daily limit. Chronicle=With does not suffer from either of the previous deficiencies, since the user's request and the coordination attribute update are performed as an atomic action. However this means that transactions have to be enacted on the coordination database, a two phase commit protocol with the resource and the coordination database will be needed by the PEP, and the coordination attributes will have to be write locked for the

duration of the user's action. This may cause an unacceptable overhead and bottleneck to performance in many grid applications that can run for hours or even days. Therefore the policy writer has to choose the most appropriate Chronicle setting for his resources and the applications that use them.

Further details about the coordination policy specification and its refinement can be found in [7].

## 3. THE COORDINATION DATABASE GRID SERVICE

The coordination database is a grid service with a backend SQL database that provides access to the coordination attributes needed by the multiple PDPs. The service supports seven methods, namely:

- checkWS checks if the service is available or not, and returns true or false
- getCoordAttrVal returns the value of a coordination attribute given its name and a request context (that contains the values of the dimensional attributes that are embedded in the coordination attribute name). If no value currently exists for this element, then the service creates a new one and initializes it with the initial value known to the coordination object (note that the initial value is part of the meta-data or schema for the coordination attribute).
- setCoordAttrVal sets the value of a coordination attribute. The service returns an exception if it fails, but void if it succeeds
- isCoordAttr queries if a coordination attribute of this name exists in the coordination database, and returns the value true if it does, or false if it does not
- getAttributeDefinition returns an XML element which contains the definition of the coordination attribute including the attributes that are embedded in its name, or an exception if the attribute does not exist
- lockCoordAttrs allows one or more attributes in the database to be read or write locked, and returns true if the lock succeeds, or false otherwise
- unlockCoordAttrs removes all the locks in the database held by the current thread, and returns true if it succeeds or false if it does not.

The structure of the backend SQL database is a series of tables, in which each table represents one coordination attribute. In every table there is one column for each subject, action, resource or environment attribute that is contained in the definition of that coordination attribute plus one column to hold the coordination attribute values. The general formula for the size of a table is:

$$(|SubDim|+|ResDim|+|ActDim|+|EnvDim|+1) \times N$$

where |XDim| represents the number of members in the set XDim and N represents the number of rows in the table. For example, the coordination attribute recording the number of user accesses in different modes to files in different filestores could be held as numberOfAccesses [id(S),id(R),mode(A),fileName(A)](C) and is represented as a table which consists of 5 columns. Every unique combination of user id, filestore id, mode of access and filename will create a new row in the table, with the final column recording the number of accesses for that user to that file in that access mode. Assume a subject (id = X) wants to access a file (mode = M, fileName = F) on a filestore (id = Y), then the current value of the numberOfAccesses may be located from this table by the following SQL command: SELECT value FROM numberOfAccesses WHERE id(S)=X AND id(R)=Y AND mode(A)=M AND fileName(A)=F. If no record can be located using these dimensional attribute values, this means that it is the first user access of this kind to this file, and a new row, consisting of these dimensional attribute values and the initial value for the

numberOfAccesses should be inserted into the table. The initial value is part of the semantics of the coordination object and is defined in the schema for the coordination database service.

### 3.1 Securing Access to the Coordination Database Grid Service

The coordination database service, being a standard grid service, is protected by GSI and its own stateless PDP that ensures that only authorised coordinated PDPs can access it. This is easily achieved by assigning digitally signed X.509 attribute certificates (ACs) containing a role of "Coordinator" to the coordinated PDPs (actually to the Coordinator component in Figure 3), and having a standard role based access control (RBAC) policy in the stateless PDP that says that only subjects with the role of "Coordinator" are allowed to access the coordination database. This allows any number of coordinated PDPs to access the coordination database without needing to modify the RBAC policy, providing each has been given the "Coordinator" role AC issued by the trusted issuer. Each Coordination PIP has its own public key certificate and DN so that it can strongly authenticate to the coordination database grid service via GSI. For authorisation, we use the attribute pull model, so that the "Coordinator" role ACs are pulled by the database service's Subject PIP (see below) from the Coordinator's LDAP entry in the VO's LDAP directory service. We could have used the attribute push model instead, and packaged the "Coordinator" role AC inside the proxy certificate, as in the VOMS model [12], but this would have meant extra overhead at the client side. We have used the PERMIS PDP to protect the coordination database, since this supports both the push and pull modes of attribute retrieval.

## 4. THE COORDINATED PDP

The GT4 authorisation framework implements a decision engine which is capable of evaluating a chain of PDPs in order to determine the access rights of the user making a request for a particular Grid service or resource (the Target in Figure 3). This authorisation chain may also include multiple Policy Information Points (PIPs), which do not return any decisions but instead are used to collect information i.e. attributes or attribute assertions, necessary for the decision-making process. Both PDPs and PIPs are classified by GT4 as interceptors. Globus Toolkit itself is the PEP that enforces the decisions made by the PDPs, and passes the information returned by the PIPs to the PDPs. PIPs are needed to pick up the attributes of the subject, the action, the resource and the environment. GT4 requires that PDPs implement the *org.globus.wsrf.security.authorization.PDP* interface and return a permit or deny decision on the basis of the subject's distinguished name (DN) (obtained from the proxy certificate), the requested operation and the request context. PIPs must implement the *org.globus.wsrf.security.authorization.PIP* interface, and must place the set of retrieved subject, action, resource and environmental attributes in the request context.

Ideally we would like GT4 to be modified in order to integrate our coordinated authorization infrastructure directly into the PEP. However, if we did this it would cause integration problems for other users and long term support issues for us. Consequently we have adopted an interim approach of plugging our entire coordination infrastructure into GT4 as a single custom PDP. This has limitations as described in section 7, but still allows secure coordination to take place. In section 7 we describe how the coordination infrastructure could be more tightly integrated into GT to become an integrally supported feature.

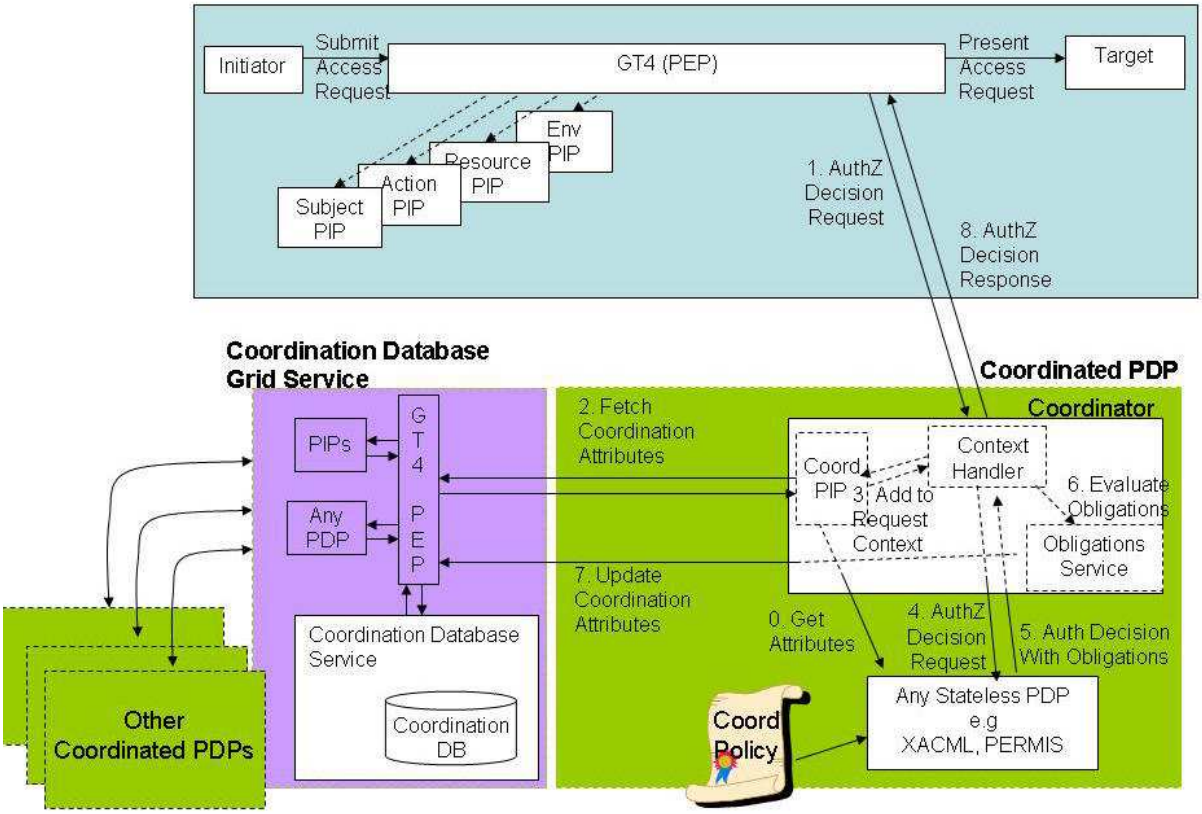


**Figure 3. Adding coordination to GT4**

The coordinated PDP comprises two components: a Coordinator object and a stateless PDP (such as an XACML or PERMIS PDP). The Coordinator object comprises three functional components:

- a context handler which is responsible for: receiving requests from and providing decisions to the GT4 PEP, making requests to and receiving decisions from the stateless PDP, and communicating with the other two internal objects,
- a Coordination PIP which is responsible for retrieving coordination attributes from the coordination database, and
- an Obligations Service which is responsible for evaluating obligations and updating the coordination database.

When GT4 is given a user's request to access a grid service, the coordinated PDP is called via the GT4 authorisation chain (step 1 in Figure 3). The Coordinator's context handler calls the Coordination PIP to write lock and retrieve the required coordination attributes from the coordination database grid service (step 2).

The obvious question to ask is, how does the Coordinator know which coordination attributes are needed, since the coordination database could contain many thousands of attributes. However, this question is not a new one, since even without coordination attributes the PEP still needs to know which other attributes (environmental, action etc) are needed by the PDP. There are (at least) three possible solutions to this problem. Firstly, the PEP can be configured in an application specific manner with the correct set of attributes that need to be passed to the PDP in each request context. Secondly, a getAttributes method can be added to the PDP which is called at initialisation time and returns the complete set of all attributes that are needed by the current authorisation policy. Thirdly, the PDP can dynamically return the set of additional attributes that are needed in order to answer the

current authorisation decision request. This third mechanism is the one adopted by XACML. Since the second mechanism is simply an automation of the first they will be treated as equivalent in the following discussion.

Having the complete list of attributes at initialisation time will mean that the stateless PDP only needs to be called once per authorisation decision, but may result in surplus information being passed in the request context. Not knowing the correct set of attributes to pass when the PDP is called will result in multiple calls to the PDP, and possibly to the coordination database, but no surplus information will be passed in the request context. The latter approach is expected to be most efficient when policies are large and different attributes are needed for different policy rules, and especially when coordination is not needed for the majority of the access control decisions. As we will see from the performance results in Section 6, accessing a coordination database via GSI poses the largest overhead in all the steps of coordinated decision making, and therefore should not be taken when a coordinated decision is not needed. The former approach is expected to be the most efficient for small policies that only require a limited number of attributes, and when coordination is required for the vast majority of the access control decisions.

We have currently implemented the second mechanism in our Coordinator and have added a getAttributes method to the XACML and PERMIS PDPs. This method is called during the initialisation phase of the coordinated PDP (Step 0 in Figure 3). A future development would be to implement the third mechanism as well in the Coordinator, and to repeatedly call the stateless PDP, but as described in Section 7, we think a better solution would be to implement this in GT4.

When the appropriate coordination attributes have been returned from the coordination database grid service, the Coordination PIP adds them to the request context obtained from the GT4 PEP (step 3), and passes this expanded context back to the context handler for it to use in its call to the stateless PDP (step 4). The coordination attributes are actually packaged as environmental ones in the request context, since the stateless PDP cannot tell the difference between them. The stateless PDP returns an authorisation decision, according to its evaluation of the user's request against the coordinated access control policy (step 5). If the stateless PDP returns access granted, and the decision is a coordinated one, then the authorisation decision will contain obligations to update the appropriate coordination attribute values. The context handler makes various calls to the Obligations Service (see below) which evaluates the obligations (step 6), updates the coordination database, and removes the locks (step 7). The context handler returns the granted response to the GT4 PEP (minus the obligations) (step 8). The user is then allowed access to the service by GT4. The reader will note that this design only supports the Chronicle=Before option for obligations. In section 7 we describe how the other Chronicle options can be implemented.

The obligations service has 3 methods:

- getChronicle. This method is given the authorisation decision response and returns the value of the Chronicle parameter (Before, With or After), or an exception if the response does not contain any obligations.
- evaluateObligation is passed the request context and authorisation response and evaluates the various attribute assignments in the obligation, including the arithmetic expressions such as addition, subtraction, multiplication etc. e.g. assign balance[id(S)](C) + amount(A) to balance[id(S)](C), and places the result in the response.
- performObligation extracts the coordination attribute values from the response, updates the coordination database by making repeated calls to setCoordAttrVal, and finally removes the locks from the database.

We have currently implemented a couple of simple services as proof of concept. One simulates requesting different amounts of storage from different resources on the grid, and the user is

constrained to a maximum allocation for his job. The other simulates multiple withdrawals from a series of ATM machines, and the user is constrained to withdrawing a maximum amount per day. We will describe the latter more fully and present the performance measurements for it with and without coordination between the withdrawals.

## 5. PERFORMANCE MEASUREMENTS

### 5.1 The ATM Service

When the user asks to withdraw money from the ATM service, GT4 calls the custom PIPs to retrieve the attributes of the user, the requested action, the resource and the environment. GT4 then calls the custom PDP to make an authorisation decision. Without a coordination capability, a stateless PDP is used (XACML or PERMIS), but it can only have a policy constraint that limits each withdrawal to a particular fixed value (say £250) each time (*amount(A) ≤ 250*). If a user requests £250 or below, the request will be granted by the stateless PDP and the ATM service will return a positive response to the user. If a user requests greater than £250 the stateless PDP will deny access to the service. Of course, a user can make multiple consecutive requests for £250 or less and therefore withdraw an unlimited amount of money. Once the coordinated PDP is configured into GT4, the ATM service can be coordinated and the policy can now limit withdrawals to a fixed amount per person per day. The Coordinator retrieves the value of the coordination attribute (*balance[subject-DN, date](C)*) from the coordination database and adds this to the request context. Note that the initial value (250) for the coordination attribute *balance* is defined in the schema for the coordination database. The Coordinator now calls the stateless PDP, which can be the same one as was used by the uncoordinated ATM service, only now it has a different policy in which the constraint refers to the coordination attribute value rather than a fixed value i.e. (*amount(A)≤balance[subject-DN, date](C)*), and also contains an obligation on permit to update the coordinated amount (*balance[subject-DN, date](C)←balance[subject-DN, date](C) – amount(A)*). If the request is granted, the stateless PDP returns a granted decision plus the obligation. The Coordinator updates the Coordination database and returns granted to the GT4 PEP. When the user makes a second ATM request, the updated value of *balance* is retrieved by the Coordinator. Once the user has been granted his daily allowance, all subsequent requests are denied. The system works regardless of how many different ATM services and PDPs are plugged into GT4, since all the PDPs coordinate their decisions via the same coordination database service. Initially, each *balance* array element has the value of 250 and it drops when an amount of money is withdrawn by a subject on a date. The subject DNs and dates are collected by the custom PIPs, so that when different subjects withdraw money on different days, the coordination attribute *balance[subject-id, date](C)* will refer to different values, e.g. *balance[“cn=mary,o=uok,c=gb”, “2007-01-26”]* and *balance[“cn=jack, o=uok,c=gb”, “2007-01-25”]*. Whatever the user cn=jack,o=uok,c=gb withdraws on 25th January 2007 will not affect his next day’s limit because *balance[“cn=jack,o=uok,c=gb”, “2007-01-26”]* refers to a different data value. Similarly whatever Mary withdraws on one day will not effect what Jack can withdraw on the same day.

### 5.2 Performance Measurements

The performance measurements were conducted on a set of six Linux machines – see Figure 4. The coordination database grid service and ATM application services 3 and 4 ran on Intel Pentium(R) D 2.8GHz machines with 1GB memory. ATM applications 1 and 5 ran on 2GHz machines with 0.5GB memory, with ATM service 2 running on a 1.4GHz machine. ATM service 5 was on the same LAN segment as the Coordination service which accounts for its faster performance than ATM service 1. All six machines had Globus Tool kit 4.0.0 installed. We performed all the performance tests using the ATM simulation service described above (or a variant of it). We performed three distinct sets of tests. The first set was designed to measure the performances of the individual functional components that make up the overall coordinated access control decision making, and used a single

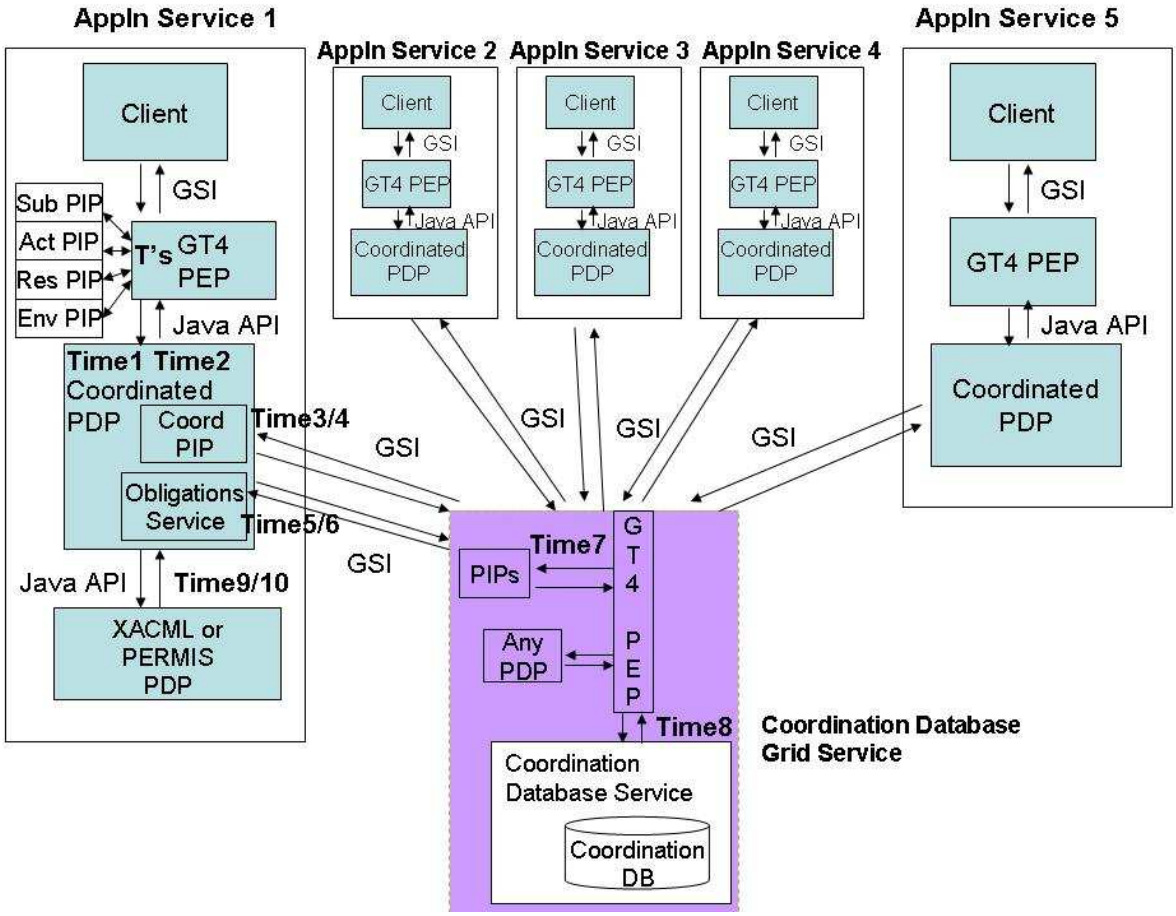


**Figure 4. Configuration for Performance Measurements showing various Time collection points**

ATM service and the coordination database service both running on 2.8 GHz machines. We tested this system using both the PERMIS PDP [28] and Sun's XACML PDP [29], in order to compare their performances as well as that of the coordinated PDP.

The second set of test was designed to test scalability and functionality, by using multiple distributed PDPs simultaneously accessing the same coordination attribute in the coordination database. For this, we deployed the ATM service on between one and five PCs, with each simultaneously withdrawing money from the same user account so that they could coordinate their decision making via a single coordination attribute. The functional results of this testing were as expected, that the user was only allowed to withdraw a total of £250, regardless of how much was attempted to be withdrawn from which service machine. The performance results showed a marked degradation as more services joined the network, since each user's request blocks all the other requests due to the coordinator locking the attribute in the coordination database until it has finished.

The third set of tests was designed to test concurrent accesses to the coordination database service from different applications. For this set of tests we modified the ATM service on each machine to specify a different coordination attribute, so that each coordinated PDP would lock different attributes (tables) in the coordination database and consequently would not block or hold each other up. The performance degradation in these tests was significantly lower than that of concurrent access to the same coordination attribute.

Note that in all the above tests, whether GT4 calls the coordinated PDP or the PERMIS PDP or the XACML PDP is specified in the service's security descriptor, so it is possible to alter the system architecture purely by configuring the security descriptor and changing the access control policy to suit.

### *5.2.1 Performance of Individual Components*

Each set of performance measurements was carried out 100 times. In each set of 100 results we found some spurious results which were significantly larger (~200ms) than the others, sometimes an order of magnitude or more. According to Shewhart [30], 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. We suspect that our spurious out of control measurements are due to Java garbage collection that kicks in at random intervals, although those that depended upon network communications could also be affected by random network delays. In order to counteract this, we set the upper control limit to 3 times the standard deviation, and removed all the spurious measurements that were above this limit. This necessitated us having to remove between 1% and 10% of the measurements when computing the results.

We measured the performance of each PIP, the stateless PDP (Time10-Time9), the obligations service, the various coordination database actions (lock, read, write, unlock), the time for GT4 to transfer messages to and from the Coordination Service (Time4-Time3-Time8+Time7) both with and without GSI, and the overall time for coordinated decision making (Time2-Time1) both with and without GSI. The overall time comprised the sum of the times consumed by the coordination PIP, the obligations service and the stateless PDP. We made sure that each request in a set of 100 was either grated or denied by only withdrawing £1 each time for grants, or >£250 for denies. The average times and standard deviations for coordinated decision making (Time2-Time1) are shown in Table 1.

**Table 1. Decision Making with Coordination (ms)**

| Use of GSI | Response | Average time (ms) | STD DEV |
|---|---|---|---|
| With GSI | DENY | 824 | 70 |
| Without GSI | DENY | 320 | 32 |
| With GSI | GRANT | 809 | 77 |
| Without GSI | GRANT | 345 | 34 |

Whilst we might expect coordinated Grants to take longer than coordinated Denies since they have to process obligations, the overheads of GSI and making network calls to the coordination database service appear to completely swamp any minor differences in processing time, so that we can conclude that both coordinated grants and denies take approximately the same amount of time to complete in this configuration.

By way of comparison with uncoordinated decision making, the time taken to make an uncoordinated authorization decision (time10-time9) using a stateless PDP was 1.5±0.26ms for the PERMIS PDP and 6.4±2.2ms for Sun's XACML PDP. This is approximately two orders of magnitude faster than coordinated decision making. We note that the PERMIS PDP always outperforms Sun's XACML PDP by a factor of approximately 4. There are several reasons for this. PERMIS policies are monotonic, being based on the paradigm that everything is denied except that which is allowed by the policy. Thus once a grant is discovered, policy processing can stop and the grant result can be returned. XACML policies on the other hand are not monotonic and allow conflicting grant and deny rules to be present. These are mediated by rule and policy combining rules. Consequently XACML always has to process every policy rule, along with the combining

rules, before it can return a result. Finally XACML policies are syntactically more complex (although more comprehensive) than PERMIS policies, as is the structure of their obligations, and this also contributes to the increased processing cost.

The overall time taken by the coordinated PDP consists of locking the coordination attribute in the database, reading its current value, merging this together with the attributes from the other PIPs, making an authorisation decision by calling the stateless PDP, getting the decision response, extracting and evaluating the obligations, updating the coordination attribute value in the coordination database, releasing the locks, and finally returning the result to the GT4 PEP. In order to reduce the delays introduced by network round trips, we combined the database lock/read commands into one network message, and the update/unlock commands into a second message thereby reducing four message exchanges to two. Table 2 present the performance results of the various components of the Coordination PDP when it returns grant responses using the PERMIS PDP. We present results with and without GSI securing the GT4 connection to the coordination database service.

**Table 2. Time Taken by Each Component of the Coordinator (ms)**

| | Coordination PIP (Time4-Time3) (lock/read) with GSI | Coordination PIP (Time4-Time3) (lock/read) without GSI | Merge attributes | Authz decision | Get response/ Evaluate obligations | Write-unlock (Time6-Time5) with GSI | Write-unlock (Time6-Time5) without GSI |
|---|---|---|---|---|---|---|---|
| AVERAGE | 382 | 169 | 0.89 | 1.50 | 1.0 | 385 | 170 |
| STD DEV | 17 | 17 | 0.8 | 0.26 | 0.12 | 18 | 18 |

The significant result is that lock/read the database and write/unlock the database take two orders of magnitude longer than the other coordination tasks. This is because all of these involve remote operations on the coordination database, using GT4 and optionally GSI to secure the connection. Furthermore, using GSI more than doubles the overhead of this connection. In order to determine how much of this time is spent in the coordination database server, and how much establishing the GT4 connection, we measured the time taken by each of the components in the coordination database service from the time the request was received by the Subject PIP (Time7) until the service returned its response to the GT4 PEP (Time8). The results are shown in Table 3.

**Table 3. Time Taken by each Component of the Coordination Database Service (ms)**

| | Subject PIP | Resource PIP | PERMIS PDP | Lock/Read | Write/Unlock | Overall (Time8-Time7) |
|---|---|---|---|---|---|---|
| AVERAGE | 11.0 | 3.7 | 1.0 | 1.50 | 1.21 | 20 |
| STD DEV | 1.8 | 1.0 | 0.3 | 0.26 | 0.17 | 2.8 |

It can be seen that the process of authorizing access to the coordination database service and then performing the requested actions takes approximately 20ms, of which over half of this time is spent in the subject PIP retrieving and validating the coordinator's role AC from an LDAP directory. We can deduce that the overhead of the network connection and GT4 transport of the messages between the coordinator and the database service, without GSI, is approximately 150ms per round trip, whilst GSI adds an additional 215ms to this. The coordination database service therefore accounts for only about 5% of the time taken by the Coordination PIP to read a coordination attribute.

Finally we measured the performances of the various PIPs. The task of the subject PIP is to return the subject's DN, which it obtains from the proxy certificate. It also contacts the VO's LDAP service

to pull a role attribute certificate (role=Customer) that has been assigned to the subject. The task of the action PIP is to return the requested action and its parameter, which are taken from the user's request. In these tests the action is withdraw and the parameter is the amount to be withdrawn e.g. withdraw amount=200. The task of the resource PIP is to return the service's DN, whilst that of the environment PIP is to return the current date. The performance results of the PIPs are shown in Table 4. Note that our results showed that the performance of the Action, Subject, Resource and Environment PIPs was independent of which PDP is called by GT4, or the decision that was made. You can also see that the performance of the subject PIP and resource PIP in the ATM service is approximately the same as those in the coordination database service (Table 3).

**Table 4. Custom PIP Performance Measurements (ms)**

| Action PIP | | Subject PIP | | Resource PIP | | Environment PIP | |
|---|---|---|---|---|---|---|---|
| Average | STD DEV | Average | STD DEV | Average | STD DEV | Average | STD DEV |
| 1.19 | 0.18 | 12.04 | 2.26 | 3.60 | 1.00 | 1.47 | 0.34 |

Although both the subject and resource PIPs are implemented using the same PERMIS Credential Validation Service (CVS), the different times taken by them is due to the different work that is being done. The Subject PIP is returning the subject's role attribute value (which is embedded in an X.509 attribute certificate (AC) stored in an LDAP directory) as well as the subject's DN. The Resource PIP is only returning the resource's DN because the resource does not have an X.509 AC. The environment PIP returns the current date and time taken from the system clock, but by replacing this with a PERMIS CVS it would be possible to retrieve a secure timestamp from a trusted time stamping authority and use this instead of the system clock. Clearly, the time taken by the action, subject, resource and environment PIPs will vary depending upon what attributes they are required to fetch and validate from where. For example, we have already shown that the Coordination PIP takes approximately 380ms (with GSI) and 170ms (without GSI) to fetch an attribute from a grid enabled database service. This compares very unfavourably with the 12ms it takes the Subject PIP to fetch an attribute (an X.509 AC) from an LDAP directory. An alternative future design for the coordination database service will be to replace the GT4 communications mechanism with a much lighter and faster alternative.

#### *5.2.2 Scalability Performance of Accessing the Same Coordination Attribute*

Each set of performance measurements was carried out 100 times. We first ran the tests using each ATM service on its own. This provided the base performance figures for each machine which is shown in row 1 of Table 5. The figure before ± is the average time and after ± is the standard deviation from the average. We then ran the tests again with either two, three, four or five ATM service machines running in parallel. When there were one or two machines, we found that between 3 and 5 results where larger than three times the standard deviation and these were removed from the figures shown below. When there were 3 or more machines in parallel, all the results fell within 3 times the standard deviation, so we set the upper control value to two times the standard deviation and found that approximately 10% of the measurements fell outside this range and needed to be removed before calculating the results presented below. We did this in order to get acceptable standard deviations.

**Table 5. Performance results (ms) for services accessing the same coordination attribute (Time2-Time1)**

| No of ATM Services | ATM Service1 | ATM Service2 | ATM Service3 | ATM Service4 | ATM Service5 |
|---|---|---|---|---|---|
| 1 | 809±85 | 785±90 | 730±74 | 709±71 | 684±58 |
| 2 | 867±128 | 843±104 | | | |
| 3 | 1065±108 | 1052±121 | 1102±107 | | |
| 4 | 1442±119 | 1414±111 | 1453±115 | 1456±119 | |
| 5 | 1790±149 | 1790±124 | 1812±130 | 1815±133 | 1819±124 |

From 2 services upwards, performance degraded almost linearly as the number of ATM machines increased, as shown in Figure 5. This is because each service was blocked for longer as more services were added, since they all compete to lock the same coordination attribute. By consulting Table 2, one would expect the time increase to approach 380ms per additional ATM service, as this is the time between a service taking a lock and releasing it. The reason why the second service did not appreciably degrade performance was that whilst the first service had the coordination attribute locked, and was performing the GSI communications overheads for the write/unlock request, the second service was performing the GSI communications overheads for the lock/read request. Both completed these at approximately the same time, and then the two services switched roles and the second service took hold of the lock. Thus performance degradation was small.

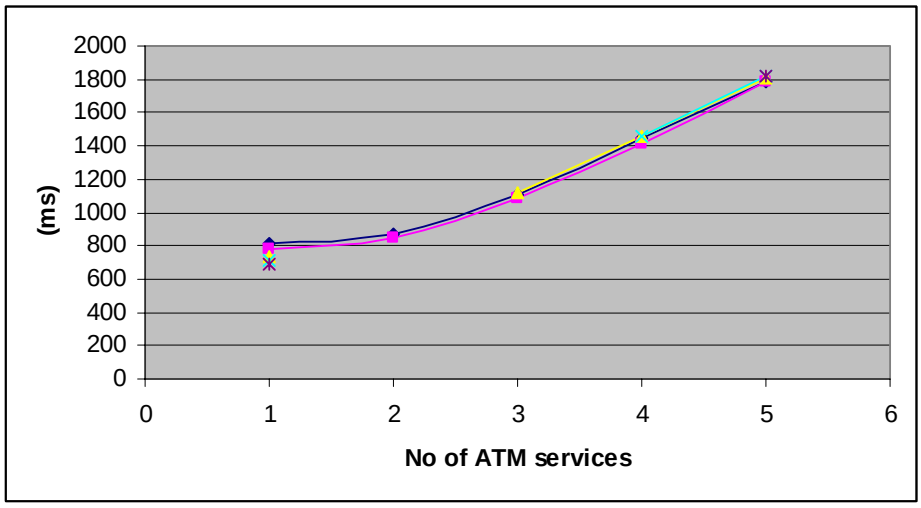


**Figure 5. Graphical display of Table 5.**

In order to ensure a fair distribution of the database lock when more than two services were competing for it, we had to introduce a FIFO queuing mechanism into the coordination service, so that as each new service request arrived, unless it already held the existing lock, it was placed at the tail of the FIFO queue, and then went to sleep for 1ms. Each time the service request awoke, if it was not at the top of the queue, it would go to sleep again for another 1ms. Eventually it would awake to find itself at the top of the queue, whereupon it would perform its database queries, and then remove itself from the queue when it was asked to execute the unlock request.

### *5.2.3 Scalability Performance of Accessing Different Coordination Attributes*

In this set of results each application service accessed a different coordination attribute, hence no process would be blocked waiting for another process to release locks on the coordination attribute. In this case the limiting factor will be the processing power of the coordination service PC, since it is

being driven in parallel by between 2 and 5 application machines. The performance figures are shown in Table 6 below and depicted in Figure 6.

**Table 6. Performance results (ms) for services accessing different coordination attributes (Time2-Time1)**

| No of Services | Service 1 | Service 2 | Service 3 | Service 4 | Service 5 |
|---|---|---|---|---|---|
| 1 | 809±85 | 785±90 | 730±74 | 709±71 | 684±58 |
| 2 | 812± 91 | 833±111 | | | |
| 3 | 875±99 | 889±110 | 881±119 | | |
| 4 | 994±138 | 968±147 | 905±138 | 940±115 | |
| 5 | 1117±126 | 1112±143 | 1038±151 | 1047±139 | 1070±148 |

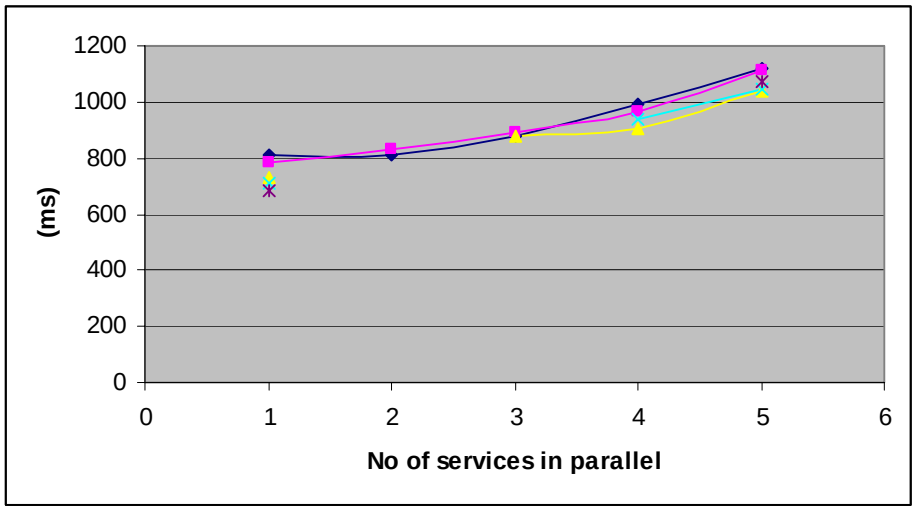


**Figure 6. Graphical display of Table 6.**

The performance appears to decrease with a mild exponential curve. In order to determine who much of this decrease was due to extra time taken by the coordination service and how much was due to extra time taken processing the GSI communications, we also measured (Time8-Time7). The results are shown in Table 7. We can see that the coordination service only accounts for about 10% of the decrease in performance.

**Table 7. Performance results (ms) for services accessing different coordination attributes (Time8-Time7)**

| No of Services | Lock/Read Att | Write/Unlock |
|---|---|---|
| 1 | 20.0±2.9 | 19.0±2.4 |
| 2 | 23.5±5.6 | 22.0±5.4 |
| 3 | 26.2±10.0 | 25.9±9.2 |
| 4 | 34.0±15.7 | 33.2±14.9 |
| 5 | 47.4±21.7 | 49.0±23.4 |

We conclude that the main inhibitors to fast overall performance and scalability of the current coordination service implementation are the overheads of using Globus Toolkit to transport the messages between the coordinator and the coordination service, with GSI posing a particularly heavy

burden to secure the communications. Further research will need to investigate more efficient ways of performing secure communications between the coordinator and the coordination database service.

## 6. RELATED RESEARCH

Numerous projects such as AKENTI [13], CAS [14], GridSite [20], GSI [5], PERMIS [2], PRIMA [9], SAZ [19] and VOMS [12] have focused on grid security and have provided technological solutions for some aspects of authorization and access control, each providing different sets of capabilities. However, none of them have included the capability of coordinating access control decisions across time and space. Likewise, many researchers have considered history based access controls systems e.g. for mobile code [21], for separation of duties [22], for running code in JVMs [23], but none have considered how these might be applied to distributed grid systems. Research into grid accounting systems is probably the closest research related to ours.

The primary objective of the SweGrid Accounting System (SGAS) [8] is to enforce shared resource quotas across organisational boundaries. Their model is based on a virtual Bank which has accounts for grid research projects, into which resource quotas are periodically placed by administrators. When a grid researcher submits a job, the request is intercepted by the Job Account Reservation Manager (JARM), and a time-limited hold is placed on sufficient funds to run the job from the project's account. When the job has finished, JARM intercepts the job execution process to calculate the actual cost of the job, then asks the Bank to remove the hold and debit the account with the actual amount of resources consumed (which must be less than or equal to the held amount). JARM gains access to the bank account by using the proxy certificate delegated to the job by the user. The Bank service keeps a complete transaction history of all accesses to a user's account, and holds on units automatically expire after the timeout period to cater for jobs that crash prematurely. The model supports soft authorisation controls by allowing an overdraft facility on accounts and setting the job's execution priority. An overdraft of zero with no differential priorities equates to hard authorisation decision making, since a user with insufficient funds will be denied permission to run his job. A large overdraft facility and/or different job priorities equates to soft authorisation decision making, since users can consume more than their quotas up to the limit of their overdraft, or have their job's priority downgraded if they have insufficient resources. Operational experience has indicated that researchers need these facilities because they typically work in burst mode, with periods of heavy activity followed by periods of calm. The SGAS Bank serves a similar purpose to our coordination database, and JARM to our coordination PDP. A primary difference is that SGAS intercepts a job before and after it has completed, whereas ours only intercepts it before submission (we discuss this further in Section 7). Project quotas in SGAS can be delegated and distributed to members of project teams, in order to increase the granularity of accounting, though it still isn't as granular as our coordination attributes which can be filtered on action and environmental parameters as well as resource and subject ones. Operational experience of SGAS found the system to be too complex and time consuming to manually allocate resources to projects, and to have PIs delegate quotas to project members. In comparison, our system has a schema which automatically allocates initial values to all coordination attributes.

Sundaram and Chapman [17] [18] have developed a policy-based decision framework, called Policy Engine, that provides authorization and accounting. Their goal is to define a resource broker that automatically selects the best resource grid that is available based on the requirements of the user's job. The solution considers both the user's permissions at the broker side and the accounting details at the resource side to ensure the allowed quotas and usage credits are not exceeded. Usage credits are not defined but depend upon a negotiated agreement between the broker and the resource site for the accounting metrics that reflect the resource's usage. In the first version [17], policies are specified as a set of (attribute, value) pairs stored in text files. Each value states the maximum

allowed usage value for the associated attribute. The policy evaluation module includes a set of attributes managers that are responsible for ensuring that the values are not exceeded by a job. In version 2 [18], policies are specified in XML format. The policy language is also improved with the use of logical and arithmetic operators. A new attribute "type" is also added to define the scope of each rule: when its value is "site", the rule applies for all users, when it is "user", it applies for a specific user. However, their attributes are not as flexible as our coordination attributes which allow any granularity to be defined over users, resources, actions and the environment to define the scope of a policy rule. Furthermore, coordination attributes allow more flexible and precise information to be coordinated rather than just resource credits. Finally, whilst their system provides history based decisions, there is no distributed coordination between PDPs.

Dumitrescu et al. propose a model for usage policy [24] and a distributed approach [10] for grid resources brokering. The policy for the allocation of resources is a set of 5-tuples of the form <resource-type, provider, consumer, epoch-allocation, burst-allocation> where epoch and burst allocations specify a percentage for a period. The management architecture [10] can include more than one VO decision point (which they name VO policy enforcement points). All the VO decision points are connected in a mesh. Each VO decision point maintains a view of the global environment via the periodic (in their experiments they used three minutes) exchange of information about the recent jobs they have authorized, which they send to the other VO decision points. Their resource types are limited to low level grid resources (CPU, network bandwidth and storage). In addition, exchanging information periodically is not a good solution for coordinating the VO decision points. If the period is too long, the VO decision points' view of the global environment will not reflect current reality. If the period is too short, many useless messages are exchanged. Finally, no obligations can be specified.

Gama et al. in [15] propose a scalable architecture that is able to enforce history-based policies. The policy language [27] is an extension of SPL [26], is named xSPL, and it allows history based constraints to be specified in obligation policies. This architecture has been adapted to enforce policies for grids [25]. In their context, they have obligations, but these don't have exactly the same semantics as in our approach. Their obligations enforce mandatory constraints on future users' actions, whereas our obligations place an immediate action on the PEP that is to be enacted along with the current user action. Furthermore, coordination between multiple PDPs is not considered by them, and there isn't anything equivalent to our Chronicle parameter.

Other related research can be found in different contexts. Prepaid systems (PPS) provide a popular service for mobile telecommunications. These require the customer to pre-purchase a certain amount of time before any calls can be made. When customers have used all the time they have paid for, they can't originate any more phone calls until they further provisions their accounts. These systems are time and space independent, and different solutions have been developed e.g. [16]. For example, in the wireless intelligent network approach, the mobile switch centers, before allowing a communication, ask a prepaid service control point that maintains the balance of the customer's account if there are sufficient funds present in it. This is similar to our coordination database approach.

## 7. LIMITATIONS, CONCLUSIONS AND FUTURE WORK

As stated previously, our current implementation only implements the Chronicle=Before option, since the Coordinator has to update the coordination database before returning the granted response to the GT4 PEP. It would have been possible to implement the After and With options as well if the GT4 PEP had been configurable to make a second call out to our Coordinator after the user's service had completed successfully. Our preferred solution is for the coordination infrastructure to be more tightly integrated into GT by taking the various components of the Coordinator and integrating them into the GT PEP as shown in Figure 7.

The Coordination PIP and Obligations Service, which are currently part of the Coordinator, would become stand alone services directly called by a future GTn PEP. In addition, the GTn PEP will need enhancing to repeatedly call the custom PIPs and the PDP, in a loop, if the PDP returns a set of additional attributes that are needed before an authorization decision can be made (as in the XACML model). Once an authorization decision has been returned the PEP will need to call the obligations service for it to process the obligations, update the coordination database, and then remove the coordination database locks.

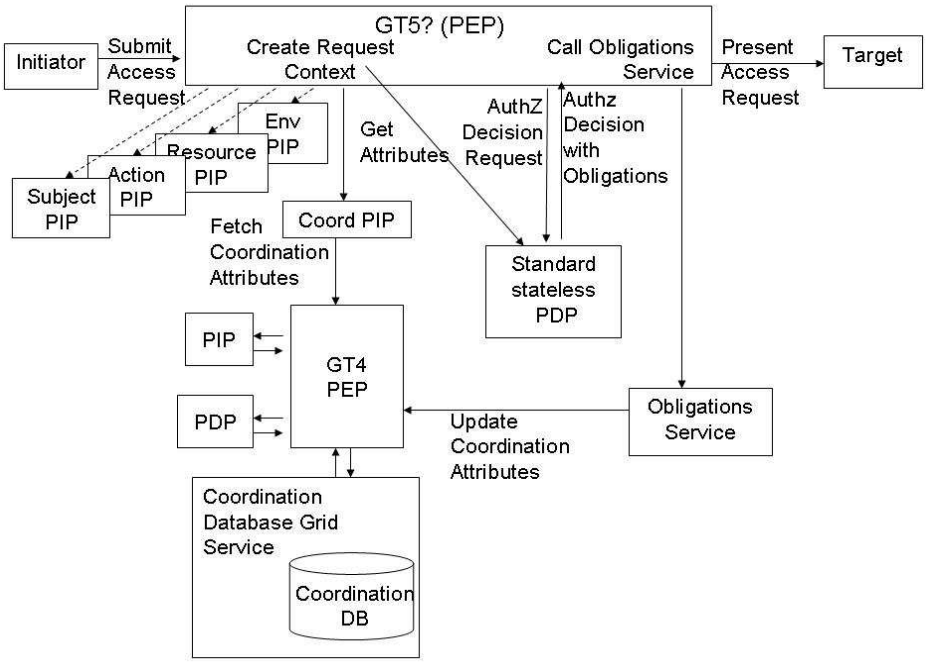


**Figure 7. Preferred Model for Integrating Coordination into GT**

In order to correctly handle the Chronicle parameter, the PEP needs to undertake enforcement of the user's access request and updating of the coordination database in different sequences. Unfortunately the type of Chronicle is not known until after the authorization decision has been made, but locking the coordination database is needed before the decision is made, i.e. when the coordination values are first read by the coordination PIP. This leads us to define two different procedures for the PEP, which we term Lock All Decide Once and Multiple Decisions. The former procedure is used if the PEP is able to get the full set of attributes that are needed at initialization time, the latter if it is not (i.e. the XACML model). Both procedures require the PEP to call the obligations service before and after the user's request has been enforced.

In Lock All Decide Once, the PEP obtains the full set of coordination and environmental attributes that are needed by the PDP in either an application specific manner or via a call to the PDP's getAttributes method at initialization time. After a user presents an access request, the PEP calls the custom PIPs, and the coordination PIP locks and fetches the appropriate coordination attributes from the coordination database. The PEP then calls the stateless PDP passing it the full set of request context attributes. An authorization decision and obligations are returned. The PEP makes the first call to the obligations service which determines the value of the Chronicle parameter. If Chronicle=After, the obligations service removes the coordination attribute locks and returns to the PEP for it to enforces the user's access request. After the user's request has finished the PEP makes the second call to the obligations service for it to update the coordination database. If Chronicle=Before, the obligations service updates the coordination attributes, removes the locks and

returns to the PEP for it to enforce the user's access request. The second call to the obligations service performs a null action. If Chronicle=With, the obligations service returns to the PEP for it enforce the user's access request. After the user's access has completed, the PEP calls the obligations service for the second time and it updates the coordination attributes and removes the locks.

In the Multiple Decisions procedure the PEP does not call getAttributes at initialization time, nor the environmental or coordination PIPs when a user's access request is first presented, but rather calls the subject, action and resource PIPs followed by the PDP. The PDP returns the set of environmental and coordination attributes that are needed for this access request, and the PEP calls the appropriate PIPs. If the coordination PIP is called, it will lock the coordination database and retrieve the coordination attributes. The PDP is then called for a second time, passing it the required environmental and coordination attributes. (Note that this process may need to be repeated again if the PDP returns further needed attributes instead of an authorization decision.) Once the authorization decision and obligations are returned the procedure continues in the same way as before in the Lock All Decide Once procedure.

In conclusion, we have shown how coordination between access control decision making can be modeled and implemented in a grid environment. We have defined the necessary coordination attributes and obligations that are needed to support the model, including a Chronicle parameter that indicates when the coordination attributes have to be updated. We have implemented the Chronicle=Before procedure in GT4 by building a coordinated PDP and presented the performance results. Finally, we have described how coordinated decision making can be more tightly integrated into GT so as to support the Chronicle=After and Chronicle=With variants as well. We are not aware of any other similar research, although several grid accounting systems have similar objectives to our work. Future research could usefully investigate integrating our coordinated authorization system with grid accounting systems and finding a more efficient way of securing the communications between the coordinated PDP and the coordination database service so as to reduce the communications overheads of GSI.

## 8. ACKNOWLEDGMENTS

We should like to thank the UK EPSRC who have funded this research under the Distributed Programmable Authorisation project (GR/S69061/02).

## 9. REFERENCES

1. S. Tuecke, V. Welch, D. Engert, L. Pearlman, M. Thompson. "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile". RFC3820, June 2004.

2. D.W.Chadwick, A. Otenko "The PERMIS X.509 Role Based Privilege Management Infrastructure". Future Generation Computer Systems, 936 (2002) 1–13, December 2002. Elsevier Science BV

3. Von Welch, Rachana Ananthakrishnan, Frank Siebenlist, David Chadwick, Sam Meder, Laura Pearlman. "Use of SAML for OGSI Authorization", Aug 2005

4. Su, L. Chadwick, D.W., Basden, A., Cunningham, J.A.. "Automated Decomposition of Access Control Policies". Proc of 6th IEEE International Workshop on Policies for Distributed Systems and Networks, Stockholm, 6-8 June 2005. pp 3-13

5. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., and Tuecke, S. (2003) "Security for Grid Services", 12th IEEE International Symposium on High Performance Distributed Computing

6. OASIS "eXtensible Access Control Markup Language (XACML) Version 2.0" OASIS Standard, 1 Feb 2005

7. David W Chadwick, Linying Su, Oleksandr Otenko, Romain Laborde. "Coordination between Distributed PDPs". Proc of 7th IEEE International Workshop on Policies for Distributed Systems and Networks, London, Ontario, 5-7June 2006 pp163-172

8. E. Elmroth, P. Gardfjell, O. Mulmo, and T.Sandholm. An OGSA-Based Bank Service for Grid Accounting Systems. In J. Wasniewksi et. al. (eds). Applied Parallel Computing. State-of-the-art in Scientific Computing. Springer Verlag, Lecture Notes in Computer Science, 2004.

9. Markus Lorch, Dennis Kafura. "The PRIMA Grid Authorization System". Journal of Grid Computing, Volume 2, Number 3 / September, 2004

10. Catalin L. Dumitrescu , Michael Wilde and Ian Foster. A Model for Usage Policy-Based Resource Allocation in Grids, in Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05), p 191 – 200, IEEE Computer Society, 2005.

11. ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 "Security Frameworks for open systems: Access control framework"

12. 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

13. Johnston, W., Mudumbai, S., Thompson, M. "Authorization and Attribute Certificates for Widely Distributed Access Control," IEEE 7th Int Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), Stanford, CA. June, 1998. Page(s): 340 -345

14. L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. "A Community Authorization Service for Group Collaboration". Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002

15. Pedro Gama, Carlos Nuno da Cruz Ribeiro and Paulo Jorge Pires Ferreira. A Scalable History-based Policy Engine, in Proceedings of the Seventh IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2006), IEEE Computer Society, 2006.

16. Yi-Bing Lin, Ming-Feng Chang, Herman Chung-Hwa Rao, Mobile prepaid phone services, in IEEE Personal Communications, Vol. 7, N°3, p6-14, 2000.

17. Babu Sundaram, Barbara M. Chapman. Policy Engine: A Framework for Authorization, Accounting Policy Specification and Evaluation in Grids Source, in Proceedings of the Second International Workshop on Grid Computing, LNCS 2242, pages 145 - 153, 2001.

18. Babu Sundaram, Barbara M. Chapman. XML-Based Policy Engine Framework for Usage Policy Management in Grids, in Proceedings of the Third International Workshop on Grid Computing, LNCS 2536, pages 194 - 198, 2002.

19. Saz. See http://www.fnal.gov/docs/products/saz/v_vo1/SAZ.htm

20. A. McNab, "The GridSite Web/Grid security system" Softw. Pract. Exper., vol. 35, no. 9, pp. 827-834, 2005.

21. Edjlali, G., Acharya, A., and Chaudhary, V. 1998. History-based access control for mobile code. In Proc. 5th ACM Conf. on Computer and Communications Security (San Francisco, California, USA, November 02 - 05, 1998). CCS '98. ACM Press, New York, NY, 38-48

22. T.T.Simon and M.E.Zurko. "Separation of duty in role-based environments". Proc. 10th Computer Security Foundations Workshop, pp.183-194. IEEE Computer Society Press, June 1997.

23. Mart´ın Abadi, C´edric Fournet. "Access Control based on Execution History". Proc of 10th Annual Network and Distributed System Security Symposium, (NDSS'03), San Diego, California, 6–7 February 2003.

24. Catalin Dumitrescu, Ioan Raicu and Ian Foster. DI-GRUBER: A Distributed Approach to Grid Resource Brokering, in Proceedings of ACM/IEEE conference on Supercomputing, 2005.

25. Pedro Gama, Carlos Nuno da Cruz Ribeiro and Paulo Jorge Pires Ferreira. Heimdhal: A History-based Policy Engine for Grids, in Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'06), IEEE Computer Society, 2006.

26. Carlos N. Ribeiro, André Zúquete, Paulo Ferreira and Paulo Guedes. SPL: An access control language for security policies with complex constraints. In Proceedings of Network and Distributed System Security Symposium (NDSS'01), February 2001.

27. Pedro Gama and Paulo Jorge Pires Ferreira. "Obligation policies: an enforcement platform", in Proceedings of Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (Policy 2005), IEEE Computer Society, 2005.

28. The PERMIS PDP can be obtained from http://sec.cs.kent.ac.uk/permis/

29. Sun's XACML PDP can be obtained from http://sunxacml.sourceforge.net/

30. Shewhart Control Chart. See http://www.itl.nist.gov/div898/handbook/mpc/section2/mpc221.htm