

# Providing Secure Coordinated Access to Grid Services

David W Chadwick  
University of Kent  
Computing Laboratory  
Canterbury CT2 7NF  
+44 1227 82 3221

d.w.chadwick@kent.ac.uk

Linying Su  
University of Kent  
Computing Laboratory  
Canterbury CT2 7NF  
+44 1227 82 4768

L.Su-97@kent.ac.uk

Romain Laborde  
University of Kent  
Computing Laboratory  
Canterbury CT2 7NF  
+44 1227 82 4769

r.laborde@kent.ac.uk

## ABSTRACT

Coordinating the cumulative use of distributed resources in a grid environment so that users do not consume too much is a difficult task. This paper presents one approach that we have implemented in Globus Toolkit version 4 (GT4), that uses an SQL database to hold “coordination” data, and policy decision points (PDPs) to make access control decisions about whether the user’s request for more resources can be granted or denied. When access is granted, obligations in the policy ensure that the coordination database is appropriately updated. In our initial implementation, the coordination service is imbedded into the GT4 authorization chain as a custom PDP so that any web service can be provided with a security policy that provides a coordination capability. In the final section we describe how coordinated decision making could be more tightly integrated into a future version of GT.

## Categories and Subject Descriptors

D.4.6 Security and Protection: *Access Controls*

C.2.4 Distributed Systems: *Distributed applications*

## General Terms

Design, Security, Standardization.

## Keywords

PDPs, coordinated decision making, policy based access controls, grid computing

## 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 that a user may request per day or per job from any storage location on the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MGC '06, November 27, 2006 Melbourne, Australia Copyright 2006 ACM 1-59593-581-9... \$5.00

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 this presents a number of challenges to the designer of a policy based coordinated access control system.

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 the PERMIS authorisation infrastructure [2] 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.

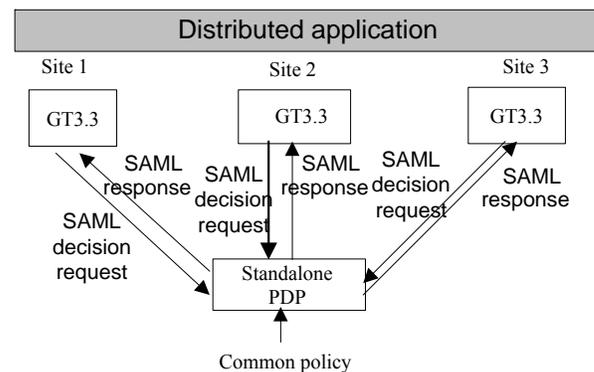
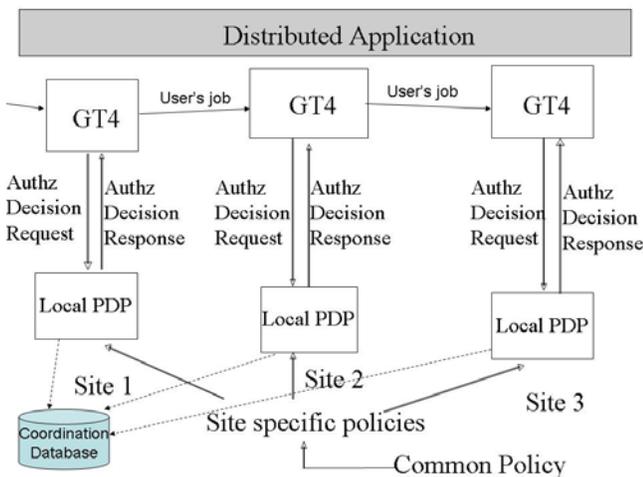


Figure 1. Use of a common policy in today’s distributed grid applications

It is far more preferable for each site to have its own PDP under its direct control and for the common policy to be distributed to it so that the administrator can load it into his PDP. This approach will increase the performance of the access control decision making, but it 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 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. 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 are stateless, so they will not need to be modified in order to support our secure coordination service. Secondly, if the state information were 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. Thirdly, by making the coordination database a grid service, we get the benefits of secure communications between the PEP and the service using the existing grid security infrastructure (GSI) [5]. Finally, by using database technology we benefit from its extensive research into fast and efficient data handling, searching, locking, distribution etc.



**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 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 coordination aware PDP and how this is currently integrated into GT4. Section 5 concludes by looking at the limitations of the current research, reviewing related research, and indicating where future research is 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. The PDP is given details (in the form of attributes) about the subject, the resource, the requested action and the environment (e.g. current time, temperature etc.). 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. may only access resources of type printer), the action (e.g. may only print a maximum of 10 pages), and the environmental (e.g. between 9am and 5pm) attributes. If any access control decision will produce state changes in this context and these changes will affect future access control decision, then coordination between the access control decisions is required. 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 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 request context attributes that need to be coordinated, makes the coordination process independent of the number of PDPs involved in the grid access control decision making.

### 2.1 Coordination Attributes

In order to store the coordination data we have defined a fifth type of request context attribute which we term 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 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 those types of attributes.

We specify a coordination attribute as follows:

$$\text{Att}[\text{SubDim}, \text{ResDim}, \text{ActDim}, \text{EnvDim}](C)$$

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 are: *usage*(C) which means the coordination attribute called *usage* has a single value that is used by all subjects accessing all resources over all actions and environments; *usage*{username(S)}(C) which means the coordination attribute called *usage* has a different value per subject, where each subject is identified by their username attribute.

An example policy with a coordinated constraint is: *users, identified by their userIDs, cannot use more than 3GB of storage each throughout the grid.* This can be written as  $\text{type}(R)=\text{storage} \wedge \text{type}(A)=\text{use} \wedge \text{amount}(A) + \text{alreadyUsed}\{\text{userID}(S)\}(C) \leq 3$ .

Most PDPs should now be capable of evaluating this type of policy, providing the request context contains the value of the *alreadyUsed* 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. All the PDP needs to do is compare the value of the coordination attribute presented in the request context with the corresponding value in a policy constraint.

## 2.2 Obligations

The next thing we need to ensure is that the coordination attributes are updated once/if the user has been given 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 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. In order to specify when the obligations should be enacted, we define the Chronicle parameter of an obligation. The Chronicle parameter 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 the With value.

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 action prior to it 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 user's action will not be recorded and the user may be allowed to perform the same request again, in contravention of the coordination 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 it does mean that transactions have to be enacted on the coordination database, and the coordination attributes have to be write locked for the duration of the user's action. This may cause an unacceptable 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,
- *getCoordAttrVal* returns the value of a coordination attribute given its name and an XACML 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.
- *setCoordAttrVal* similarly sets the value of a coordination attribute.
- *isCoordAttr* queries if a coordination attribute of this name exists in the coordination database

- `getAttributeDefinition` returns an XML element which contains the definition of the coordination attribute including the attributes that are embedded in its name
- `lockCoordAttrs` allows multiple attributes in the database to be read or write locked
- `unlockCoordAttrs` removes all the locks in the database held by the current thread.

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

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

where  $|\text{XDim}|$  represents the number of members in the set  $\text{Dim}$  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 ( $\text{id} = X$ ) wants to access a file ( $\text{mode} = M$ ,  $\text{file Name} = F$ ) on a filestore ( $\text{id} = Y$ ), 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 an 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 part of the schema of the coordination database grid service.

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

The coordination database grid service, being a standard grid service, is protected by GSI and its own PDP that ensures that only authorised coordination aware PDPs can access it. This is easily achieved by assigning a digitally signed X.509 attribute certificate with a role of “Coordinator” to the Coordinator component of the GT4 Custom PDP of Figure 3, and having a standard role based access control (RBAC) policy that says that only subjects with the role of “Coordinator” are allowed to access the coordination database. This allows any number of coordination aware PDPs to access the coordination database. Each Coordinator has its own public key certificate and DN so that it can strongly authenticate to the coordination database grid service. In our implementation the “Coordinator” attribute certificate is pulled from an LDAP repository by the service’s PDP (we use the PERMIS PDP that supports both the push and pull modes of attribute retrieval).

## 4. THE COORDINATION AWARE PDP

The GT4 authorisation framework implements a decision engine which evaluates 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 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.wsrp.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.wsrp.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 coordination aware 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 5, but still allows secure coordination to take place. In section 5 we describe how the coordination infrastructure could be more tightly integrated into GT to become an integrally supported feature.

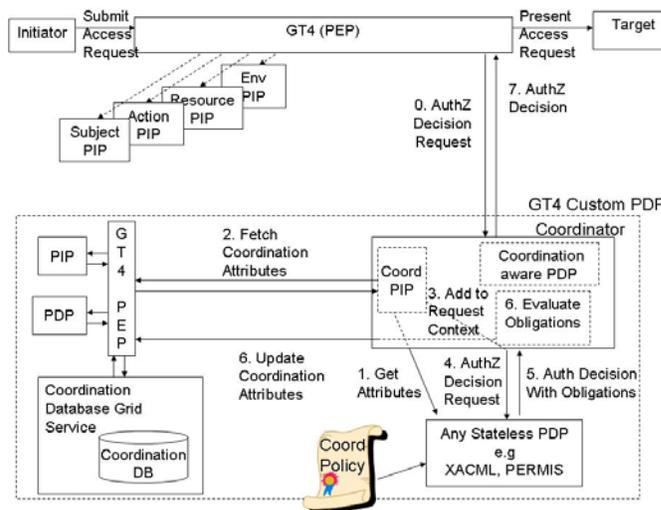


Figure 3. Adding coordination to GT4

When GT4 is given a user’s request to access a grid service, our coordination aware PDP is called via the GT4 authorisation chain. Our java code (labelled Coordinator in Figure 3) is acting as a PDP from GT4’s perspective, and a PEP from the actual PDP’s perspective. The actual PDP is the one making authorisation decisions based on the coordinated access control policy. One of the first steps the Coordinator takes is to write lock and retrieve the required coordination attributes from the coordination database grid service, using its in-built coordination PIP.

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 authorisation decision request can return the set of additional attributes that are needed in order to answer the current authorisation decision request. This latter 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 PDP only needs to be called once per authorisation decision, but will 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, whilst the former approach is expected to be most efficient for small policies that only require a limited number of attributes. We have currently implemented the second approach and have added a `getAttributes` method to the XACML and PERMIS PDPs. Future work will be to measure the performance trade offs between the two approaches.

When the appropriate coordination attributes have been returned from the coordination database grid service, the Coordinator adds them to the request context obtained from the GT4 PEP, and passes this expanded context to the actual PDP. The coordination attributes are actually packaged as environmental ones. The PDP returns an authorisation decision, according to its evaluation of the user's request against the coordinated access control policy. If the PDP returns access granted the authorisation decision may contain obligations to update appropriate coordination attribute values. The Coordinator makes various calls to an obligations service (see below) which evaluates the obligations and updates the coordination database. The Coordinator then removes the locks on the database and returns the granted response to the GT4 PEP (minus the obligations). 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 5 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, 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, and updates the coordination database by making repeated calls to `setCoordAttrVal`.

We have currently only implemented a simple service as proof of concept, which simulates an ATM withdrawal. When the service is asked to dispense money it returns a receipt to the user. Without the coordination capability, a standard PDP can only have a policy constraint that limits each withdrawal to a particular value, say £250. If a user requests £250 or below the request will be granted by the PDP and the ATM service will return a receipt saying the money has been dispensed. If a user requests greater than £250 the PDP will deny access to the service. Of course, a user can make several consecutive requests for £250 or less and therefore withdraw an unlimited amount of money. Once the coordination aware PDP is configured into the grid service, a user can only withdraw up to £250 per day, and after that all his 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 DB service.

## 5. LIMITATIONS, CONCLUSIONS AND FUTURE WORK

As stated previously, the 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 might have been possible to implement the Chronicle=After option as well if we could have made GT4 make a second call out to our custom PDP after the user's service has completed successfully, but we did not think that this extra effort was necessarily worthwhile. Our preferred solution is for the coordination infrastructure to be tightly integrated into GT by taking the various components of the Coordinator and integrating them into the GT PEP as follows.

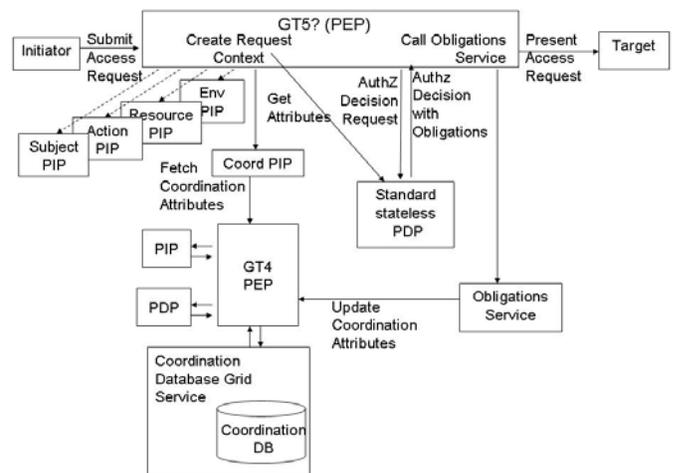


Figure 4. Integrating coordination into GT

The Coordination PIP and Obligations Service, which are currently part of the Coordinator, will become stand alone services directly called by the GT PEP. In addition, the PEP will be responsible for setting the write locks on the coordination attributes in the coordination database. The GT PEP will need enhancing to repeatedly call the custom PIPs and the PDP, if the

PDP returns a set of additional attributes that are needed before and 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 to process the obligations and update the coordination database, and then remove the coordination database locks.

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).

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. After each user presents an access request, the PEP locks the coordination attributes in the coordination database and calls the custom PIPs (subject, action, resource, environment and coordination) to fetch the required attributes. The PEP creates the expanded request context and calls the PDP passing it the full set of attributes. An authorization decision and optional obligations are returned in the response context. The PEP calls the `getChronicle` method of the Obligations Service passing it the returned authorization response. If `Chronicle=After` is returned, the PEP removes the coordination attribute locks, enforces the user's access request, and after this has finished updates the coordination database (through calls to `Evaluate` and `Perform Obligations`). If `Chronicle=Before` is returned, the PEP updates the coordination attributes first, then removes the locks and afterwards enforces the user's access request. If `Chronicle=With` is returned, the PEP enforces the user's access request and after this has finished, updates the coordination attributes and finally removes the locks.

In the Multiple Decisions procedure the PEP does not call the environmental or coordination PIPs after the user has presented an access request, since it does not know which attributes are needed, but rather calls the PDP first. The PDP returns the set of environmental and coordination attributes that are needed for this access request, and the PEP locks the coordination attributes and calls the custom PIPs to retrieve the attributes. The PDP is now 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 optional 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 custom

PDP. 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 grid accounting systems such as the SEGAS Bank service [8] address similar concerns. In this, the user's access request is intercepted by the Job Account Reservation Manager (JARM) and an amount of units, sufficient to run the job, are placed on hold in the user's account. If a user has insufficient funds his job will be rejected. JARM gains access to the user's bank account by using the proxy certificate delegated to the job by the user. Once the user's job has finished JARM debits the user's account with the actual amount of units consumed. The Bank service keeps a complete transaction history of all accesses to a user's account, and holds on units automatically expire after a user determined period to cater for jobs that crash prematurely. Future research should investigate integrating our authorization system with grid accounting systems.

## 6. ACKNOWLEDGMENTS

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

## 7. 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 Ananthkrishnan, 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 6<sup>th</sup> 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 7<sup>th</sup> IEEE International Workshop on Policies for Distributed Systems and Networks*, London, Ontario, 5-7 June 2006 pp163-172
- [8] E. Elmroth, P. Gardfjell, O. Mulmo, and T.Sandholm. An OGSi-Based Bank Service for Grid Accounting Systems. In J. Wasniewski et. al. (eds). *Applied Parallel Computing. State-of-the-art in Scientific Computing*. Springer Verlag, Lecture Notes in Computer Science, 2004.

