This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2009 proceedings

Distributed Usage Control Architecture for Business Coalitions

Maicon Stihler, Altair Olivo Santin, Alcides Calsavara, Arlindo L. Marcon Jr.
Pontifical Catholic University of Paraná / Graduate Program on Computer Science
Curitiba, PR, Brazil
Email: {stihler,santin,alcides,almjr}@ppgia.pucpr.br

***Abstract*—The dynamic environment of Business Coalition (BC) requires a flexible access control approach to deal with user management and policy writing. However, the traditional approach applied to BC assigns to access control a burden, mainly to the service provider, thus requiring ad hoc schemes to mitigate the lack of controls developed to BC needs. We present a brokered access control architecture, based on UCON$_{ABC}$, to obtain an integrated usage control management for BC. The broker intermediates contract establishment between service provider and consumer, and derives from it the policies to regulate the usage at service-level. The consumer defines user-level policies to control the usage of the contracted services. We developed a web services based prototype to evaluate the feasibility of our proposal. The proposed architecture enables distribution of duties and integration of usage control management in a loosely coupled fashion, providing the flexibility desired in BC environments.**

I. Introduction

Organizations with different resources and skills wish to join with others and compose a business coalition (BC) to perform tasks beyond their individual business capabilities, i.e. each partner organization sells services to others, aiming to collaborate in the BC. However, the dynamism of BC presents a challenge to traditional access control architectures.

User management and policy writing are cumbersome tasks in BC. In traditional approaches the provider must develop both tasks, which may negatively impact their service providing. Setting up coarse grained controls (e.g. controlling groups instead of users) is a common approach to reduce management costs, however this approach may impose security management limitations (e.g. for auditing purposes).

The intrinsic dynamism of BCs exposes the deficiencies of traditional access control approaches. Attributes related to users, objects, and the environment may change at any time in BC. However, traditional access controls can not reflect those changes, therefore exposing the provider to possible usage abuse. It is possible to set up controls to prevent such situations, but they are not naturally integrated with the access control mechanisms, thus inflating management costs.

We assume a BC environment that is composed of a Broker, Provider organizations, and Consumer organizations. A Broker maintains a set of administrative services to intermediate the establishment of contracts between Consumers and Providers. A Consumer organization is a client of the services offered by a Provider organization.

We propose an architecture that splits up the policy-writing duties between the provider and the consumer in order to provide a more rational policy management. It employs the usage control model, UCON$_{ABC}$ [1], to achieve highly dynamic control, which are more appropriate for BCs. We take an approach that diverges from the traditional ones, like Grid Computing [2], because it is decentralizing. Another distinguishing feature is the presence of a broker – an administrative entity that intermediates the deals between consumers and providers. The broker defines which consumers hold access rights to a provider's service according to the established contracts. Also, it performs the translation of contracted service agreements into their respective computational representation. Our architecture favors providers autonomy with respect to usage decision, while assuring coherence with contracted service specifications.

The paper is organized as follow. On Section II, we review the fundamentals related to our work. Section III addresses the proposed architecture. Section IV presents a prototype implementation. Section V presents some related work. Section VI considers the prototype evaluation. Section VII presents our conclusions.

II. Usage Control and Policy Architectures

Traditional access controls (i.e. discretionary access control, mandatory access control and role-based access control) are unable to take into account changes that may happen in the user, environment or object attributes, after an authorization is granted.

The usage control model (UCON$_{ABC}$) extends traditional access controls to support attribute mutability, as well as to employ a continuous evaluation of the controls. Attribute mutability means that information related to users and objects may change during object usage, and that potentially affects the authorization of a user over an object. Through continuous policy evaluation, these attribute changes are taken into account, mainly during the object usage.

The UCON$_{ABC}$ controls are defined as Authorizations, Obligations, or Conditions, which may be applied before a usage session starts (pre-controls) or during it (ongoing-controls). Authorizations verify if a user holds the rights required to access an object. Obligations are actions (e.g. to keep a pop-up window open while accessing a web site) that must be performed by someone – who is not necessarily the requesting user – to grant or maintain access to an object. Conditions deal with attributes (e.g. disk space availability and

978-1-4244-3435-0/09/$25.00 ©2009 IEEE

average system load) that are independent from the user and object being used.

Typically, a policy control architecture is composed of two main entities – namely, the policy decision point (PDP) and the policy enforcement point (PEP) – and employs either pull mode or push mode [3]. On pull mode (also known as outsourcing model), the PEP asks the PDP for authorization decision on every user access request. The PDP evaluation is based on the applicable policy retrieved from a policy repository (PAP – Policy Administration Point). The PEP enforces the PDP decision. On push mode (also known as provisioning model), the PEP is pre-configured by the PDP on start-up, i.e. the policies are provisioned on a PEP's local PAP (LPAP). Thus, the authorization evaluations are made by the PEP's local PDP (LPDP), such that no interaction happens with the remote PDP.

An alternative to that approach is to apply SDSI (Simple Distributed Security Infrastructure) [4] and SPKI (Simple Public Key Infrastructure) [5]. In SDSI/SPKI, authorization is granted by means of certificates, which by rights delegation form a chain of authorization certificates. Authenticity of delegations is assured by the digital signature of a certificate, given that in SDSI/SPKI a user/principal is a public key.

III. Proposed Architecture

In Fig. 1, we illustrate our proposed architecture, which is composed of three main entities: consumer organization, provider organization, and broker infrastructure.

The broker infrastructure contains the elements that aim at assisting the management of the BC environment. It is a combination of a notification/token service, policy repositories, provided service announcing and searching service, inter-domain identity service, and a PDP/PAP. Basically, a consumer is composed of a client-side for the provided service and the consumer administration interface for user-level policy repository handling. The provider comprehends the provided service, service monitoring and service-level policy enforcement infrastructure.

The BC partners establish a business agreement with the broker by specifying the amount of the provided service that can be contracted by the consumers. The business agreement is carried out through the delegation of rights from provider to broker when the provided service is announced. A SDSI/SPKI authorization certificate is employed for delegation. Therefore, the broker can only establish a contract with consumer regarding either the total or a partial provided service amount, thus preventing business agreement violations.

After locating the desired service and establishing a corresponding contract service agreement with the broker, the consumer receives a credential that is issued by the broker. Also, the corresponding contract service agreement document is stored on the broker's repository, and a certificate with the agreed service amounts is delegated to the consumer. This certificate enables the consumer organization to write usage policies for the respective provided service on the broker's policy repository (Fig. 1, event 1).

Such policy writing is done through the consumer's administration interface. This policy may contain predicates that refer to user, service, and environment attributes. The consumer's administrator is also responsible for issuing authorization certificates for her users, so that they can gain access to the contracted services (Fig. 1, event 3).

The service-level policy is derived from the contracted service agreement and provisioned to the service provider together with the user-level policy (Fig. 1, event 2). The service provider controls the amount that each consumer can use on the provided service (contracted service), based on the service-level policies. That dispenses with the need for the provider to write a new policy for each contract established by the broker.

The provided service's attributes are available through a user-friendly interface, so the consumer policy administrator does not need to understand how they are handled by the provider. Such facility is only accessible to the consumer because the provider implements a mechanism-independent service monitoring infrastructure.

Because we aim at keeping our architecture independent of a centralized policies management mechanism, we adopt a policy provisioning scheme that permits the providers to work more autonomously [6]. The usage policies are pushed to the provider from the broker's repository and stored on the local policy repositories (provider's LPAP). Thus, we avoid any overhead due to PEP/PDP message-exchange (applied in outsourcing model) and the single point of failure and vulnerabilities intrinsic to such policy control architecture.

Regarding the PDP provisioning each policy repository update, there is no need to be concerned about policy inconsistencies between PAP and LPAP. When the PDP goes down for any reason, the LPDP continues working even though there might be a policy inconsistency. However, as soon as the communications between them are reestablished, PAP and LPAP synchronization automatically takes place. This approach is advantageous compared to the outsourcing model, since the LPDP keeps working, even though the PDP is unavailable.

The SDSI/SPKI name certificates form the backbone of the inter-domain identification system, as shown in Fig. 1. All the provider needs to know is that the certificate presented by a user is valid; the user identity is known only by the consumer.

Immutable user attributes may be carried on its identification certificate. Thus, we remove the need for complex attribute repositories, which are prone to become a performance bottleneck. However, the provider may deploy temporary local attribute repositories for the consumer usage, as needed.

After a user has received her authorization certificate (Fig. 1, event 4), she must subscribe to the notification service (event 5). A notification service is employed to alert a user that a contracted service's amount limit has been reached and therefore a violation has been triggered. For that reason, every user that wants to use a service must be subscribed to the notification service.

The notification service (Fig. 1, event 10) may be configured

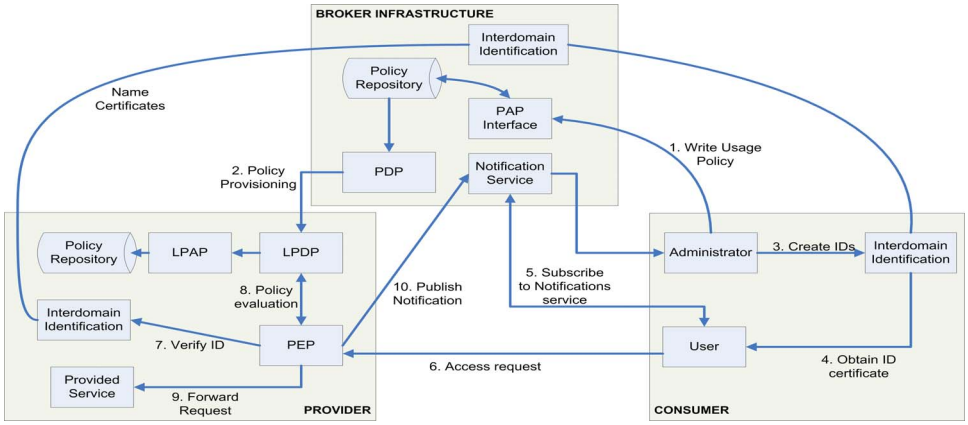

Fig. 1. Proposed Architecture

to send many types of messages (e.g. an alert that a usage quota is about to exhaust, or that a token is due to expire and therefore must be renewed). This way we make sure that the user is aware that a policy violation is imminent so that she can take corrective actions to prevent it. If a user does not regularize the violation status after a given period of time, a granted access can be revoked.

An authorization certificate and a token prove that a user is subscribed to the notification service, i.e. the user is able to make requests to the service provider (Fig. 1, event 6). The PEP intercepts a request and verifies the user certificate. Also, the PEP verifies the validity of the authorization certificate chain to enforce service-level policies (event 7). Then, the PEP requests a policy evaluation to the LPDP (event 8), forwarding the user request to the service provider when the evaluation result is permit (event 9). Otherwise, the PEP takes the correspondent enforcement actions, such as publishing a notification.

A. *Supporting UCON$_{ABC}$*

The attributes applied in the evaluation of policies may change during the usage session, either as a side effect of service consumption, or as a deliberate action defined for a policy. The policy administrator may define which attributes must be updated and what should be its new value. When the PDP evaluates a policy, it invokes an external module to update a given attribute. Such external module implements the methods to handle the local attribute repository.

In our proposal, the continuity of controls is possible because the LPDP keeps track of all the attributes associated to a usage session. When any event that is relevant to the policy evaluation (e.g. an attribute update) occurs, the LPDP is notified so that it can verify which session is affected by the change and reevaluate the applicable policies against such attributes. When the LPDP detects a policy violation, it invokes the PEP to monitor the session in violating status. In order to allow the violating session to leave the exception condition, the PEP grants a period of time for the user to take corrective measures. If the session violates the policy even after the expiration of that period of time, then the session goes to idle status and the provider's administrator is notified to take an action. Alternatively, the session can be unilaterally revoked. The LPDP enables controls and updates to be applied both before the usage session starts (the UCON$_{ABC}$ pre-controls and pre-updates) and during the usage session (the UCON$_{ABC}$ ongoing-controls and ongoing-updates).

The Obligations defined on the UCON$_{ABC}$ are treated as attributes for policy evaluation purposes, i.e., the policy administrator and users are unaware of how the obligation fulfillment is monitored. The obligation function that updates obligation attributes falls under service provider responsibility and, naturally, it can not be accessible to users.

IV. Prototype

Our prototype consists of a set of Web Services [7]. The protocol used between the interacting Web Services is the SOAP - Simple Object Access Protocol [8]. The adoption of Web Services allows easy and standardized integration of heterogeneous systems.

We used WS-Security [9], whose specification defines how to apply security to Web Services. We applied a timestamp and a digital signature, and we encrypted all messages between the entities. The signatures are based on a public key system, so it is possible to uniquely identify the parties. We also use the WS-SecurityPolicy [10] specification to define how the WS-Security mechanisms are used when securing the communications between the entities of the proposed architecture. Apache Axis2 [11] is the basis for building the prototype services. WS-Security and WS-SecurityPolicy are provided by the Apache Rampart module.

The provider issues a SDSI/SPKI authorization certificate to delegate rights to the broker which, in its turn, (either fully or partially) delegates such rights to the consumer. The consumer that owns this certificate chain may create user-level policies in the PAP repository, and is able to issue authorization certificates to its domain users. Based on such chains, the users can be identified on the service provider.

The policy provisioning follows the OASIS SPML specification [12]; we employed the OpenSPML toolkit [13]. XACML policy evaluation is performed through the Sun's XACML

Implementation [14]. Since that toolkit is not aware of usage control concepts, we extended its functionalities to implement the UCON$_{ABC}$ engine.

```
******* Pre Controls ************************
01 verifyGroup( user )
02   (user.group eq "Developers")
03 verifyRight( user )
04   contains(user.permissions, "Write")
05 isSubscribed( user )
06   pep.isValid( user.token )
******* Ongoing Controls ********************
07 verifyQuota( user )
08   (( service.quotaOrg(user.OrgID) le 50)
09    and
10    (service.quotaUser(user.ID) lt 10)
11   )
12   or
13   (service.quotaUser(user.ID) lt 5)
14 verifyTimeShif( user )
15   ( env.now ge user.startTS )
16   and
17   ( env.now le user.endTS )
18 verifyToken( user )
19   pep.isValid( user.token )
```

Fig. 2. Example of predicate-based policy

A. Policy Writing and Parsing

The administrator of the consumer's access control must write a predicate-based policy (UCON$_{ABC}$) for its domain users. The policy rules are separated in sets corresponding to each session state. That is, when writing a policy, the administrator chooses to which specific control (authorization, obligation, condition) and state (pre or ongoing) it applies.

In Fig. 2, we present a predicate-based policy created through the consumer policy administration interface. In row 01, we show a predicate that verifies, in a pre-authorization, if the user group is called *developers*. Row 03 presents a predicate to verify if the user holds rights to *write* on the data storage service. The notification service subscription is checked, in a pre-obligation, in row 05. In row 06, the PEP checks the validity (authenticity and time stamp) of the notification token.

Some ongoing-controls that apply composite predicates (by using logical connectives *and* and *or*) are also shown in Fig. 2, from row 07 up to 19. In row 07, the predicate rule verifies first, by an ongoing-condition, how much quota is being used by the consumer organization as a whole (row 08). If it is *less than or equal* (le) to 50 gigabytes, the user can be graced with an extra quota of 10 gigabytes, if such amount is not already consumed (row 10). Otherwise, the user is constrained to its regular quota (row 13) - the operator *lt* means *less than*. In row 14, a predicate checks if the user is working within his/her time shift, and another predicate that verifies if the current environmental time is within the limits defined on the user certificate (row 17). In row 18, there is a predicate to check if the user fulfills its obligation with the notification service.

We use XACML [15] version 1.1 which is an open standard that provides interoperability and extensibility. As XACML is not a predicate-based language, we need a parser to convert the predicate-based policy to XACML policies in order to enable them to be evaluated by the PDP.

When editing the predicate-based policy, the system administrator is actually editing several individual policies, each one for different controls of the UCON$_{ABC}$ (e.g. pre-authorizations, ongoing-obligations and so on). Each of these policies is then converted to XACML constructs.

Our implementation of a UCON$_{ABC}$ predicate consists in creating simple and complex XACML rules. A rule can consist of complex conditional structures, and it will be evaluated exactly as true or false. We put all the rules of the same kind (e.g. pre-authorizations) on a single policy file. When the UCON Engine searches for the pre-authorizations, it will find this policy and then the PDP can verify if the pre-authorization predicates are met. The same is applicable to the other controls (conditions and obligations).

A XACML rule supports conditional tags that enable the parser to create the equivalent structures of UCON$_{ABC}$ predicates. These conditional tags are boolean expressions that help to find out if a given rule is true or false.

In Fig. 3, we present a rule that is derived from the *verifyQuota* (rows from 07 to 13) presented in Fig. 2. We suppressed some syntactical elements, like name space declarations and canonical function names.

```
01 <Rule RuleId="verifyQuota" Effect="Permit">
02  <Condition FunctionId="function:or">
03   <Apply FunctionId="function:and">
04    <Apply
05     FunctionId="function:integer-less-than-or-equal">
06     <Apply
07      FunctionId="function:integer-one-and-only">
08      <ResourceAttributeDesignator
09       AttributeId="service.quotaOrg.userOrgID"
10       DataType="integer"/>
11     </Apply>
12     <AttributeValue DataType="integer">
13      50
14     </AttributeValue>
15    </Apply>
16    <Apply FunctionId="function:integer-less-than">
17     <Apply FunctionId="function:integer-one-and-only">
18      <SubjectAttributeDesignator
19      AttributeId="service.quotaUser" DataType="integer"/>
20     </Apply>
21     <AttributeValue DataType="integer">
23      10
23     </AttributeValue>
24    </Apply>
25   </Apply>
26   <Apply FunctionId="function:integer-less-than">
27    <Apply FunctionId="function:integer-one-and-only">
28     <SubjectAttributeDesignator
29      AttributeId="service.quotaUser" DataType="integer"/>
30    </Apply>
31    <AttributeValue DataType="integer">
32     5
33    </AttributeValue>
34   </Apply>
35  </Condition>
36 </Rule>
```

Fig. 3. Example of XACML Policy Fragment

The *or* operator (Fig. 2, row 12) is the base for verifyQuota predicate. The *function:or* is the corresponding XACML code (row 02, Fig. 3). The first operand (rows from 08 to 11,

Fig. 2) of such *or* operator is an *and* operator, and its XACML equivalent is the *function:and* (Fig. 3, row 03). The *and* operator evaluates two functions: *function:integer-less-than-or-equal* and *function:integer-less-than* (rows 04 and 12, Fig. 3).

The second *or* operand (row 13, Fig. 2) is a predicate that verifies if the storage consumed by the user is *less than* its allowed quota, thereby it gets converted to a *function:integer-less-than* (row 21, Fig. 3).

The session stage to which the XACML is to be applied to is defined on the target of the policy file. We may have pre-authorizations, pre-conditions, pre-obligations, ongoing-authorizations, ongoing-conditions, and ongoing-obligations. By simply observing Fig. 3, it is not possible to say to which session stage it applies to. This happens because the target is inherited from the policy scope, therefore it is not required to explicitly define a target for each individual rule.

```
01 if( session.state == "open" ) {
02     perform(getPreUpdates(req))
03     evaluate( getPreAuthorizations(req))
04     evaluate( getPreConditions(req))
05     evaluate( getPreObligations(req))
06 } else {
07     perform(getOngoingUpdates(req))
08     evaluate( getOngoingAuthorizations(req))
09     evaluate( getOngoingConditions(req))
10     evaluate( getOngoingObligations(req)) }
```

Fig. 4. UCON$_{ABC}$ engine pseudo-code

B. UCON$_{ABC}$ Engine

When the PEP submits a request to the LPDP, such request is intercepted by the UCON$_{ABC}$ engine, which identifies what is the state of the usage session and thus instructs the LPDP to apply the adequate controls. A pseudo-code for the UCON$_{ABC}$ engine is presented in Fig. 4.

If the UCON$_{ABC}$ engine detects that the request is starting a session (row 01), it first performs the corresponding pre-updates (row 02), and then proceeds to the evaluation of the pre-authorizations (row 03), pre-conditions (row 04), and pre-obligations (row 05). If any of the evaluations fails, the process stops and the request is denied. When the session is already started, the UCON$_{ABC}$ engine follows the same procedure, but then performing the ongoing updates and evaluating the on-going controls (from row 08 to 10).

Pre-updates and ongoing-updates differ from the other policies, since they guide actions to be performed before the controls evaluation. The update policies return the identification of the functions that must be invoked to perform the updates and their arguments.

V. Prototype Evaluation

In order to evaluate our proposal, we performed a few tests. The LPDP performance is tested under stress condition to observe its behavior for the requirement of continuously evaluating UCON$_{ABC}$ policies, such that, at any time, it is possible to have as many policy evaluations occurring as there are ongoing usage sessions and starting usage sessions. We also performed experiments considering changes in the user, environment, and object attributes to check whether the policy enforcement would work as expected. The tests were repeated a hundred times to measure the average response time. The coefficient of variation observed during the measurements was always under 5%.

The tests were carried out by using three hosts (consumer, broker, provider) connected through a 100 Mbps Ethernet. The processors were AMD64 2800+ with 1024 MB of RAM. All tests were performed on a Linux 2.6.22 operating system, except for the SDSI/SPKI processing that was performed on a Windows XP operating system.

Fig. 5 presents the chart of response time (in milliseconds) measured on one of our experiments. It shows the time needed for the requester (the PEP) to receive a response from the LPDP. The XACML policy had 1000 rules, and the LPDP responds up to 100 simultaneous requests. Parallel requests considerably affects the response time, mostly due to the fact that the XACML engine is single threaded and can not handle a large number of requests efficiently.

Fig. 5 also presents the average time to reply a request as policy size increases. The time is related only to the XACML engine evaluation time, i.e., no network time is measured. It is clear that policy size plays an important role on the evaluation performance, since the XACML format is very verbose.

The evaluation showed that the approach is usable and that it presents a good potential for practical usage. However, we also observed that there is plenty of room for optimizations.

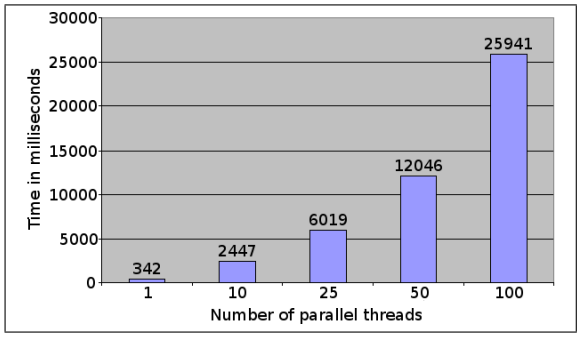

Fig. 5. Prototype evaluation chart

VI. Related Work

Zhang and his colleagues developed a prototype showing usage control concepts applied to collaborative computing [16]. The prototype does not support adequately the requirements of UCON$_{ABC}$ on distributed systems. The centralization of policy and user administration may create a single point of failure and degrade system performance. UCON$_{ABC}$ obligations are partially supported and the prototype does not comply with the XACML specification as it employs non-standardized extensions. Its performance evaluation did not mention policy size, which can influence considerably the proposal performance, nor did they study the scalability of the PDP when handling parallel requests.

Martinelli and his colleagues [17] investigate usage control on computational grids. They proposed a system that is mainly focused on usage control of a virtual machines that run programs submitted by users. The policy language is not standardized. This approach may be adequate to classical standalone applications, but it is inappropriate for distributed collaboration environments, as policies must be written for each virtual machine at system call level.

Alfieri and his colleagues [18] proposed VOMS, which is a membership service that aims at managing authorization information on grid environments in a structured fashion, by creating a central repository of users and their relationship to the Virtual Organization (VO). In VOMS the access control policies are maintained by the corresponding resource owner. The user can only get permissions from the VO administrator. This approach reduces the burden on the provider, who deals only with policies. The authorization credentials management is delegated to a third party and the users are identified by proxy-certificates. However, we consider this proposal inadequate for large-scale collaborations since it is too centralizing.

In [19], Thompson and his colleagues proposed Akenti, which tries to solve two problems: user identification management and access control policy specification by multiple stakeholders. Public key infrastructure solves the first problem. The stakeholders publish a set of certificates with resource usage conditions. The PDP pulls all these certificates to make the authorization decision, thus solving the second problem. Akenti decentralizes policy management for multiple stakeholders, that is, the service owners still have to manage everything, and this can be troublesome on a dynamic collaboration environment.

VOMS and Akenti are focused on modeling who is responsible for specifying the access control policies and authorization credentials management. They are an evolution from the scheme where all the policy specification must be done by the provider organization, but they do not consider usage control concepts, thus requiring external additional controls to achieve controls which are similar to that provided by UCON$_{ABC}$.

VII. Conclusions

Our proposal shows the viability of creating a UCON$_{ABC}$-based authorization framework for BCs, by enabling the service provider to delegate policy writing to a broker that in its turn can delegate to the consumers, thus enhancing the policy management process. It also prevents the broker from trading service usage rights that were not previously delegated by the provider. The usage control model adopted in our proposal supports the needs of a BC environment better than other related works.

The application of an interface for predicate-based policy writing eases the task of the administrator without requiring a framework for a proprietary language. The notification service promotes a better experience to the service consumer, who is warned before any service usage revocations. Also, the notification service is an example of real system implementation of obligations in our proposal.

The implemented prototype shows that it is possible to implement UCON$_{ABC}$ without the need for XACML extensions, thus showing the feasibility of our approach, while using the available technology. Moreover, the prototype evaluation showed that it is possible to integrate the available technologies in a fashion that does not cause unreasonable impact on the overall system performance. We conclude therefore that, although there is space for performance optimizations, our proposal brings a considerable set of qualitative advantages over previous related work.

References

[1] J. Park and R. Sandhu, "The uconabc usage control model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, 2004.

[2] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management, and processing," *ACM Comput. Surv.*, vol. 38, no. 1, p. 3, 2006.

[3] R. Yavatkar, D. Pendarakis, and R. Guerin. (2000) A Framework for Policy-based Admission Control. RFC2753. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2753.txt

[4] R. L. Rivest and B. Lampson. (1996) SDSI - Simple Distributed Security Infrastructure. [Online]. Available: http://people.csail.mit.edu/rivest/sdsi10.html

[5] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. (1999) SPKI Certificate Theory. (rfc2693). RFC2693. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2693.txt

[6] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. (2001) Usage for Policy Provisioning (COPS-PR). RFC3084. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3084.txt

[7] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. (2004) Web Services Architecture. W3C Working Group Note 11. [Online]. Available: http://www.w3.org/TR/ws-arch/wsa.pdf

[8] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. (2007) SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). [Online]. Available: http://www.w3.org/TR/soap12-part1/

[9] K. Lawrence and C. Kaler. (2004) OASIS Web Services Security: SOAP Message Security 1.1. [Online]. Available: http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[10] ——. (2007) OASIS WS-SecurityPolicy 1.2. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[11] (2007) Apache axis2. [Online]. Available: http://ws.apache.org/axis2/

[12] G. Cole. (2006) OASIS Service Provisioning Markup Language (SPML) Version 2. [Online]. Available: http://xml.coverpages.org/SPMLv2-OS.pdf

[13] (2008) OpenSPML 2.0. [Online]. Available: http://www.openspml.org/

[14] (2006) SunXACML 1.2. [Online]. Available: http://sunxacml.sourceforge.net/

[15] S. Godik and T. Moses. (2002) OASIS eXtensible Access Control Markup Language (XACML) 1.1. [Online]. Available: http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf

[16] X. Zhang, M. Nakae, M. J. Covington, and R. Sandhu, "Toward a usage-based security framework for collaborative computing systems," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 1, pp. 1–36, 2008.

[17] F. Martinelli, P. Mori, and A. Vaccarelli, "Towards continuous usage control on grid computational services," in *ICAS-ICNS '05: Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*. Washington, DC, USA: IEEE Computer Society, 2005, p. 82.

[18] R. Alfieri, R. Cecchini, V. Ciaschini, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro, "an authorization system for virtual organizations," in *Proceedings of the 1st European Across Grids Conference, Santiago de Compostela*, 2003, pp. 13–14.

[19] M. R. Thompson, A. Essiari, and S. Mudumbai, "Certificate-based authorization policy in a pki environment," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 566–588, 2003.