Resolving Policy Conflicts - Integrating Policies from Multiple Authors

Kaniz Fatema[1] and David Chadwick[2]

[1] Irish Center for Cloud Computing & Commerce, University College Cork, Ireland
k.fatema@cs.ucc.ie
[2] University of Kent, Canterbury, Kent, UK
D.W.Chadwick@kent.ac.uk

Abstract. In this paper we show that the static conflict resolution strategy of XACML is not always sufficient to satisfy the policy needs of an organisation where multiple parties provide their own individual policies. Different conflict resolution strategies are often required for different situations. Thus combining one or more sets of policies into a single XACML 'super policy' that is evaluated by a single policy decision point (PDP), cannot always provide the correct authorisation decision, due to the static conflict resolution algorithms that have to be built in. We therefore propose a dynamic conflict resolution strategy that chooses different conflict resolution algorithms based on the authorisation request context. The proposed system receives individual and independent policies, as well as conflict resolution rules, from different policy authors, but instead of combining these into one super policy with static conflict resolution rules, each policy is evaluated separately and the conflicts among their authorisation decisions is dynamically resolved using the conflict resolution algorithm that best matches the authorisation decision request. It further combines the obligations of independent policies returning similar decisions which XACML can't do while keeping each author's policy intact.

Keywords: Dynamic policy conflict resolution, XACML, authorization system, multiple policy authors.

1 Introduction

Attempts to protect the privacy of personal data by including policies from the data subject in an access control system are not new [1-3]. These works mainly focus on how to have policies from the data subject and how to enforce them. One issue that is often ignored is that multiple parties may have policies that contribute to the overall decision of whether personal data can be accessed or not. These parties include: the legal authority, the issuer of the data, the subject of the data and as well as the controller (who is currently holding and controlling the flow of the data processing). A strategy that combines all these policies into a single 'super' policy may not work for all situations. In this paper we give such an example. We propose a dynamic

L. Iliadis, M. Papazoglou, and K. Pohl (Eds.): CAiSE 2014 Workshops, LNBIP 178, pp. 310–321, 2014.

conflict resolution strategy that integrates the decisions of the policy decision point(s) (PDPs) evaluating the polices provided by the various parties.

Using XACML [4, 5], due to its static policy / policy set combining algorithm, it is very hard to integrate policies from multiple parties in a dynamic way, in order to always obtain the correct authorisation decision. Our proposed model allows different policy authorities to provide their individual, independent polices and it will integrate the decisions returned by their PDPs in a dynamic way, according to the authorisation decision request.

2 Use Case

By way of a use case scenario, we consider a university which awards degrees and scholarships and maintains a profile for each student/alumnus containing various personal data such as degree certificates, transcripts, and awarded scholarships. For personal data like degree certificates and transcripts, the university may want to deny access to anyone unless the data subject (i.e. the alumnus) has specifically granted access to the requestor in her policy (for example, she can authorise a potential employer to access her degree certificate). For the scholarship awards, the university may want to publish these on its web site for marketing purposes, unless the data subject (i.e. the student) has specifically requested that the public be denied access to it. Since scholarships are usually regarded as an achievement by students most of them will usually like to be honoured in this way, (and the university might also make it a condition of the scholarship that the award can be published except in exceptional circumstances). In the degree certificate case the conflict resolution rule will be grant overrides, since the issuer's (i.e. the university's) policy denies access but the subject's policy may override this with a grant decision. In the scholarship case, the conflict resolution rule will be deny overrides, since the issuer's policy grants access but the subject's policy may deny access. It is not possible to combine into a single policy set the issuer's policy for both resources with a subject's policy for both resources since two different conflict resolution rules are required, whilst XACML only allows one conflict resolution rule to be applied to a set of policies. In order to implement this scenario in a single XACML "super" policy, both the subject's and issuer's policies would need to be dissected into their separate rules for each resource and then combined together per resource with separate conflict resolution rules per resource. Depending upon the number of types and subtypes of resource covered in any policy, this splitting and merging could get very complex. Furthermore it might be envisaged that different conflict resolution rules are needed for different actions or subjects on the same resource, which would make the splitting even more complex. We conclude that in a single organisation, there may be the need for various policy conflict resolution strategies which are not possible to satisfy with one static XACML policy, without sacrificing the integrity of the individual policies provided by the different authors. We therefore propose a solution where each author's policy remains intact and is evaluated as is, but the conflicts between policies are dynamically resolved based on a dynamically determined conflict resolution rule. In this use case

example, this means that if the access request is to read a degree certificate, the conflict resolution rule is grant overrides, but if the access request is to read the scholarship awards, the conflict resolution rule is deny overrides.

3 Related Works

Nicole Dunlop et al. [6] have classified the conflicts of policies into four different categories. They have proposed different strategies for when and how to resolve conflicts. With the Pessimistic Conflict Resolution approach preventive steps are taken to resolve conflicts so that conflicts do not arise. With the Optimistic Conflict Resolution approach different conflict resolving steps are taken when conflicts do arise such as: new rule overrides old rule, assigning explicit weights or priorities to rules and so on. However the strategies are static i.e. the conflict resolution strategy is not chosen at run time.

Chen-hia Ma et al. [7] have defined a way for static and dynamic detection of policy conflicts. For resolving conflicts a *conflict_resolution_policy* is used which has a number of *priority_rules* that actually define different precedences. These *priority_rules* are tried one by one until the conflicts are resolved. In comparison, our proposed system chooses one combining rule based on the request context and uses that to resolve all conflicts.

Various conflict resolution strategies have been discussed in [8-10]. However, all those strategies are static. Mazzoleni et al. [11] argued that XACML policy combining algorithms are not sufficient to integrate policies where there is no centralised control and presented a system to integrate policies for different organisations. Nevertheless, their policy combining algorithm is static.

Apurva Mohan et al. [12] have argued that the static composition may not be suitable for dynamic environments where there is need to adapt the policies dynamically with the environment. They proposed a dynamic conflict resolution strategy that chooses an applicable policy combining algorithm based on a set of environmental attributes. But the problem with their system is that the policy combining algorithm (PCA) rules have to be mutually exclusive. If more than one PCA rule is evaluated to be applicable then an error is generated. This is not suitable where multiple parties provide individual PCAs and ensuring mutually exclusive PCAs in such a dynamic environment would be difficult. The conflict resolution strategy of our system is such that individual PCAs are provided by different authorities and one PCA is selected based on the precedence of the authority.

4 An Overview of the XACML Policy Combining Algorithm

The highest level element of an XACML (v2 and v3) policy is the ***Policy set.*** A ***Policy set*** can be a combination of ***Policy*** or ***Policy sets***. **Policy** is the basic unit used by the PDP (Policy Decision Point) to form an authorisation decision and it can have a set of **Rules**. XACML defines a set of rule-combining- algorithms and

policy-combining-algorithms which form a single authorisation decision from the set of decisions obtained by evaluating either a set of rules or set of policies respectively. The standard combining algorithms of XACML v2 and v3 are defined as:

- Deny-overrides (Ordered and Unordered),
- Permit-overrides (Ordered and Unordered),
- First-applicable and
- Only-one-applicable.

In the case of the Deny-overrides algorithm, if a single Rule or Policy element evaluates to "Deny", then, regardless of the evaluation result of the other Rule or Policy elements, the combined result is "Deny". For the Ordered Deny-Overrides the behaviour of the algorithm is the same except that the order in which the collection of policies is evaluated will match the order as listed in the policy set.

Similarly, in the case of the Permit-overrides algorithm, if a single "Permit" result is encountered, regardless of the evaluation result of the other Rule or Policy elements, the combined result is "Permit" and the obligation attached to the policy or policy set forming the decisions is also returned with the “Permit” decision. For the Ordered Permit-overrides the behaviour of the algorithm is the same except that the order in which the collection of policies is evaluated will match the order as listed in the policy set.

In the case of the “First-applicable” combining algorithm, the first decision encountered by the Rule, Policy or PolicySet element in the list becomes the final decision accompanied by its obligations (if any).

The "Only-one-applicable" policy-combining algorithm only applies to policies and ensures that only one policy or policy set is applicable by virtue of their targets. The result of the combining algorithm is the result of evaluating the single applicable policy or policy set. If more than one policy or policy set is applicable, then the result is "Indeterminate".

In XACML v3 some other combining algorithms are also defined, such as Deny-unless-permit (which returns Deny only if no Permit result is encountered; Indeterminate or NotApplicable will never be a result), Permit-unless-deny (which returns Permit only if no Deny result is encountered; Indeterminate or NotApplicable will never be a result)

In XACMLv2 a policy or policy set may contain one or more obligations. In XACML v3 a rule, policy or policy set may contain one or more obligation expressions which are evaluated to obligations when such a rule, policy or policy set is evaluated. For each combining algorithm the “Obligation” that was attached to the rule, policy or policy set that returned the decision is also returned with that final decision. In both XACMLv2 and XACMLv3 an obligation is passed to the next level of evaluation only if the effect of the rule (for v3), policy or policy set being evaluated matches the values of the FulfillOn attribute of the obligation. An obligation will not be returned to the PEP if the rule (for v3) policy or policy set from which it is drawn is not evaluated.

5 An Overview of Our Policy Combining Strategy

Our authorisation system was described in [13]. Here we provide a brief overview only. Our system receives an independent access control policy and conflict resolution policy from each policy author. Four types of policy author are supported, in decreasing order of precedence: the legal authority, the data issuer, the data subject and the data controller. A conflict resolution policy comprises an ordered set of Conflict Resolution Rules (CRRs), and each CRR comprise a Decision Combining Rule (DCR), plus the rule for which access requests this DCR applies to. The access request is received by the Policy Enforcement Point (PEP) (step 1 of Fig 1). The PEP passes the request to the Master PDP (step2 of Fig 1). The Master PDP determines which DCR applies to the current request by evaluating the CRRs of the various authors, in order (Fig 2).

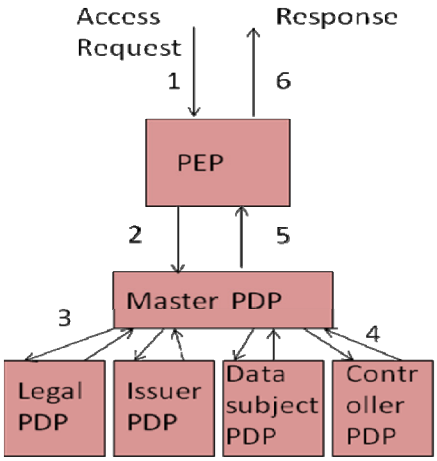

Fig. 1. The authorisation system in a simplified form

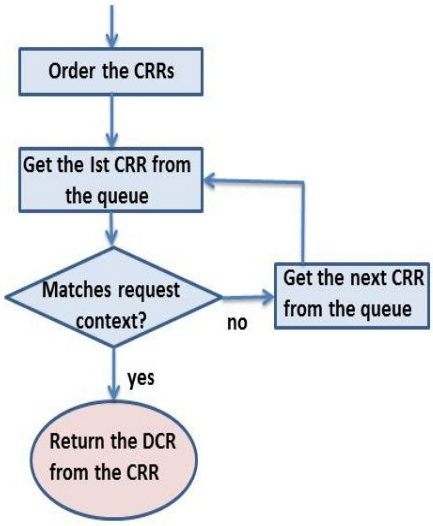

Fig. 2. The process of selecting a Decision Combining Rule (DCR) by the Master PDP

Our implementation currently supports 3 DCRs: FirstApplicable, DenyOverrides and GrantOverrides which have their standard XACML meanings, but in principle any conflict resolution algorithm can be supported.

The Master PDP collects all the CRRs defined by the different policy authors as well as having one default rule (which is configurable). From the request context it knows the data issuer and data subject and so can determine the ordering of the CRRs. It orders the CRRs of the legal authority, data issuer, data subject and data controller sequentially. For the same author the CRRs are ordered according to their times of creation so that the latest CRR always comes first in the author's list.

All the conditions of a CRR need to match the request context for it to be applicable. The CRRs from the ordered CRR queue are tested one by one against the request context. If a CRR's conditions do not match the request context the next CRR from the queue will be tested. The default CRR is placed at the end of CRR queue and it will only be reached when no other CRR's conditions match the request context. If a CRR's conditions match the request context the DCR of that CRR is chosen. After obtaining the DCR the PDPs are called (step 3 of Fig 1) and the Master PDP gathers the responses from them (step 4 of Fig 1). The final result is computed according to the chosen DCR as described below.

If DCR=FirstApplicable the Master PDP calls each subordinate PDP in order and stops processing when the first Grant, Break the Glass (BTG) [14] or Deny decision is obtained. If none of these is obtained, then Indeterminate is returned if at least one PDP returned this, otherwise NotApplicable is returned.

For DenyOverrides and GrantOverrides the Master PDP calls all the subordinate PDPs and combines the decisions using the following semantics:

- DenyOverrides – A Deny result overrides all other results. The precedence of results for deny override is Deny>Indeterminate>BTG>Grant>NotApplicable.
- GrantOverrides – A Grant result overrides all other results. The precedence of results for grant override is Grant>BTG>Indeterminate>Deny>NotApplicable

When a final result returned by the Master PDP is Grant (or Deny) the obligations of all the PDPs returning a Grant (or Deny) result are merged to form the final set of obligations.

In our system the policies of the different authors remain independent of each other and are independently evaluated. This also helps to enforce them in a distributed system, when they are transferred as "sticky policies" to other systems along with the data they control. The receiving system does not need to employ complex processing algorithms in order to create an integrated complex 'super' policy from the received policies and the organisation's own policy; it only needs to start an independent PDP and order the incoming CRRs along with its existing ones.

6 Policy Creation and Integration Strategy Comparison

In this section we shall first look into the strategy that can be taken to convert the previous example use case scenario into policies for our system and then the same into one XACML 'super' policy.

While forming the conflict resolution policy the conflict resolution rules (CRRs) obtained from legislation [15] come first, then come the CRRs from the issuer, then from the data subject and then from the controller and finally the default one.

For the scenario of our use case example the issuer has 2 different CRRs to contribute to the conflict resolution policy. The CRRs of the issuer are:

1. If resource_type=scholarship_info, DCR=DenyOverrides
2. If resource_type=degree_certificate, DCR=GrantOverrides.

We assume the first CRR was written after the second CRR, as they are ordered according to their times of creation.

The issuer has an access control policy saying "for resource_type = scholarship_info, effect = Permit and for resource_type = degree_certificate, effect = Deny". These apply to all accessors i.e. the public.

Suppose that the data subject is embarrassed to have been given a hardship assistance scholarship. He has a policy saying "for resource_type = scholarship_info, scholarship_type = hardship assistance, effect = Deny".

The data subject can have any conflict resolution rule and that CRR will come after the CRR of the issuer on the ordered list of CRRs and so it will only be evaluated if there is no CRR from the law or the issuer matching the request context. However, as we have already seen, the issuer does have two CRRs that will match.So the CRP will have the following CRRs in order:

1. CRR from the law: none.
2. CRR from the issuer (CRR no. 1. If resource_type = scholarship_info, DCR=DenyOverrides; CRR no. 2. If resource_type = degree_certificate, DCR=GrantOverrides).
3. CRR from the data subject: <anything>.
4. CRR from the controller: none.

Suppose that a request to view a student's scholarship information has arrived. The Master PDP will evaluate the ordered list of CRRs. The law has no CRR regarding this [15]. The next CRRs on the list are the CRRs from Issuer. The CRR no. 1 of issuer will match the request context for which the DCR=DenyOverrides. So the DCR=DenyOverrides is chosen by the Master PDP. The Master PDP now calls the independent PDPs of the law, issuer, data subject and controller. In this case the legal PDP always returns NotApplicable, and the issuer PDP always returns Grant. The data subject PDP will return Deny when the scholarship_type=hardship assistance otherwise it will return NotApplicable. There is no controller PDP for this use case scenario as the controller and issuer are the same for this use case (i.e. the university). So in case of scholarship_type= hardship assistance the final result will be Deny (according to the DenyOverrides DCR). For other types of scholarship information the DCR will remain the same i.e. DenyOverrides. The legal PDP returns NotApplicable, the issuer PDP returns Grant, the data subject's PDP returns NotApplicable. Therefore, the final decision becomes Grant.

Suppose that a request to view an alumni's degree certificate has arrived. The Master PDP will evaluate the ordered list of CRRs. The law has no CRR regarding this [15]. The next CRRs are the CRRs from the issuer. Issuer CRR no 2 will match the request context which has a DCR=GrantOverrides. So it chooses the DCR=GrantOverrides. The Master PDP now calls the independent PDPs of the law, issuer, data subject and controller. In this case the legal PDP returns NotApplicable, the issuer PDP returns Deny. The data subject PDP will return NotApplicable. So according to the GrantOverrides DCR the final result will be Deny (unless the alumni specifically grants the access in his/ her policy).

Now if we try to combine the policies from the law, issuer, and data subject under one XACML PDP policy the policies from the different authors can't remain independent anymore. In order to make sure that the law always has the highest priority the top policy combining algorithm can't be DenyOverrides or GrantOverrides as the decision of the legal policy can then be overridden by the other authors. To give the legal policy the highest priority the top combining algorithm needs to be first applicable with the legal policy coming first. To implement our example policies the policies from the issuer and data subject need to be combined under the DenyOverrides algorithm for one case (for resource_type=scholarship_info) and under the GrantOverrides algorithm for the other case (resource_type= degree_certificate). This means it requires splitting the policies of the different authors and then combining them together into PolicySets based on the resource type. This may require manual interpretation and implementation depending on the needs of the organization and the complexity of the legal, issuer and subject policies. The integration of polices from different authorities into one 'super' policy may not be impossible but it is not an easy task and might be difficult or impossible to fully automate. It will certainly damage the integrity of the individual policies by splitting them up and may make it more difficult to prove compliance with data protection legislation. In contrast, our system keeps the policies written by the different authors integral and independent of each other, and makes it easier to show compliance with the law and to transfer them to remote systems as sticky policies.

7 Integration of Obligations

Each XACML policy document contains one Policy or PolicySet element as a root XML tag. A PolicySet can contain a number of Policies or PolicySets. A Policy represents a single access control policy, expressed through a set of Rules. Each Policy or PolicySet or Rule (for v3 only) element can define Obligations which can contain a number of Obligation elements. Each Obligation element has an Obligation ID and a FulfillOn attribute. XACML's obligation combination strategy can be viewed as a vertical procedure where the Obligations of a contained Rule/Policy/PolicySet are combined with the Obligations of the containing Policy/PolicySet. An Obligation associated with a Rule or Policy is returned with a decision only if the effect of the Rule or Policy being evaluated is the same as the FulfillOn attribute of the Obligation. If the Policy is contained in a PolicySet, the Obligations associated with the PolicySet having a FulfillOn attribute value matching the effect of the PolicySet are combined with the returned Obligations of the contained Policy. For example, if policyset A has obligation o1 and it contains policy A with obligation o2 and policy B with obligation o3 then the final obligations returned could be o1 and o2 or o1 and o3 (assuming they all have the same FulfillOn attribute) depending upon the combining algorithm (see below) and the order in which policy A and B are evaluated. This procedure continues recursively.

The limitations of the XACML obligations combining algorithm is that if a rule, policy or policy set is not evaluated then no obligations from them are returned to the

PEP [5 (p82), 4 (p87)]. With XACML's GrantOverrides / DenyOverrides combining algorithms as soon as a Grant/ Deny decision is encountered the Grant / Deny is returned without evaluating the rest of the policies. Also with the FirstApplicable combining algorithm as soon as a decision (Grant/Deny) is obtained it is returned. This strategy of obligation combination may result in losing important obligations that ought to be returned. For example, if the controller's policy said that every time a Grant decision is returned, there is an obligation to "log the request" whilst the data subject's policy had a similar requirement that when a Grant decision is returned there is an obligation to "e-mail the data subject"; then if these policies are combined in a single XACML PolicySet with a GrantOverrides combining algorithm, then one of these obligations will always be lost. In light of the above one can see that the integration of polices from different authorities into one 'super' XACML policy is more complex than simply splitting the policies of the different authors and then combining them together into PolicySets based on the resource type, as this may result in the loss of obligations..

In contrast to XACML, our system's policy evaluation and obligation combination strategy can be viewed as a horizontal procedure. In our system for both GrantOverrides and DenyOverrides all the PDPs are evaluated and the final decision is chosen based on the DCA. The obligations that are returned by all the PDPs that have a decision equal to the final decision are combined. For example if the controller PDP returned a decision Grant with an obligation to "log the request" and the data subject's PDP returned a decision Grant with an obligation to "e-mail the data subject" and the final decision is Grant; then in our system the returned obligations will be the combination of the obligations attached to the Grant decisions. If the policies are implemented in a single XACML PDP with either a GrantOverrides or DenyOverrides combining algorithm, the returned obligation(s) will only be the obligation(s) attached to the policy that was encountered first and that contributed to the final decision.

8 Implementation and Testing

The authorisation system is implemented as a standalone web service using JAVA which runs in a servlet container (apache Tomcat). The present implementation can choose a DCR dynamically from a fixed set of configured CRRs. Each CRR is a policy either written in XACML or PERMIS [16] and the DCR is obtained as an obligation. The correct functioning of the system was validated using over 100 test cases based on different use case scenarios [18].

It can be argued that increasing the number of PDPs will unduly affect the system's performance, so we ran some performance tests to determine the scale of this. The authorization infrastructure was installed in a single machine running Ubuntu 10.04 whose configuration was: duel core processors each with cpu speed = 2993.589 MHz; cache=2048 KB; and 2GB total memory. The configuration of the client machine that made the authorisation decision requests was: Duel core processor with 2.53 GHz CPU speed, 2.98 GB memory and running Windows XP. The client software was

SOAPUI [17]. The client was operating across a local area network. Two series of tests were performed. The first set tested the reduction in performance for an increasing number of PDPs running inside the authorization service. The second set tested the reduction in performance as the number of rules in a policy is increased for a single PDP inside the authorisation service.

Table 1. Time (in ms) to make an authorization decision for different number of PDPs

Test	Mean	Std Dev	% Discarded	Mean PDPi – Mean PDPi-1
1 PDP	5.27	0.51	4.07	
2 PDPs	6.34	0.74	2.47	1.07
3 PDPs	14.82	1.47	2	8.48
4 PDPs	22.64	1.53	2	7.82
5 PDPs	30.37	1.9	1.4	7.73
6 PDPs	38.30	1.99	1.6	7.93
7 PDPs	46.47	2.32	1.2	8.17
8 PDPs	54.26	2.28	2.4	7.79
9 PDPs	62.51	2.59	0.8	8.25
10 PDPs	69.61	2.55	1.2	7.1

In this first series of tests the authorization server was configured with an increasing number of policies/PDPs, each containing 1 rule. In the first test the authorization server only had 1 policy configured into it (the legal PDP with 1 rule). In the second test the authorization server was configured with the legal policy and the data controller's policy/PDP (with 1 rule). In the third test the authorization server had 3 policies/PDPs: the data subject's sticky policy (with 1 rule), and the legal and controller's configured policies. In the subsequent tests an additional sticky policy PDP with 1 rule was added.

In each case we measured the time taken for an authorization decision to be made when the client asked to read a data record, and a grant result was obtained. This necessitated all configured policies being interrogated and the Master PDP determining the combining rule (GrantOverrides was chosen) the overall decision and set of returned obligations. The tests were run 100 times and results lying more than two standard deviations from the mean where discarded as outliers. The results are shown in Table 1.

From the results one can observe that there is a decision making overhead of approximately 8 ms per additional sticky policy PDP. The reason the 1st PDP was only 5ms and the 2nd PDP only added 1 ms is that they are both built in PDPs and not sticky policy PDPs (which are added dynamically).

For the second set of performance test the authorisation system is configured using only one PDP. The number of rules is increased at each test and the time to get a

response for a request is measured. Each test was run 100 times and the outliers (> 2 standard deviations) were discarded. The results are shown in Table 2.

Table 2. Time (in ms) to make an authorization decision for different number of Rules

Test	Mean	Std Dev	% Discarded	Additional Time per rule
1 Rule	5.9	0.7	5.2	5.9
10 Rules	15.58	0.9	5.2	1.07
100 Rules	69.26	1.59	4.2	0.64
1000 Rules	581.19	8.12	0.98	0.58

We can see that as the number of rules increases, the time taken to reach a decision reduces per rule until we reach a steady state of approximately 0.58ms per rule (for our implementation and configuration). For our particular implementation and configuration, the effect of adding a new sticky policy PDP to the authorisation server running in a PC, in the worst case is approximately equal to adding 14 new rules to an existing PDP, and in the best case only 8 rules.

Taking our earlier use case example, policies for the law, controller/issuer and data subject contain 15, 2 and 1 rules respectively. The complete legal policy that enforces the European Data Protection Legislation can be found in [15]. The combined 'super' policy contains 18 rules. Using tables 1 and 2 we can calculate that the time taken to evaluate the 3 separate policies (using 2 configured PDPs and 1 sticky PDP), for our particular implementation and configuration, would be approximately 31 ms (20+2.5+8.5) whereas the time taken to evaluate the 'super' policy would be approximately 23 ms. So whilst there is a performance overhead of approximately one third in this use case, this should be offset against the complexity needed to create a combined 'super' policy which always returns the correct decisions and obligations.

9 Conclusions

In this paper the policy combining strategy of our system is compared with the static policy combining approach of XACML. Our strategy allows the policy engine to dynamically choose different combing algorithms based on the request context while allowing the policies written by different authors to remain separate, integral and independent of each other. The separation of policies facilitates the easy integration of "sticky" policies into the system allowing personal data to be transferred to a different domain along with its policies. Furthermore, our system allows the return of combined obligations written by different authors, which the current XACML policy combining strategy is unable to do when keeping the policies of individual authors intact. However, further work is still needed to see how easy it might be to integrate the polices from different authors into one 'super' policy, as an alternative to the approach presented here.

References

1. Karjoth, G., Schunter, M., Waidner, M.: Privacy-enabled services for enterprises. In: 13th International Workshop on Database and Expert Systems Applications, pp. 483–487. IEEE Computer Society, Washington, DC (2002)
2. Mont, M.C.: Dealing with Privacy Obligations: Important Aspects and Technical Approaches. In: International Conference on Trust and Privacy in Digital Business, Zaragoza (2004)
3. Ardagna, C.A., Bussard, L., Vimercati, S.D.C., Neven, G., Paraboschi, S., Pedrini, E., Preiss, F.-S., Raggett, D., Samarati, P., Trabelsi, S., Verdicchio, M.: PrimeLifePolicy Language. In: Workshop on Access Control Application Scenarios, W3C 2009(2009)
4. OASIS XACML 2.0. eXtensible Access Control Markup Language (XACML)Version 2.0, http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=xacml#XACML20 (October 2005)
5. OASIS XACML 3.0. eXtensible Access Control Markup Language (XACML) Version 3.0, http://docs.oasisopen.org/xacml/3.0/xacml-3.0-corespec-en.html (April 16, 2009)
6. Dunlop, N., Indulska, J., Raymond, K.: Methods for Conflict Resolution in Policy-Based Management Systems. In: Proceedings of Seventh International Enterprise Distributed Object Computing Conference, pp. 98–109. EEE press, New York (2003)
7. Ma, C., Lu, G., Qiu, J.: Conflict detection and resolution for authorization policies in workflow systems. Journal of Zhejiang University Science A 10, 1082 (2009)
8. Russello, G., Dong, C., Dulay, N.: Authorisation and conflict resolution for Hierarchical Domains. In: Eight IEEE International Workshop on Policies for Distributed Systems and Networks (2007)
9. Syukur, E., Loke, S.W., Stanski, P.: Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments. In: Policy Management for Web Workshop in Conjunction with WWW2005 Conference, Chiba, Japan, May 10-14 (2005)
10. Lupu, E.C., Sloman, M.: Conflicts in Policy-Based Distributed Systems Management. IEEE Transactions on Software Engineering, 852–869 (1999)
11. Masoumzadeh, M.A., Jalili, R.: Conflict detection and resolution in context-aware authorization. In: 21st International Conference on Advanced Information Networking and Applications Workshops (2007)
12. Mohan, A., Blough, D.M.: An Attribute-based Authorization Policy Framework with Dynamic conflict Resolution. In: IDtrust, Gaithersburg, MD (2010)
13. Chadwick, D.W., Fatema, K.: A Privacy Preserving Authorisation System for the Cloud. Journal of Computer and System Sciences, vol 78(5), 1359–1373 (2012)
14. Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zhao, G., Chilro, R., Antunes, L.: How to securely break into RBAC: the BTG-RBAC model. In: Annual Computer Security Applications Conference, Honolulu, Hawaii, pp. 23–23 (2009)
15. Fatema, K., Chadwick, D.W., Van Alsenoy, B.: Extracting Access Control and Conflict Resolution Policies from European Data Protection Law. In: Camenisch, J., Crispo, B., Fischer-Hübner, S., Leenes, R., Russello, G. (eds.) Privacy and Identity 2011. IFIP AICT, vol. 375, pp. 59–72. Springer, Heidelberg (2012)
16. Chadwick, D., Zhao, G., Otenko, S., Laborde, R., Su, L., Nguyen, T.A.: PERMIS: A modular authorization infrastructure. Concurrency and Computation: Practice and Experience 11(20), 1341–1357 (2008)
17. SOAP UI, http://www.soapui.org
18. Fatema, K.: Adding Privacy Protection to Policy Based Authorisation Systems:PhD thesis, University of Kent, UK (to appear, 2014)