eGovernment service security policy: obligation conflict resolution in XACMLv3

Ibrahim Yonis Omar, Romain Laborde, Ahmad Samer Wazan, François Barrère, Abdelmalek Benzekri
Institut de Recherche en Informatique de Toulouse
University Paul Sabatier
Toulouse, France
{Yonis, Romain.Laborde, Ahmad-Samer.Wazan, Francois.Barrere, Abdelmalek.Benzekri}@irit.fr

Abstract— Today, many governments tend to propose e-services to their citizens. However, implementing an eGovernment environment shall face up to several security challenges including integrating security requirements coming from multiple stakeholders. In this article, we analyze the conflicts that can occur between eGovernment security requirements. Since these security requirements can contain both authorizations and obligations, we cover these two aspects. Then, we propose a new conflict resolution algorithm that handles conflicts between authorizations as well as obligations. This work has been implemented in XACMLv3.

***Keywords*— eGovernment; Access control; Obligations; conflict; XACMLv3**

I. INTRODUCTION

Towards a reduction process of gaps between user expectations and public services, public administration tends to use ICT in order to offer efficient services. This paradigm is known as electronic Government (eGovernment) [1] and can be classified according to different target areas as:

- Government-to-Government (G2G), also known as e-administration, refers to electronic collaboration between different government agencies,
- Government-to-Citizen (G2C), is the process that electronically provides on-demand and personalized public services to citizens,
- Government-to-Business (G2B), sets up online relationship between government and the business sector in order to interactively provide information on regulations, advice, and procedures.

This modernization of relations with a government takes an interest because it is generally offered in a centralized way – with a one-shop portal: all eGovernment services are available in one place and exposed from a common portal [2].

eGovernment services are classified depending on its levels of paperless known as maturity level. Designing models of maturity levels has been the subject of several studies [3]. Although the number of phases differs from one model to another, all the models are based on four main phases to measure the maturity of a system of e-government. These main level starts from level 1, with a simple informational website to level 4 with an advanced shared services between public administration.

However, implementing eGovernment must address several challenges [4]. Among them the way to design and write a security policy remains complex [5] because it must consider different High Level Security Requirements (HLSRs) given by stakeholders [6]. Security policy of an eGovernment service must comply at the same time with HLSRs expressed by law issuers (Li), Executive governance (Eg) and government departments (Gd).

Our proposed research is done in the context of the Djiboutian eGovernment Cloud Community (eGCC) which aim to implement a G2G infrastructure: two main issues have been highlighted when the common security policy of eGCC was analyzed. .

First, according to its own area of occupation, each stakeholder expresses its HLSR using an expression model that may differ from other stakeholders. DAC (Discretionary Access Control) - – MAC (Mandatory Access Control) model; - RBAC (Role Based Access Control) [7] [8][9] are some example of such models. While specific constraints must be considered, it remains that the policy within eGCC must adopt unified way of expression.

This issue was discussed in [10] where we proposed the usage of ABAC (attributes based access control) with XACMLv3 [11] standard. A common policy-based language for eGovernment was presented in order to express multiple specific constraints (thanks to ABAC) and applied our approach to an open source Cloud Computing solution – OpenStack [12].

The second issue is related to the consistency of HLSRs. HLSRs can contain both authorizations (*permission on resources with defined conditions*) and obligations (*duties to execute*). The common security policy for eGovernment service is established by combining all stakeholders' HLSRs. Each HLSR, written in XACMLv3 is delivered by stakeholders and contain specific constraints. As a consequence a simple HLSR combining may result into conflicts and inconsistencies.

Many works have studied conflicts between authorizations. E.g., XACMLv3 provides twelve authorization conflict resolution algorithms. However, much less researches have explored conflicts between obligations and how to manage these conflicts. As consequence, XACMLv3 doesn't include any obligation conflict resolution. In this article, we analyze the obligation conflict management issue in the context of

eGovernment that involves multiple stakeholders. Also, we propose an obligation conflict resolution algorithm that we implemented in XACMLv3.

The rest of the article is structured as follows. In section II, we present security management issues in eGovernment. In section III, we present XACMLv3 and its capability to express eGovernment policy security requirements. In section IV, we introduce our approach to enhance XACMLv3 with an algorithm for eGovernment obligation conflict management. In section V, we list some related work. Finally, we draw our conclusion and perspectives in section VI.

II. EGOVERNMENT SECURITY MANAGEMENT

The security of eGovernment services is governed by a set of HLSRs that we classify according to the institution source: legal, governance and business.

Legal HLSRs are expressed from legislation and concerns compliance to legal texts applied to information and data collected by public administration. Data sensitivity in the context of eGovernment requires regulation. To prevent abuse of data usage in administrative procedures and thus establish trust between the users and the e-Government service, a number of laws have been voted and must be respected.

Governance HLSRs is expressed from executives and ensure the proper organization of security within eGovernment organizations IS. It corresponds to the general policy of government on eGovernment and expresses requirements on how eGovernment is implemented.

Business HLSRs expressed from organization are essentially dealing with business needs. They are driven by the profession's needs of ministerial departments.

Given the multiple policies with its HLSR, security compliance of eGovernment services to those policies may be subject to conflicts. We propose to address these conflicts by prioritizing them according to their sources. This priority is based on the natural hierarchy characterizing the machinery of government

Legal HLSRs should have greater weight than those dictated by the executives and business stakeholders. Executive HLSRs should have more weight than those of business.

A weight is given to each HLSR according to its source (i.e., legal, governance or business). Based on that weight, HLSRs are prioritized to resolve the conflicts. Thus, the order relation that states is:

Legal Policies (LP) > Executive Policies (EP) > Business Policies (BP).

To highlight HLSRs consistency challenge, let us consider, for instance, the Tax Income Public Agency (TIPA) that provides tax information and services. In order to enhance its service treatments, TIPA decide to offer an eGovernment services. At the first stage of this migration, TIPA will only provide informational eGovernment service (Maturity level 1). Thus TIPA creates Virtual Machines (VM) which hosts a web server within a virtual data center (VDC), offered by an eGovernment Cloud provider (eGCp).

Resources of TIPA is governed by a set of policies with multiple HLSRs from 1) Law, 2) Executive and 3) Business. Enforcement of policies must follow the order of the predefined hierarchy: Legal (LP)> Executives (EP) > Business (BP). Let's consider that the policy of TIPA is the following:

- LP1: Identifiable data collected from users shall not be transferred or used in any other purpose without the prior consent of its user.
- LP2: Regulation *requires* encrypting eGovernment services resources.
- BP1: Resource encryption is required for eGovernment service classified as Maturity Level 3 [3] and above only.
- BP2: In case of cyber attack any executive administrative task of eGovernment system should not be available except for Chief Information Security Officer (CISO).
- EP1: In case of cyber attacks to eGovernment system, access to the system is forbidden until constitution of a team by ministerial Decree.

Clearly, the policy of TIPA entails the handling of different HLSRs coming from different sources and the preservation of authorizations and obligations orders. Thus, the following criteria must be filled:

Criterion 1 — There must have policies hierarchy management systems.

Criterion 2 — Security management system has to apply both authorizations and obligations policies.

Criterion 3 — Regardless of policy selection order, enforcement must respect the predefined hierarchy.

In order to handle policies whose expressions (e.g., RBAC, MAC or DAC) and sources (Law, Executives and Business) are different, we have selected the language XACMLv3 to implement our solution. The extensibility of this language permits us to enhance its capability to the eGovernment context.

III. XACMLV3 AND EGOVERNMENT POLICY SECURITY HLSR

We briefly present in this section the XACMLv3 standard and how it could meet the constraint of HLSRs.

XACML (eXtensible Access Control Markup Language) version 3 is an XML-based specification for access control that has been standardized by OASIS [8]. XACMLv3 describes an architecture, an attribute-based access control policy language and a request/response language.

The XACMLv3 policy language is used to describe general access control constraints in terms of constraints on attributes. Specifically, attributes could be any characteristics of any category such as the subject, the resource, the action, or the

environment in which the access request is made. Attributes have an identifier, which is a Uniform Resource Name (URN), and a data type also identified by a URN. Considering attributes makes the language very flexible. Moreover, XACMLv3 language is natively extensible. A XACMLv3 policy is composed of:

- A *target* element which is a first filter for searching the applicable policy
- A set of *obligation expressions* that are instantiated when a matching request is processed. PEPs must enforce obligations.
- A set of *advice expressions* that are instantiated when a matching request is processed. Advice is similar in its form to an obligation. However, PEPs may or may not enforce advice.
- A set of *rules* that are expressions to determine if a request is denied or permitted. A *rule* contains a *target* and may include *obligations* and *advice* specific to this rule.
- Policies can be grouped in *policy sets*.

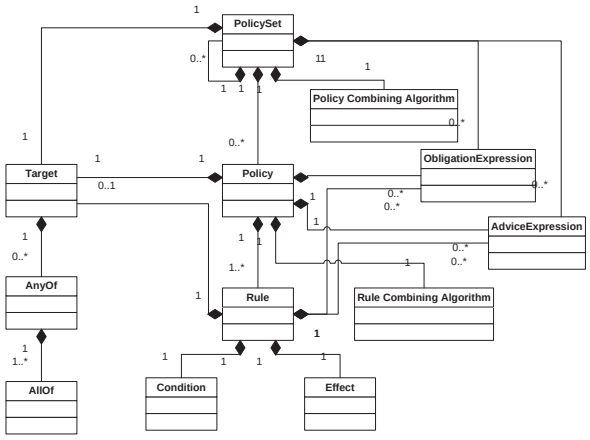

Figure 1. The XACMLv3 policy language mode [11]

The architecture of XACMLv3 consists mainly in two management components: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). The PEP is the guard of the resources. It intercepts the request expressed in the native application format, translates it into the XACMLv3 request format and sends it to the PDP. The PDP is the "brain" of the system. Once it receives a request from a PEP, it looks at its XACMLv3 policy for matching rules. Each rule leads to a specific decision, which is a triplet (permit/deny, set of obligations, set of advice). If only one rule match then the decision is applied .If the request matches two or more rules, the PDP builds a unique decision by applying the rule combining algorithms and the policy combining algorithms. This unique decision is then returned to the PEP that enforces it in the actual system.

XACMLv3 includes a set of predefined *policy/rule combining algorithm*s, used to resolve the eventual conflicts in the authorizations:

- Deny overrides: This algorithm combines decisions of policies / rules so that if any decision is *Deny*, then that decision is applied.
- Permit-overrides: This algorithm does the same work as the above algorithm, but in this case *Permit* decisions are the dominant ones.
- First applicable: This algorithm applies the first decision (Deny or Permit) found and returns the first match as result.
- Only one applicable: This algorithm is used only for combining policies. It cannot be used to combine rules.
- Deny unless permit: The algorithm result will be Deny unless an explicit Permit Decision is found.
- Permit unless Deny: Same as the above algorithm except default result will be Permit unless explicit Deny is found.

These algorithms also exist in *ordered* mode where policy, *policy set* and *rules* are considered in the order in which they are defined. Thus, prior establishment of hierarchy can be fulfilled with XACMLv3 predefined combining algorithms.

Although XACMLv3 supports natively authorization conflict management with its combining algorithms, these algorithms don't take into account the obligations. In the scenario proposed above, as VM creation is authorized for TIPA, BP1 and LP2 HLSRs obligations conflict. LP2 (law) requires all VMs must be encrypted on creation. BP1 (business) does not claim such encryption as the service provided is informational (maturity level1) and is not an advanced one (maturity level3). As XACMLv3 does not have an obligation conflict management algorithm, such obligations are together sent to PEP which generates a problem of applicability for PEP or obligation Service unless formal handling methodology.

IV. A NEW CONFLICT RESOLUTION ALGORITHM THAT CONSIDERS OBLIGATIONS

In this section we present our new algorithm to resolve eGovernment obligations conflict issues in XACMLv3. A conflict resolution algorithm for obligations consists of two parts: conflict detection and conflict resolution.

A. Detection of Conflict

We represent a XACML rule $R \in RULES$ as a triple $(cond_R, effect_R, obligations_R)$ where $cond_R \in COND$ is a Boolean expression with free variables, $effect_R \in$ {Permit, Deny} and $obligations_R \in \mathbb{P}$(OBLIGATIONS) is a set of obligation expressions with free variables.

Evaluating a rule R for a given request *req* can be achieved by executing three tasks:

1. The bounding of the free variables of the condition using the request attributes. We note the bounded condition with bound($cond_R$,req).
2. The interpretation of the bounded condition that provides a Boolean value (the condition matches or not). We represent it by interpret (bound($cond_R$,req)).
3. The bounding of the free variables of the obligations using the request attributes. We note the bounded obligations with bound($obligation_R$,req).

Detecting a conflict for a given request *req* can then be formalized as follows:

$\exists$ (R1, R2) $\in$ RULES^2,

R1= *($cond_{R1}$, $effect_{R1}$, $obligations_{R1}$), R2 = ($cond_{R2}$, $effect_{R2}$, $obligations_{R2}$)* ,

interpret(bound($cond_{R1}$,req)) = $\top$ ⅄

interpret(bound($cond_{R2}$,req)) = $\top$

where at least one of the following two conditions is true:

Condition 1) $effect_{R1} \neq effect_{R2}$

Condition 2) $\exists$ (o_{R1}, o_{R2}) $\in$ $obligations_{R1} \times obligations_{R2}$,
conflict(interpret(bound(o_{R1},req),
interpret(bound(o_{R2},req)))

Detecting a conflict must be performed at the decision stage, i.e. by the PDP, in order to provide a unique decision to the PEP. When rules don't include obligations, the detection is easy since the PDP natively performs interpret(bound($cond_R$,req)) and evaluating *condition 1* requires only to compare two values (Permit/Deny).

However, evaluating *condition 2* is not as simple as *condition 1*. In fact, obligations conflict detection requires the analysis of the semantic of the obligations and a dynamic detection of possible conflict is not obvious. E.g., what is the result of interpret (bound(o_{R1},req))? How to detect the execution of an obligation is conflicting with another one? Such issue is pointed with BP1 against LP2. Thus, for obligation conflict detection, we use the follow manual discovery algorithm to handle semantic means of obligation action.

Algorithms 1 ObligationConflictsDetect()

1 Let p *be the parameter of an eGovernment service resource* Obli *represent, for each i, 1≤ i ≤, a set of obligations applied to* p *and* P_{Obl} *possible obligation conflict.*

2 P_{Obl} *= if* $\exists$(OblBP,OblLP) x p(TIPA)

3 if $\exists$ P_{Obli} ⁄\ OblBP ¬ OblLP ➔ OblC

End Algorithms

Our algorithms detect obligation conflict, if two or more obligations are designed towards the same parameters of single resource to the same eGovernment services (TIPA). For TIPA, for instance, we identify the set of obligation HLSR applied to it. Afterwards, we identify whether any of the obligations potentially conflict with each other. If positive conflict match, we select conflicting obligations.

B. Obligation conflict resolution

To detect and resolve obligation conflicts in XACMLv3, we propose to extend the PDP with obligation inconsistency management algorithms. We adopt answer set programming (ASP), a form of declarative programming [13], to formally represent our model. ASP is based on the stable model semantics of propositional logic programming and allows non-monotonic reasoning. Syntactically, ASP is closed to Prolog. However, instead of asking a question and using inference to find the solution like in Prolog, ASP grounds the variables and computes stable models (for more details [14]).

We recall quickly some basics on the ASP syntax. Rules are of the form "`h :- b.`" where `h` is the head and `b` is the body. It can be understood as if predicate `b` is true in an answer then `h` is also true in the answer. When the rule has no body, for example "`h.`" then `h` is a fact and must be in all the answers. When the rule has no head, for example "`:- b.`", the rule is a constraint and means that `b` must not be true in any answers. Finally, it is also possible to specify choice. For example, the following rule "`{h1;h2}:-b.`" can be understood as if `b` is true then there can be an answer where `h1` is true and another one where `h2` is true.

```
#show finalDecision/1.
%issuer/1 is the source of the obligation.
issuer(law).
issuer(executive).
issuer(business).
%superior/2 defines that LAW > EXECUTIVE > BUSINESS.
superior(issuer(law), issuer(executive)).
superior(issuer(executive), issuer(business)).

%%%%%%%%%%%
% GUESS
%%%%%%%%%%%

%conflict/2 defines explicitly conflicts between two obligations. This relation is a symetric.
conflict(obligation(OBL1), obligation(OBL2)) :- conflict(obligation(OBL2), obligation(OBL1)).

% candidate solution : Any obligation can be in the final decision
{finalDecision(obligation(OBL)): decision(obligation(OBL), issuer(_))}.

% If 2 obligations are in conflict then choose the one from the requirement with higher priority
finalDecision(obligation(OBL1)) :- conflict(obligation(OBL1), obligation(OBL2)), decision(obligation(OBL1),issuer(F1)), decision(obligation(OBL2),issuer(F2)), superior(issuer(F1), issuer(F2)).
% Otherwise choose one
1{finalDecision(obligation(OBL1));finalDecision(obligation(OBL2))} :- conflict(obligation(OBL1), obligation(OBL2)), decision(obligation(OBL1),_), decision(obligation(OBL2),_).

%An obligation that has no depency issue can be in the final decision
finalDecision(obligation(OBL)) :- not conflict(obligation(OBL), obligation(_)), decision(obligation(OBL),_), not dependencyIssue(obligation(OBL)).

%%%%%%%%%%%
% CHECK
%%%%%%%%%%%

%Two conflicting obligations cannot be in the final decision
:- conflict(obligation(OBL1), obligation(OBL2)), finalDecision(obligation(OBL1)), finalDecision(obligation(OBL2)).

%There is a depency issue when an obligation depends on another one that is not in the final decision
dependencyIssue(obligation(OBL1)) :- dependsOn(obligation(OBL1),obligation(OBL2)), not finalDecision(obligation(OBL2)).

%An obligation that has some depency issue cannot be in the final decision
:- finalDecision(obligation(OBL)), dependencyIssue(obligation(OBL)).
```

Figure 2. Our Answer Set Program for resolving conflicts.

Since determining matching XACMLv3 rules and detecting authorization effects conflict is already done by any XACMLv3 PDP, we focus only on obligation conflict resolution. Thus, we consider that a set of rules matches a specific XACMLv3 request and these rules have the same effects. However, some rules contain obligations.

For our implementation, we used clingo 4 [15]. We followed the guess and check methodology [14] which consists in:

1) **Guess** : Create candidate solutions to the problem
2) **Check**: Check with rules/constraints whether a candidate solution is valid or not.

Thus, based on a set of obligations as input, we generate candidate solutions (**Figure 2**). If conflicts exist, we choose the obligation with the higher priority calculated based on the issuer (Law > Executive > Business). We then check if there is no conflicting obligations or functional dependency issue in a candidate solution. Conflicting obligations and obligation dependencies have to be manually expressed using predicates *conflict*/2 and *dependsOn*/2 (Figure 3). This means that all obligations applied on eGovernment services must be predetermined and analyzed to produce this data.

```
%%%%%%%%%%
% Example of an initial knowledge database
%%%%%%%%%%
obligation(obl1).
obligation(obl2).
obligation(obl3).
obligation(obl4).
% there is a conflict between obl2 and obl3
conflict(obligation(obl2), obligation(obl3)).

% dependsOn/2 represents the dependency between obligations.
% obl4 depends on obl3
dependsOn(obligation(obl4),obligation(obl3)).
```

Figure 3. Example of an initial knowledge database for conflict resolution.

```
% translated from the matching XACML rules
decision(obligation(obl1), issuer(law)).
decision(obligation(obl2), issuer(law)).
decision(obligation(obl3), issuer(executive)).
decision(obligation(obl4), issuer(executive)).
```

Figure 4. Predicates translated from matching XACMLv3 rules.

Finally, when the PDP has to take a decision, it translates the candidate obligation into predicate *decision*/2. **Figure 4** gives an example where obligations *obl1* and *obl2* are coming from law HLSRs and obligation *obl3* and *obl4* from executive HLSRs. After being processed, the final decision calculated by the ASP program is cancelled *obl3* (in conflict with *obl2* that has higher priority) and *obl4* (it depends on *obl3*).

C. Obligation enforcement planning

We complete the obligation conflict resolution with an obligation enforcement planner to ensure that obligations are executed in the right order (compliance to our criterion3). Indeed, unwanted side effects may arise if obligations are applied in any arbitrary order. We propose to specify known side effects using predicate *before*/2 meaning that an obligation must be applied before another one (Figure 5). Using the methodology Guess&Check, we build the following obligation enforcement planner. For example, if a final decision consists in applying obligations *obl1*, *obl2*, *obl3*, *obl4*, planner proposes several solutions like the following sequences <*obl1*, *obl3*, *obl4*, *obl2*> or <*obl3*, *obl1*, *obl2*, *obl4*>

```
#show enforce/2.
%%%%%%%%%%
% Example of an initial knowledge database
%%%%%%%%%%

% before/2 represents that fact that an obligation must be executed before another one.
before(obligation(obl1),obligation(obl2)).
before(obligation(obl3),obligation(obl2)).
before(obligation(obl5), obligation(obl2)).

%%%%%%%%%%
% GUESS
%%%%%%%%%%

step(X):- X=1..N, N=#count{OBL:obligation(OBL)}.

{enforce(obligation(OBL),step(T)): obligation(OBL), step(T)}.

%%%%%%%%%%
% CHECK
%%%%%%%%%%

% Any obligation of the XACML decision must be enforced.
:- not enforce(obligation(OBL),step(_)), obligation(OBL).

% An obligation that has been defined to be executed before another one cannot be
enforced after.
:- enforce(OBL1,step(T1)), enforce(OBL2, step(T2)), T1 < T2, before(OBL2, OBL1).

% Only one obligation can be enforce at some time T.
:- enforce(obligation(OBL1),step(T)), enforce(obligation(OBL2), step(T)), OBL1 != OBL2.

% An obligation is enforced only once.
:- enforce(obligation(OBL),step(T1)), enforce(obligation(OBL), step(T2)), T1 != T2.
```

Figure 5. Our obligation enforcement planner

V. RELATED WORK

eGovernment security policy.

Security in eGovernment is largely acknowledged as a challenge [16] [17][18] . As part of a European project, Lambrinoudakis et al. [19] propose PKI-based security policy for eGovernment services. According to eGovernment service, its level of paperless and users involved, a risk level, which can be low, medium or high, is labeled. Based on this level, they define security requirements. They then deal with these levels of requirements with a PKI-based security policy.

Drogkaris et al. [20] have acknowledged privacy concerns in eGovernment security policy with user preference involvement. They propose a Privacy Controller Agent (PCA), an engine that manages privacy enforcement in eGovernment. They underline existing of various rules in the service provider privacy policy document. For Drogkaris et al., conflict can occur between service provider and user preferences. Although this approach is dealing with the security concerns (privacy aspect) of modern eGovernment with centralized one stop shop portal, it ignores the potential conflict between various inherited rules expressed by services provider policies.

A. XACMLv3 conflicts analysis

Hwang et al. [21] propose a tool that generates the XACML-represented policy and check the consistency of these policies both statically and automatically. Verification focuses on policy coherence, specifically whether the authorization result is produced as expected or not.

To detect inconsistencies and conflicting XACML-represented policy, Martin and Logrippo [22] use Alloy [23] a first order logic model checking tools. They represent XACML element as a logical model and translated into Alloy in order to

detect inconsistency of policies. Inconsistency is produced when "two rules return two different decisions (permit and deny) in a context of a specific request".

Fisler et al. [24] propose verification and validation policy tool Margrave (ref) for XACML-represented policy. With verifier component integrated into margrave, different possible decisions from XACML policy are represented as a form of diagram and are verified to detect the eventual conflict between decisions.

Mohan et al. [25] highlight the problematic of authorization in taxonomy-based biomedical databases. They propose strategies and algorithms to detect policies conflict and potential inference attacks resulting from how policies are formulated. Their proposition is implemented in XACML.

Martin et Xie [26] determine the gap between result of decision and expected behavior of policies written in XACML by generating request on policies and use the responses as input to a tool using machine learning algorithms. As an output these tool generate behaviors of policies by listing, "inferred properties that may not be true for all requests but are true for most requests in order to highlight possible special case requests.

However, all these works have uniquely focused on the management of inconsistency and conflicts of the authorization side of XACML. We have shown that conflicts in policies can also be produced because of opposed obligations to carry out. Since XACMLv3 takes into account the obligation representation and due to the lack of obligation conflict management in the current works, we have proposed an algorithm to detect and resolve the eventual conflicts produced by different opposed obligations in XACMLv3 policies in the eGovernment context.

VI. CONCLUSION

Managing the security policies in context of eGovernment entails the construction of a security management system that allows to: 1) Combine different policies expressed by different models, 2) Handle conflict decisions produced at policy and rule levels (using combining algorithms), and 3) Handle conflict obligations.

In a previous work [10], we addressed the first point. In the current work, we have handled the second and third points. Specifically, we have exploited the existing capabilities of XACMLv3 to address the second point. However, since XACMLv3 does not support natively obligation conflict management, we propose an obligation conflict management algorithm that can be executed by a PDP. Also, we have implemented an obligation planner intended to preserve obligation orders.

Our contribution didn't consider the real security state of different stakeholders. Indeed, the higher the maturity levels of eGovernment services are, the more resources are available on the Internet. Also, advanced high level eGovernment services require involvement of multiple stakeholders. Thus, the security of these resources becomes an essential matter to consider. However, due to divergence state of security preparation of stakeholders, defining formal security responsibility of stakeholders towards advanced eGovernment service is not obvious. How we can determine security responsibility of involved stakeholders? Thus, defining a scale of security competency levels of stakeholders may help to preserve the security of the advanced high-level eGovernment services. We believe that such levels can be considered as conditions to delimit the scope of each stakeholder. These issues constitute the main activity that we are conducting currently.

VII. REFERENCE

[1] M. J. Moon, "The evolution of e-government among municipalities: rhetoric or reality?," *Public Adm. Rev.*, vol. 62, no. 4, pp. 424–433, 2002.

[2] A. Tat-Kei Ho, "Reinventing local governments and the e-government initiative," *Public Adm. Rev.*, vol. 62, no. 4, pp. 434–444, 2002.

[3] K. Layne and J. Lee, "Developing fully functional E-government: A four stage model," *Gov. Inf. Q.*, vol. 18, no. 2, pp. 122–136, 2001.

[4] D. S. Jones and B. Crowe, *Transformation not automation: The e-government challenge*. Demos, 2001.

[5] M. Al-Sebie and Z. Irani, "Technical and organisational challenges facing transactional e-government systems: an empirical study," *Electron. Gov. Int. J.*, vol. 2, no. 3, pp. 247–276, 2005.

[6] J. Rowley, "eGovernment stakeholders—Who are they and what do they want?," *Int. J. Inf. Manag.*, vol. 31, no. 1, pp. 53–62, 2011.

[7] G.-J. Ahn, "Discretionary Access Control," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Springer US, 2009, pp. 864–866.

[8] P. Samarati and S. D. C. Di Vimercati, "Access control: Policies, models, and mechanisms," *Lect. Notes Comput. Sci.*, pp. 137–196, 2001.

[9] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," *ArXiv Prepr. ArXiv09032171*, 2009.

[10] I. Y. Omar, R. Laborde, A. S. Wazan, F. Barrere, and A. Benzekri, "G-Cloud on Openstack: Adressing access control and regulation requirements," in *International Symposium on,Networks, Computers and Communications (ISNCC)*, 2015, pp. 1–6

[11] "eXtensible Access Control Markup Language (XACML) Version 3.0." [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html. [Accessed: 08-Feb-2016].

[12] "Documentation — OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/Documentation. [Accessed: 08-Feb-2016].

[13] V. Lifschitz, "What Is Answer Set Programming?.," in *AAAI*, 2008, vol. 8, pp. 1594–1597.

[14] T. Eiter, G. Ianni, and T. Krennwallner, *Answer set programming: A primer*. Springer, 2009.

[15] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Clingo= asp+ control: Extended report," Technical report, University of Potsdam, 2014.

[16] Z. Zhou and C. Hu, "Study on the e-government security risk management," *Int. J. Comput. Sci. Netw. Secur.*, vol. 8, no. 5, pp. 208–213, 2008.

[17] C. Kalloniatis, E. Kavakli, and S. Gritzalis, "Security requirements engineering for e-government applications: analysis of current frameworks," in *Electronic Government*, Springer, 2004, pp. 66–71.

[18] R. Breu, M. Hafner, B. Weber, and A. Novak, "Model driven security for inter-organizational workflows in e-government," in *E-Government: Towards Electronic Democracy*, Springer, 2005, pp. 122–133.

[19] C. Lambrinoudakis, S. Gritzalis, F. Dridi, and G. Pernul, "Security requirements for e-government services: a methodological approach for developing a common PKI-based security policy," *Comput. Commun.*, vol. 26, no. 16, pp. 1873–1883, 2003.

[20] P. Drogkaris, S. Gritzalis, C. Kalloniatis, and C. Lambrinoudakis, "A Hierarchical Multitier Approach for Privacy Policies in eGovernment Environments," *Future Internet*, vol. 7, no. 4, pp. 500–515, 2015.

[21] J. Hwang, T. Xie, V. Hu, and M. Altunay, "ACPT: A tool for modeling and verifying access control policies," in *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, 2010, pp. 40–43.

[22] M. Mankai and L. Logrippo, "Access control policies: Modeling

and validation," in *5th NOTERE Conference (Nouvelles Technologies de la Répartition)*, 2005, pp. 85–91.
[23] D. Jackson, "Alloy 3.0 reference manual," *Softw. Des. Group*, 2004.
[24] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 196–205.
[25] A. Mohan, D. M. Blough, T. Kurc, A. Post, and J. Saltz, "Detection of conflicts and inconsistencies in taxonomy-based authorization policies," in *Bioinformatics and Biomedicine (BIBM), 2011 IEEE International Conference on*, 2011, pp. 590–594.
[26] E. Martin and T. Xie, "Inferring access-control policy properties via machine learning," in *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*, 2006, p. 4–pp.