Authorisation Policy Federation in Heterogeneous Multi-Cloud Environments

Ioram S. Sette, *Federal University of Pernambuco (UFPE) and Recife Center for Advanced Studies and Systems (CESAR),*
David W. Chadwick, *University of Kent at Canterbury (UKC),*
and Carlos A. G. Ferraz, *Federal University of Pernambuco (UFPE)*

***Abstract*—Current Infrastructure as a Service (IaaS) cloud platforms have their own authorisation system, containing different access control policies and models. Clients with accounts in multiple cloud providers struggle to manage their rules in order to provide a homogeneous access control experience to users. This work proposes a solution: an Authorisation Policy Federation (APF) of heterogeneous cloud accounts. These federated accounts share a centrally managed policy written in Disjunctive Normal Form (DNF) using a cloud-independent ontology. This shared abstract policy can be translated to local cloud formats, and back again. Prototypes were implemented for OpenStack and Amazon Web Services (AWS) cloud formats, and rules were successfully translated with a Level of Semantic Equivalence (LSE) higher than 80%.**

***Index Terms*—multi-cloud, heterogeneous clouds, access control, authorisation policy federation.**

I. Introduction

SECURITY is a significant barrier to the adoption of cloud technologies when they are vendor-provided infrastructures shared by multiple tenants [1]. This problem is aggravated in multi-cloud environments, a recent development to avoid vendor lock-in and provide service diversity and price competition, amongst other advantages [2]. In this scenario, each cloud may use a different platform. If the user's services and resources are spread across them, this increases the complexity of management and of ensuring the security of the user's information.

Authentication in multi-cloud environments can be facilitated by the Identity Federations, which provide facilities such as Single Sign-On (SSO) and strong or multi-factor authentication. Users authenticate once via their Identity Provider (IdP) to gain access to multiple heterogeneous Cloud Service Providers (CSPs). Users are identified by a unique set of attributes issued by their IdP, giving them a homogeneous identification and authentication experience.

Homogeneous authorisation is also a desire for multi-cloud environments. One approach could be an "Authorisation Federation", in which a central Policy Decision Point (PDP) allows or denies service access requests. This Authorisation as a Service (AaaS) model [3] means that a single homogeneous policy can be used by all CSPs, which leads to ease of policy management, and less chance of policy conflicts. However, it suffers from a number of disadvantages: it is a single point of failure, each CSP has to be modified to call the central PDP, and poor performance can result due to network latency.

Alternative approaches such as fully decentralised solutions [4], [5] may be complex for security administrators to manage, in order to provide a consistent set of authorisation rules. How can we define homogeneous policies when each PDP could use a different policy language with different functionality and access control model? How can we synchronise the policy between CSPs? The advantages of distributed PDPs are clear: the CSPs do not need to be modified as they use their existing PDPs, and the PDPs are already tailored to their respective CSPs for performance, functionality, manageability etc. Providing the disadvantages can be overcome we believe this is a preferable solution.

Our solution, which we call Authorisation Policy Federation (APF), allows clients of heterogeneous multi-clouds to define a homogeneous authorisation policy that applies equally to all users across all clouds. A central Policy Administration Point (PAP), called the Federated Authorisation Policy Management Service (FAPManS), stores the abstract authorisation policy using a cloud-independent ontology. Translation/mapping engines (adaptors) convert the abstract policy into cloud dependent policies (and vice-versa), so that they can be enforced using the existing cloud authorisation mechanisms. A publish-subscribe infrastructure is responsible for keeping the abstract and cloud dependent policies synchronised. Policy management occurs in the background with offline updates, and therefore does not affect the performance of online policy decisions. The performance of the authorisation itself is determined by the local cloud PDP engines and is not impacted by our solution.

Prototypes of FAPManS and adaptors for OpenStack and AWS IaaS cloud platforms were designed and implemented. Policies were successfully translated from their local format to the abstract format and back again with Level of Semantic Equivalences (LSEs) higher than 80%. Rules that could not be translated were either cloud or tenant-specific, or indicated that mapping rules were missing in the prototype adaptors.

The rest of this paper is structured as follows. Section 2 provides a literature review. Section 3 presents the requirements and architecture of our proposed solution. Section 4 has the implementation. Section 5 validates our solution. Section 6 concludes and indicates where further research is still required.

II. Literature Review

According to Celesti et al. [6], cloud environments will evolve from "monolithic" islands of cloud services to "horizontal federations", in which clouds cooperate to increase their capacities and reduce cost. In [2], environments with multiple clouds are classified according to their level of interoperability, ranging from isolated CSPs on multi-cloud environments to cloud federations.

Anastasi et al. [7] propose a Usage Control (UCON) system for Contrail (http://contrail-project.eu) - a broker-based solution to interconnect clouds. The authorisation decision takes place in the broker's security service, using eXtensible Access Control Mark-up Language (XACML)-based policies. The clouds interoperate with this system through Attribute Managers (AMs), which transfer fresh values of mutable attributes to the central Policy Information Point (PIP). Tests on prototypes showed an acceptable scalability for realistic setups. Another model for federated access control using brokers is defined in [8]. This solution is proposed for heterogeneous multi-provider multi-cloud environments under the context of the Intercloud Architecture Framework (ICAF). Federated Identity Management establishes trust relationships among the broker and the federated clouds. Authorisation is performed by domain specific access control engines (PDP) using XACML policies.

Service Access and Manipulation Operation Specification (SAMOS) [9] is a semantic-aware multi-cloud orchestration solution based on ontologies. Although its architecture includes a central authorisation manager, it is not clear how the authorisation policies are translated and configured inside the cloud platforms.

Tang et al. [3] formalised and extended the Multi-Tenancy Authorisation System (MTAS), which is a centralised solution based on hierarchical Role-Based Access Control (RBAC) in the Platform as a Service (PaaS) layer that serves the Software as a Service (SaaS) layer. This solution provides a central PAP and PDP AaaS, using XACML policies. Bernabe et al. [10] propose a semantic-aware version of MTAS, which uses an ontology for IaaS resources defined in Web Ontology Language (OWL). Policy translation into XACML is performed at authorisation time, prior to the authorisation decision. The system performance is not reported, but we anticipate it will be slow.

Ngo et al. [4] propose a distributed Multi-Tenant Attribute-Based Access Control (MT-ABAC) infrastructure for multi-provider heterogeneous environments. The Dynamic Access Control Infrastructure (DACI) provides dynamic trust establishment for entities in IaaS clouds. However, cloud providers need to integrate their authorisation engines to DACI. Almutairi et al. [5] propose a distributed access control architecture for a multi-tenant multi-cloud environment based on a decentralised PAP integrated with an identity federation. It provides semantic and contextual constraints to protect services and resources.

In contrast, our work proposes a distributed authorisation architecture for heterogeneous multi-cloud environments, with abstract authorisation policies being defined in a central PAP using a common language, and translated to each PDP using cloud specific adaptors. Table I compares related works with ours.

TABLE I
SUMMARY OF RELATED WORKS

Work	Het.	Model	Policy lang.	PDP	PAP
[7]	No	ABAC	XACML	Centr'ed*	Centr'ed*
[8]	Yes	ABAC	XACML	Centr'ed*	Centr'ed*
[9]	No	not mentioned	Ontology-based	Centr'ed*	Centr'ed*
[3]	No	RBAC	XACML	Centr'ed	Centr'ed
[10]	Yes	**	XACML	Centr'ed	Centr'ed
[4]	***	MT-ABAC	XACML	Distr'ed	Distr'ed
[5]	Yes	RBAC	XML-based	Distr'ed	Distr'ed
Ours	Yes	**	Ontology-based and DNF	Distr'ed	Centr'ed

*) Broker model.
**) Many Access Control (AC) models are supported, such as Attribute-Based Access Control (ABAC) and (hierarchical) RBAC
***) The architecture supports heterogeneity, but the clouds must implement the DACI engine.

III. HOMOGENEOUS ACCESS CONTROL IN HETEROGENEOUS MULTI-CLOUD ENVIRONMENTS

The requirements for our solution evolved from working in cloud security over several years and discovering where implementors and users were having problems - the pain points. In particular we worked on OpenStack, one of the most popular open source cloud implementations. We spoke with many users and developers whilst performing our research. The requirements were developed in an organic informal way, rather than by following a strict software engineering method such as questionnaires or brainstorming.

A. Requirements

1) **Support SSO and multiple Identity Federation protocols.** SSO and Identity Federation allow users to access multiple services from different IaaS cloud providers by using a single set of authentication credentials [11]. However, there is no ubiquitous federation protocol, so any solution must be capable of supporting any set of federation protocols, e.g. OpenID Connect (http://openid.net/connect/), Security Assertion Mark-up Language (SAML) (http://saml.xml.org) and Application Bridging for Federated Access Beyond web (ABFAB) [12]. The authors already introduced protocol independent federated identity management to OpenStack [13], and this remains a key requirement. After the user is authenticated, the IdP provides a set of user identity attributes to the CSPs.
2) **Provide equivalent authorisation policies on accounts of multiple heterogeneous IaaS clouds.** It is essential for homogeneous authorisation that the policy rules are equivalent on each heterogeneous CSP account.
3) **Provide simple policy manageability.** Security administrators must be able to define a single common authorisation policy that can be used in multiple heterogeneous IaaS CSPs. Managing multiple policies in different languages via different interfaces is a complex task that must be avoided.
4) **Be scalable and available.** The entire authorisation process must be scalable. The evaluation of authorised access to a CSP can be performed multiple times for a

single cloud job, for instance, as each different service is accessed (database, network, cpu etc.). Even access to a single service might require several access control decisions, e.g. if UCON is used. This is different from authentication, which usually takes place just once for a set of accesses. Therefore, the performance of authorisation decisions must not decrease when new cloud accounts share a common policy. Furthermore, the authorisation process must not be a unique point of failure. The IaaS CSPs must be available and authorising properly regardless of any external component of the solution.

5) **Be easy for adoption.** The proposed solution must require minimal intervention on existing CSPs to facilitate its adoption. For instance, the replacement of their authorisation engines is not desired, neither is forcing them to be compatible with a different policy language, e.g. XACML.
6) **Provide quick policy synchronisation.** The solution must provide an efficient synchronisation mechanism to quickly update the policies of all cloud accounts when the common policy is modified.

B. *Architecture*

We achieve homogeneous access control in multi-cloud environments with an architecture composed of the existing heterogeneous PDPs provided by the CSPs and a centralised global PAP, called the FAPManS. This architecture is instantiated for two APFs (α and β) on three CSPs in Figure 1.

Users authenticate through a federated IdP in order to gain access to multiple clouds (requirement 1). IdPs are responsible for authentication of their users, and also for user attribute assignment that is subsequently used in the authorisation. Note that as part of federated authentication, each CSP maps the IdP assigned attributes into the locally equivalent ones [13]. Administrators also use an IdP to access the FAPManS interface.

Security administrators create and manage common homogeneous policies using the FAPManS interface (requirements 2 and 3). These policies are defined in a common language, using the DNF. DNF is very useful for representing a union of independent rules that each grant access (the “deny all except” type). Deny rules are only needed as a subset of a grant rule e.g. allow all staff except the janitor. Since it is a normal form, any policy that can be represented as logical expressions can be converted to DNF. We believe this is the case for all policy formats used by current IaaS cloud platforms.

The normalized aspect of DNF makes it easy to compare different policies written in different languages. This is not true when using other policy languages, such as XACML, which is recursive, non-normalised, and one policy can be defined in multiple ways.

A DNF Policy is composed of “rules” combined with the “OR” logical operator. These “rules” are in turn composed of “conditions” combined with the “AND” operator. Conditions are the basic element of DNF policies, and they represent a single comparison of an attribute and an expected value. Attributes can be a characteristic of a subject, an action, a

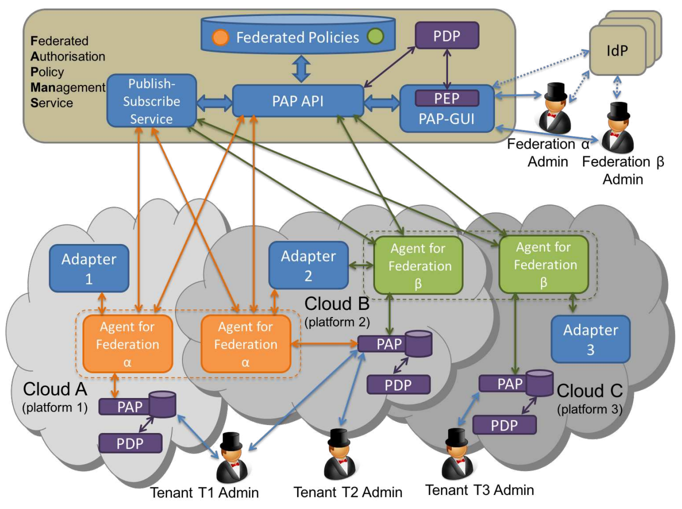

Fig. 1. Example instantiation of the architecture.

resource or the environment. Common attributes, comparison operators and values are defined in an ontology for IaaS cloud elements [14].

Members of an APF are tenants of federated heterogeneous cloud providers that want to enforce globally defined authorisation policies on their accounts. Each global abstract authorisation policy is translated to and from the multiple policy languages used in the different cloud technologies of the tenant APF members via Adaptors.

Each cloud runs an instance of the Adaptor, that can translate DNF policies into their cloud specific equivalents and vice versa. This allows the original cloud authorisation engines to remain unchanged.

Agents are clients of the publish-subscribe synchronisation mechanism, which each tenant must execute in their APF member accounts (requirement 6). Agents are notified when an abstract policy is updated, and they are responsible for retrieving the updated rules from FAPManS, calling the adaptor to translate them into the local syntax and semantics, check that no conflict exist, and to update the local policy in the local PAP. It is the tenant administrator’s responsibility to activate this, using the CSP’s existing mechanism. Conversely, when a local policy is updated, the agent is responsible for verifying if it conflicts with the global policy, and if not, calling the adaptor to translate it to the abstract policy and then updating FAPManS. When an agent detects any conflict it notifies the tenant administrator to take the proper mitigating action before any updates are applied to FAPManS.

When a cloud tenant joins an APF, the rules in the common policy must be enforceable in their account. The agent is able to validate compliance by calling the adaptor. Additionally, tenant administrators are able to create local rules, valid only under the scope of their local accounts, and these will not be translated into the abstract policy (although they will be copied to/from FAPManS). Importantly, these local rules must not conflict with the common federated ones, i.e. they can not override the decision of any rule from the common policy, but they can supplement them. For example, a common policy rule might grant staff from any organisation in the federation access to a local resource, whilst a local rule might additionally grant students from the local organisation access to the same resource. However, a local rule would not be able to deny access to staff from one or more organisations in the federation. As in a political federation, APF members are autonomous but they must obey the federal rules.

The original cloud AC mechanisms are not modified in our solution. This makes it easy for adoption (requirement 5). Local policies can have a mix of federated and non federated rules, providing the latter do not conflict with the former.

As seen in Figure 1, the FAPManS architecture comprises a policy repository, a PAP-Application Programming Interface (API) that allows access to policies, a PAP-Graphical User Interface (GUI) that allows human administrators to manage their policies via the PAP-API, a PDP that grants access to authorised administrators and agents, and a publish-subscribe service that notifies each federated member when the abstract policy is updated.

Regarding the scalability (requirement 4) of FAPManS, since it operates only in background mode, multiple instances of it can be deployed in order to handle a large number of APFs. Moreover, the distributed nature of the architecture, as seen in Figure 1, makes it scalable and available to suit many different scenarios.

In the Figure 1 scenario, APF α is composed of two accounts of a cloud tenant T_1 in heterogeneous clouds A and B. T_1 shares the same set of rules between its different accounts, therefore all its rules are stored and managed in FAPManS. On the other hand, APF β is formed by two different cloud tenants T_2 and T_3, who have accounts in clouds B and C, respectively. They have their local authorisation policies already stored in clouds B and C. Since they now have a common project, they want to share some resources stored in both of their cloud accounts. Their shared authorisation rules will be stored in FAPManS, and depending upon their local PDPs, their local rules likewise.

IV. Implementation

The FAPManS module and two adaptors were implemented as a proof of concept. The source code for these modules is publicly available at https://github.com/ioram7/{cfas,os_adaptor,aws_adaptor}.

All modules were implemented as Representational State Transfer (REST) APIs in Python using the Django REST Framework (http://www.django-rest-framework.org/) and a MySQL (https://www.mysql.com) database. Python is the programming language used in the OpenStack project, and there is also a powerful library called Python Electronic Design Automation (PyEDA) (https://pyeda.readthedocs.org/en/latest/), which can convert logical expressions to their DNF equivalent - a very useful function for our application.

A. FAPManS

FAPManS stores DNF policies in a relational database and provides an API and a GUI for policy management. The API allows administrators to create, read, update and delete abstract policies and also individual rules. Conditions are reused in multiple rules for space saving. Therefore, they are automatically created when policies or rules are defined. They cannot be updated or deleted as this could cause undesirable side effects. Their components (attributes, operators and values) may contain sensitive information, like names of secret unreleased projects or products. Consequently, APF administrators are only allowed to list the conditions referenced by their own rules. Policies and rules returned in the API calls are in a defined JavaScript Object Notation (JSON) format.

The FAPManS implementation supports policies with hierarchical attributes. For example, if the definition of attribute “role” says that a “professor” is superior to a “student”, and there is a rule saying that students can “read” certain files, then PDP engines that support hierarchical attributes will automatically authorise “professors” to “read” these files. Cloud PDPs that do not support hierarchical attributes can signal this, and FAPManS will automatically expand the set of rules, creating the additional one(s) needed to grant “professors” access.

Another feature implemented in the FAPManS GUI/API allows administrators to “search” for policy rules that match some conditions. This allows administrators to search for which roles are needed to perform a specific action, or which actions a certain role can undertake, important functionality currently missing from OpenStack. The search criteria allows multiple conditions combined by the operators “or” or “and” indicating, respectively, that “any” or “all” conditions must match.

B. Adaptors

Two adaptor prototypes were implemented to validate the solution, one for AWS and one for OpenStack. Policy translation comprises syntactic and semantic translation. The former is responsible for transforming the logical expression of a local policy into DNF JSON format, or vice-versa. The latter is responsible for mapping the attributes, operators and values in the conditions to semantically equivalent elements in the opposite language (i.e. abstract ontology to/from local policy language terms).

OpenStack policies are defined in files called “policy.json”, one for each cloud service (e.g. Keystone, Nova etc.). Rules are grouped by the OpenStack API calls, which usually contains a service, an action and a resource (e.g. identity:get_user or compute:update). Syntax translations splits these rules into multiple conditions like: “resource_service = identity” and “action = get” and “resource_type = user”; or “resource_service = compute” and “action = update” and “resource_type = vm” (which is implicit for a compute resource). Each of these API calls contains the on subject conditions that determine who is allowed to perform the action. An empty subject condition means that any authenticated user can perform that action. In the syntactic translation, rules containing multiple subject conditions (e.g. “role: admin or is_admin”) are split into separate sets of AND rules in the DNF. The adaptor needs to combine back into one API rule when the policy is translated back to the local syntax.

AWS policies are also defined in a proprietary JSON format. They comprise “user-based” and “resource-based” policies, attached to users and resources respectively. An AWS policy is composed of independent statements (rules) combined by the operator “OR”. Differently from OpenStack rules, which are all implicitly permit, AWS rules may be “allow” or “deny”. Statements with the explicit “deny” effect have precedence over the “allow” ones. This means that if one rule says “role=professors, action=read, file=A, effect=allow” and another one that says “role=professor, name=David, action=read, file=A, effect=deny”, the second one must prevail for David. Consequently the implemented prototype first collects together all the rules attached to both users and resources before performing syntax translation. Then, “allow” and “deny” rules are combined together with logical “AND”, the effect is discarded, and the “deny” rules are negated, e.g. “(role=professor, action=read, file=A) AND NOT (role=professor, name=David, action=read, file=A)”. Further logical simplification then takes place. Whilst this transformation maintains the policy’s original semantics, the current AWS adaptor implementation discards the deny rules, meaning that if an administrator subsequently creates a DNF rule that allows a subject to perform an action that was originally denied, e.g. ”name=David, action=read, file=A” , the original deny rule will not be considered. As a future enhancement, FAPManS should store the deny rules so that they can be applied to subsequent DNF policy updates.

Semantic translation is responsible for mapping condition elements from specific cloud terms to abstract ones defined in the ontology, and vice-versa. Both adaptors implemented these mapping rules as entries in tables of a relational database. A simple ontology for IaaS clouds was also defined in OWL. For example, an OpenStack condition “resource_type = compute” may be mapped to “resource.type = VM” in the ontology. The adaptor flags any condition attribute or operator that could not be translated as ”cloud-X specific”. This is stored in FAPManS so that all agents can decide if the rule should be applied to their local policy or not.

C. Agents

Since a complete agent was not implemented, API calls to FAPManS and the adaptors were manually activated through the curl (https://curl.haxx.se) application.

V. Validation

As our objective was to provide homogeneous policies on heterogeneous IaaS clouds, its success relies significantly on the accuracy of the policy translations.

The following tests were used to validate our implementation. Rules from default OpenStack Keystone and Nova policies as provided in the open source releases were translated from OpenStack to DNF/ontology in step 1. In step 2, some examples of policy rules in AWS format from the Elastic Compute Cloud (EC2) User Guide (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ExamplePolicies_EC2.html) were translated from AWS to DNF/ontology. Rules in DNF/ontology resulting from step 1 were translated back to OpenStack in step 3, and to AWS in step 4.

TABLE II
LSE for the four validation scenarios.

Step	Description	Initial Number of Rules	Intermediary Rules in DNF syntax and local semantics	Number of Rules Fully translated (final format)	LSE
1	OpenStack to DNF	34 (OpenStack)	**39**	**33** (DNF syntax and ontology)	84.6%
2	AWS to DNF	3 (AWS)	**6**	**5** (DNF syntax and ontology)	83.3%
3	DNF to Open-Stack	**39** (33 ontology, 6 OpenStack)	**39**	34 (OpenStack syntax and semantics)	100%
4	DNF to AWS	39 (**33** ontology, 6 OpenStack)	**33**	33 (AWS syntax and semantics)	84.6%

In order to measure the success rate of adaptor translation, the metric “Level of Semantic Equivalence (LSE)” was defined as

$$L_{SE} = \frac{Rules_{Translated}}{Rules_{All}}$$

$Rules_{Translated}$ is the number of DNF rules that could be translated by the adaptor and $Rules_{All}$ is the total number of DNF rules in a policy (including cloud specific ones).

A cloud specific rule cannot be translated if its destination cloud uses a different cloud technology, or any of its conditions’ elements does not have a mapping rule defined for it in the adaptor/ontology. The latter could be because the APF administrator forgot to define the element in the ontology, or because the element is specific to a cloud technology and does not have a global meaning.

The results of these validation scenarios, are presented in Table II. All LSEs are above 80%, showing that most of the rules could be successfully translated. However, it is important to analyse the rules that could not be translated, in order to determine the cause, since these rules will contain important access control specifications.

In step 1, 34 local OpenStack rules were syntactically translated into 39 DNF Rules, still using OpenStack semantics. Six of these rules could not be semantically translated because some of the terms were not defined in the ontology (e.g. Keystone’s actions “list_groups_for_user” and “check_user_in_group”) or mapping rules were missing in the adaptor (e.g. Nova’s project_id should be mapped to the ontology’s tenant.id, but the mapping rule was not defined). If these missing ontology elements are deemed to be common to all the APF’s clouds, they can easily be added to the Ontology. Otherwise, they can be kept as OpenStack specific rules. The missing mapping rule can be easily added to the adaptor.

In step 2, three AWS rules were syntactically translated into six DNF Rules using local AWS terms. One of these six rules could not be translated into the ontology because the following condition had no equivalence: `''StringEquals'': ''ec2:ResourceTag/purpose'': ''test''`. This rule should either be rewritten using equivalent ontology terms (e.g. resource.owner.group="test"), or remain an AWS cloud specific rule.

Step 3 translated the 39 DNF rules from step 1 (in ontology and OpenStack semantics) back to OpenStack format. As expected, the LSE was 100%.

Step 4 translated the same 39 DNF rules from step 1 to AWS format. The 33 rules defined in ontological terms were successfully translated to AWS. Six OpenStack specific rules obviously could not be translated because they are not in the ontology, and do not have an equivalence in the AWS language. These six rules include the actions "list_groups_for_user and check_user_in_group". So, they were discarded by the AWS (pseudo)agent.

Measuring the performance of the translation was not performed in this validation, since normally it is not a time-critical part of the access control. Policy administration, which includes policy management and translation, happens prior to access control decisions being made. The authorisation enforcement and decision processes are the responsibility of the cloud providers. Since their PDPs have not been modified, our solution does not add any performance overhead to them. However, performance could be an issue if the policy administrator noticed a vulnerability in the abstract policy that could allow an attacker to penetrate the CSP, and wanted the policy update to be activated as soon as possible. Consequently we propose to measure the performance as future work.

We could not find other research that validated the translation process in terms of semantic equivalence, so we cannot compare our solution to theirs. Other researchers have measured the performance of their policy translations, but this was usually because the translation was part of the decision making process.

VI. Conclusion

This paper has presented a solution to enable the enforcement of homogeneous authorisation policies across multiple heterogeneous IaaS clouds. Authorisation Policy Federations (APFs) allow policies to be defined and stored in a common ontology in DNF, and managed from a central PAP, named FAPManS. Adaptors are responsible for translating abstract policies to local cloud-specific formats, and vice-versa, preserving their semantics.

Prototypes of FAPManS and adaptors for OpenStack and AWS cloud platforms were implemented to validate the solution. These adaptors could translate basic policies with a Level of Semantic Equivalence (LSE) greater than 80%. Security administrators can use the LSE to determine how closely their existing policies match the abstract terms defined by the APF. Rules that cannot be translated can be analysed to determine if they are cloud-specific, or if the APF's ontology should be expanded to incorporate them.

As future work, the common policy format should be improved to preserve the semantics of the explicit deny rules. The proposed architecture could also be validated on other cloud types besides IaaS, such as PaaS or SaaS. Currently the mapping rules are hardcoded into the adaptors, but these could be read from the ontology by adding extra tables to FAPManS.

References

[1] N. Gonzalez, C. Miers, F. Redígolo, M. Simplício, T. Carvalho, M. Näslund, and M. Pourzandi, "A quantitative analysis of current security concerns and solutions for cloud computing," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, no. 1, p. 11, 2012. [Online]. Available: http://dx.doi.org/10.1186/2192-113X-1-11

[2] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 7:1–7:47, May 2014. [Online]. Available: http://doi.acm.org/10.1145/2593512

[3] B. Tang, R. Sandhu, and Q. Li, "Multi-tenancy authorization models for collaborative cloud services," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, May 2013, pp. 132–138.

[4] C. Ngo, Y. Demchenko, and C. de Laat, "Multi-tenant attribute-based access control for cloud infrastructure services," *Journal of Information Security and Applications*, vol. 27–28, pp. 65 – 84, 2016, special Issues on Security and Privacy in Cloud Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212615000654

[5] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, and A. Ghafoor, "A distributed access control architecture for cloud computing," *Software, IEEE*, vol. 29, no. 2, pp. 36–44, March 2012.

[6] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Three-phase cross-cloud federation model: The cloud sso authentication," in *Proceedings of the 2010 Second International Conference on Advances in Future Internet*, ser. AFIN '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 94–101. [Online]. Available: http://dx.doi.org/10.1109/AFIN.2010.23

[7] G. Anastasi, E. Carlini, M. Coppola, P. Dazzi, A. Lazouski, F. Martinelli, G. Mancini, and P. Mori, "Usage control in cloud federations," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, March 2014, pp. 141–146.

[8] Y. Demchenko, C. Ngo, C. de Laat, and C. Lee, "Federated access control in heterogeneous intercloud environment: Basic models and architecture patterns," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, March 2014, pp. 439–445.

[9] D. Fang, X. Liu, I. Romdhani, and C. Pahl, "An approach to unified cloud service access, manipulation and dynamic orchestration via semantic cloud service operation specification framework," *Journal of Cloud Computing*, vol. 4, no. 1, p. 14, 2015. [Online]. Available: http://dx.doi.org/10.1186/s13677-015-0039-3

[10] J. B. Bernabe, J. M. M. Perez, J. M. A. Calero, F. J. G. Clemente, G. M. Perez, and A. F. G. Skarmeta, "Semantic-aware multi-tenancy authorization system for cloud architectures," *Future Generation Computer Systems*, vol. 32, pp. 154 – 167, 2014, special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12001070

[11] P. Cigoj and B. J. Blažič, "An authentication and authorization solution for a multiplatform cloud environment," *Information Security Journal: A Global Perspective*, vol. 24, no. 4-6, pp. 146–156, 2015. [Online]. Available: http://dx.doi.org/10.1080/19393555.2015.1078424

[12] J. Howlett, S. Hartman, H. Tschofenig, and J. Schaad, "Application bridging for federated access beyond web (abfab) architecture," Internet Requests for Comments, RFC Editor, RFC 7831, May 2016.

[13] D. W. Chadwick, K. Siu, C. Lee, Y. Fouillat, and D. Germonville, "Adding federated identity management to openstack," *Journal of Grid Computing*, vol. 12, no. 1, pp. 3–27, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10723-013-9283-2

[14] I. S. Sette, "Access control in iaas multi-cloud heterogeneous environments," Ph.D. dissertation, Universidade Federal de Pernambuco, Recife, PE, Brazil, August 2016.