Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/jisa

Multi-tenant attribute-based access control for cloud infrastructure services

Canh Ngo [*], *Yuri Demchenko*, *Cees de Laat*

Informatics Institute, University of Amsterdam, Science Park 904, 1098XH Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Available online

Keywords:
Access control
Multi-tenant
Attribute-based access control
Intercloud
Cloud computing

ABSTRACT

Cloud Computing is developed as a new wave of ICT technologies, offering a common approach to on-demand provisioning of computation, storage and network resources that are generally referred to as infrastructure services. Most of currently available commercial cloud services are built and organized reflecting simple relations between single provider and customers with the simple security and trust model. New architectural models should deliver multi-provider heterogeneous cloud services environments to organizational customers representing multiple user groups. These models need to be enforced by consistent security services operating in virtualized multi-provider cloud environment. They should incorporate complex access control mechanisms and trust relations among cloud actors. In this paper, we analyze cloud services provisioning use-cases and propose an access control model for multi-tenant cloud services using attribute-based access control model. We also extend the model for Intercloud scenarios with the exchanging tokens approach. To facilitate attribute-based policy evaluation and implementing the proposed model, we apply an efficient mechanism to transform complex logical expressions in policies to compact decision diagrams. Our prototype of the multi-tenant attribute-based access control system for Intercloud is developed, tested and integrated into the GEYSERS project. Evaluations prove that our approach has a good performance in terms of numbers of cloud resources and numbers of clients.

1. Introduction

Cloud Computing is effectively used to improve scalability, availability, elasticity and security of IT management in many application areas. It adopts advantages of many technologies such as virtualization, service-oriented architecture, and Utility computing to allow customers and providers to cut costs on system deployments and operations. Many studies and best practices documents related to clouds deployment, design, development, operations and management have been proposed to incorporate above technologies (Dillon et al., 2010; Foster et al., 2008; Fox et al., 2009; Hogan et al., 2011; Höfer and Karagiannis, 2011; Mell and Grance, 2011). Clouds in such approaches enable users' data to store on share virtualized cloud infrastructures, which are on-demand provisioned at providers' facilities. The virtualized infrastructures capacities can be elastically scaled up and down depending on varying users' demands. Cloud providers build up their systems based on the multi-tenant architecture (Chong et al., 2006; Garcia-Espin et al., 2012; Guo et al., 2007; Mell and Grance, 2011). Thus, security in general as well as access control for cloud service management should be aware of the multi-tenancy pattern in this architecture.

[*] *Corresponding author.* Informatics Institute, University of Amsterdam, Science Park 904, 1098XH Amsterdam, The Netherlands. Tel.: +31 20 525 6918.
E-mail address: t.c.ngo@uva.nl (C. Ngo).

http://dx.doi.org/10.1016/j.jisa.2015.11.005
2214-2126/© 2015 Elsevier Ltd. All rights reserved.

In cloud resource management, resources in cloud are virtualized and managed in a common resource pool (Chong et al., 2006; Garcia-Espin et al., 2012; Ghijsen et al., 2013; Hogan et al., 2011; Mell and Grance, 2011). Depending on the stage in its life cycle, the resource may be administrated by one or multiple entities in the the multi-tenancy pattern:

- At the initial stage, resources are managed by the provider, who is the economic and management owner of available idle resources.
- When a tenant subscribes set of cloud resources, their economic and administrative ownerships will be transferred exclusively to this tenant during subscribed period.
- The subscribed tenant may want to allow accesses from its users, or due to collaboration requirements, share part of its resources to another trusted tenant with specific conditions like allowed actions, time, location.
- The trusted tenant in turn can manage the shared resources by defining access control policies for its users, or share to another one.

A use-case of multi-tenancy pattern could be as follows: a cloud provider offers cloud services to commercial companies (tenants) in which their employees can access subscribed cloud resources (e.g., storage, Virtual Machine (VM), databases, spreadsheets, etc.) during subscribing time. The provider should guarantee isolations between subcribed resources of tenants. On the other side, tenants may want to collaborate with each other by sharing resources. The commercial firm C, wants to audit its finance statements. It signs the contract with the auditing firm A to allow A's consultants can read-only to parts of C's resources during a specific time-frame.

According to Hogan et al. (2011), cloud services relies mainly on virtualization, multi-tenant architecture, elasticity and diversity of accesses. The characteristics of multi-tenancy pattern bring distinctions between Cloud Computing and previous distributed systems. For this reason, access control for clouds should be designed to support such scenarios.

The on-demand self-service and rapid elasticity properties in clouds (Mell and Grance, 2011) requires that the access control design must handle dynamic changes of entities as well as fine-grained authorization. For example, a typical cloud Infrastructure as a Service (IaaS) service provides different plans (e.g., storage size, speed, computing powers, bandwidth, lifetime, etc.) to customers in which the number of subscribers could reach thousands. Each of them can then manage hundreds of end-users. In these cases, numerous resource objects are provisioned over time with dynamic identifiers. Besides, clouds must handle accesses from users using diversity of clients in both types and numbers (e.g., mobile devices, laptops, workstations).

Traditional access control models are designed to manage accesses from subjects to objects with specific operations via authorization statements. A trivial statement is a triple of ⟨*subject, object, operation*⟩, in which the ⟨*object, operation*⟩ is known as a permission. Role-based Access Control (RBAC) approaches (ANSI, 2004; Ferraiolo et al., 2001; Sandhu et al., 1996) were introduced with roles as an abstraction layer decoupling subjects and permissions. RBAC was supported to apply in different areas, from stand-alone, enterprise-level or cross-enterprise applications. However, even the design purpose of RBAC is to large enterprise systems with even hundreds or thousands of roles and users in tens thousands (Sandhu et al., 1999), such systems may have problems on scalability in role and object explosions (Franqueira and Wieringa, 2012; Kuhn et al., 2010). Analysis in Franqueira and Wieringa (2012) estimates that RBAC should be used for systems with static structure where roles and hierarchy are clearly defined; entities individuality and locality are limited; and managed objects are stable. Large-scale cloud services management systems often have dynamics of provisioned pooling objects, varieties of entities and sophisticated fine-grained authorization regarding dynamical context-specific attributes, in which RBAC approaches may not be suitable.

To overcome limitations of RBAC systems, Attribute-based Access Control (ABAC) was identified with the central idea that access can be determined based on present attributes of objects, actions, subjects and environment in the authorization context (Hu et al., 2014; Jin et al., 2012; Yuan and Tong, 2005). The ABAC can be used to model RBAC as well as other traditional access control models (Jin et al., 2012) The fine-grained authorization feature of ABAC makes it more flexible and scalable than RBAC. Thus, ABAC is mostly suitable for cloud management services.

However, using large numbers of attributes in ABAC and the elasticity of clouds produce challenges in management and deployment. The complexity of attributes criteria in rules and conflict resolutions may arise during applying ABAC in access control for large-scale systems like cloud. ABAC implementation like eXtensible Access Control Markup Language (XACML) standard (OASIS, 2013) only limits at defining a general ABAC policy language but without indicating how to integrate with system resource information models for attribute management. There is also no related work on ABAC defining required constraints in policy composition and management for multiple authorities in multi-tenant systems.

With all such challenges and motivated by cloud and Intercloud scenarios analyses (GEANT, 2010; GEYSERS, 2010; Ngo et al., 2011, 2012), as well as related work on access control for clouds (Amazon, 2013; Bernal Bernabe et al., 2012; Bethencourt et al., 2007; Calero et al., 2010; Goyal et al., 2006; Jin et al., 2014; Sahai and Waters, 2005; Tang et al., 2013), we introduce the Multi-tenant Attribute-based Access Control (MT-ABAC) approach which formalize the ABAC applied for the multi-tenancy pattern. It not only aims to provide a scalable and flexible resources and entities management of the ABAC, but also contains related policy constraints facilitating delegations and collaborations among tenants and users in multiple levels. The extended model is applied for Intercloud scenarios with the exchanging tokens approach for fine-grained dynamic trust establishment. To facilitate attribute-based policy evaluation and implementing the proposed model, we apply an efficient mechanism to transform complex logical expressions in policies to compact decision diagrams. Our prototype of the multi-tenant access control system for Intercloud is developed, tested and integrated into the GEYSERS project. Evaluations demonstrate that our system has good performance in terms of number of cloud resources, clients and policies.

The rest of the paper is organized as follows. Section 2 reviews the related work on access control for clouds, Attribute-based Encryption (ABE) and distributed authorizations. Section 3 contains Preliminaries on the cloud information models and the basic ABAC. These materials are used to formulate our proposed approach in Section 4. The proposed model is then analyzed and validated in Section 5. Section 6 discusses a mechanism to efficiently manage contexts in our model. We also extend our model for Intercloud scenarios in Section 7. After that, Sections 8 and 9 contain our design, implemenation and evaluation of the prototype in the GEYERS project. Finally Section 10 concludes our paper.

2. Related work

A number of approaches and contributions in access control for cloud service management have been proposed. Based on the Common Information Model (CIM), authors in Bernal Bernabe et al. (2012) and Calero et al. (2010) proposed a RBAC model integrating with the cim. In this work, an authorization statement defined as the 4-tuple of ⟨*issuer, subject, privilege, resource*⟩ is written as a rule using Semantic Web Rule Language (SWRL) (Horrocks et al., 2004). The rule is then reasoned by a DL Reasoner to transform into RDF statements. Bernal Bernabe et al. (2012) illustrated that it is possible to use proposed model to support RBAC features for users of a tenant. Inter-tenant collaborations are represented by sharing context information, so the trustee can define authorization statements. However, they do not support multiple level delegations among tenants as well as granularity of inter-tenant trusts is limited. Besides, if cloud systems scale up with number of resources or subjects (tenants and users), complexities of policies with number of authorization statements, the DL reasoner mechanism could be the potential bottleneck when number of RDF statements may explode. Moreover, by utilizing an OWL DL implementation (SWRL) as the access control language, the expressiveness and flexibility of the policy language is limited comparing to other policy languages like XACML.

The Multi-tenant Role-based Access Control (MT-RBAC) (Tang et al., 2013) extended the basic RBAC with a set of models including administration features. Beside regular intra-tenant permission and role assignment operations, the cross-tenant collaborations are performed by sharing roles. The truster tenant can define either all roles to trustee tenants (MT-RBAC$_0$), the same public roles to all trustees (MT-RBAC$_1$), or separated public roles to different trustees (MT-RBAC$_2$). In turn, the trustee can perform two administrative operations: (i) user assignment (UA) to its users and (ii) role hierarchy on the shared roles. The prototype was carried out using the attribute-based policy language XACML with the RBAC profile extension. Even though MT-RBAC resolved problems of access control for multi-tenancy by using RBAC, it suffered existing issues of RBAC on scalability and flexibility. The proposed model also did not contain constraints as in our approach to protect isolation and prevent policy confliction.

Jin et al. (2014) extended the ABAC model from Jin et al. (2012) to IaaS scenarios. In their model, entities were classified into cloud root user who can manage Virtual Infrastructure (VI) and tenants; tenant root user who can configure attribute profile and manage tenant admin users; tenant admin users in a tenant can manage tenant regular users and finally tenant regular users who can operate on cloud resources. Using the policy language defined in Jin et al. (2012), the prototype was integrated with an OpenStack system. Although the approach used ABAC in applying for multi-tenant scenarios, its model is not aware of policy confliction problems in multitenancy when multiple entities can define policies. Thus, they also did not contain isolation and grant constraints for policies as in our approach, which would resolve the policy confliction problems. In addition, their model also did not define collaborations between tenants.

To protect data in outsourced environments like clouds, ABE research (Bethencourt et al., 2007; Goyal et al., 2006; Sahai and Waters, 2005) was proposed for security of outsourcing storage, while homomorphic encryption (Brakerski and Vaikuntanathan, 2011; Smart and Vercauteren, 2010) was proposed to secure computation on hostile systems. In the key-policy ABE approach (KP-ABE) (Goyal et al., 2006), request attributes were associated to ciphertexts, and policies were associated to users' keys. The ciphertext-policy ABE (CP-ABE) scheme (Bethencourt et al., 2007) provided a mechanism that allows creating users' keys based on their attributes, and attribute-based policies to protect data are associated in ciphertexts. The ABE schemes were extended and applied to secure data on cloud storage services (Li et al., 2013; Yu et al., 2010). Although the homomorphic encryption may provide confidentiality in outsourcing computation, its applications on cloud were still limited due to the complexity and performance overhead (Naehrig et al., 2011). All these cryptographic mechanisms can be seen as the AEF in the ISO 10181-3 access control framework (ISO, 1996), while our work focuses on access control decision components. Therefore, ABE and homomorphic encryption are orthogonal with our proposal.

Amazon AWS Identity and Access Management (IAM) (Amazon, 2013) is the integration of an identity management system and an access control mechanism. Upon subscribing to an Amazon AWS product, each customer is assigned an AWS tenant account. All operations on AWS products are then bound to this account. Amazon IAM provides a mechanism to create and manage multiple users binding to the AWS tenant account. Using JSON-style authorization policies storing at the IAM side or attaching at the AWS product side, the IAM could control user activities on AWS resources. To guarantee security requirements on confidentiality and integrity, users are allocated their own security credentials to access AWS resources. However, supporting policy language in Amazon IAM is not very expressive with simple attribute-based policies with limited RBAC features. Cross-account access is defined by creating an IAM policy of the trustor account to the trustee account. The trustee then can delegate these privileges to its users. It does not support multiple-level cross-account collaborations.

OAuth authorization framework (D. Hardt and Recordon, 2012) enables a third-party to access a HTTP resource by approval of the data owner via tokens. It provides a workflow protocol for distributed authorization currently applied in various cloud-based services such as Google APIs and Twitter APIs. However, OAuth 2.0 does not used any cryptographic mechansism but relies its security based on HTTPS, which is not flexible in scenarios when clients using proxies or cannot have direct connections with the resources.

Compared to the related work, our proposal resolves the access control problems for multi-tenancy more complete. We formalize the ABAC in the multi-tenant model with isolation and grant constraints, that prevent policy conflict problems. Our approach also supports flexible collaborations between tenants with multiple levels of delegation. Our extended model for Intercloud uses the token exchange approach designed for MT-ABAC which synchronizes constraints and applies cryptographic mechanisms to solve the distributed authorization with delegation constraints issues, which is more suitable for our approach than the current OAuth 2.0 framework.

3. Preliminaries

To solve mentioned challenges in access control for cloud resource management systems, our MT-ABAC contains the following features:

- Support diversity of subjects: different entities are able to compose policies to manage their objects, including the cloud provider, subscribed tenant and shared tenants.
- Fine-grained access control based on the ABAC model.
- Flexible inter-tenant collaborations that allows tenants to share subscribed cloud resources in multiple levels.
- Dynamic constraints applied to policy management for multiple authorities: we define isolation and grant constraints to check if updated policies are compliant to the multi-tenant policy management properties. This guarantee would improve system performance when no conflict occurs during evaluation run-time. Previous works done by Calero et al., 2010; Bernal Bernabe et al., 2012; Tang et al., 2013; Jin et al., 2014 are not aware of this feature.
- Policy generation for dynamic objects: In cloud systems, provisioned cloud resources are the objects of authorization. Their identifiers are generated during provisioning phases and released at the end of the subscription. The cloud provider needs to define authorization statements for such objects automatically according to the on-demand self-service property. Our model is defined to integrate with the resource information model that can be used to generate policies from predefined policy templates binding with cloud service plans.

In following sections, firstly we introduce an information model used to manage cloud resources in our scenarios in the GEYSERS project (GEYSERS, 2010) and the basic ABAC model. Based on the basic ABAC, we propose our MT-ABAC model in Section 4.

3.1. *Information model for virtual cloud infrastructure*

Cloud resources management for both physical and virtual aspects requires a well description for modeling, discovery, composition, monitoring and synchronization. For such purposes, we use the Infrastructure and Network Description Language (INDL) (Ghijsen et al., 2013) to model cloud resources. The ontology of IaaS cloud layer is briefly illustrated in Fig. 1. INDL could model Virtual Resources (VRs) implemented on physical devices in different stages (e.g., abstracted, reserved and instantiated) that belong to different cloud providers. The VI is the composition built up from different types of general VRs. It enables on-demand provisioning and elastic scaling of resource capabilities.

Because the cloud information model such as INDL could represent the extensibility and flexibility of cloud resources configuration in run-time life cycles, integration of the access control with the information model permits the policies can be updated automatically upon such on-demand provisioning changes. For example, a cloud provider requires that upon subscribing an IaaS plan, the tenant could perform or manage (i.e., allows its users to act on behalf) on provisioned VMs and network links (e.g., instantiate, reconfigure, monitor), but not exceeding tenant's subscribed capabilities. This requirement can be done either by implementing directly inside the cloud management system, or decoupled the configuration with access control policies. The latter option using policies to manage capabilities of the tenant is more flexible due to any changes in the plan can reflex easily by configuration updates, not implementation changes.

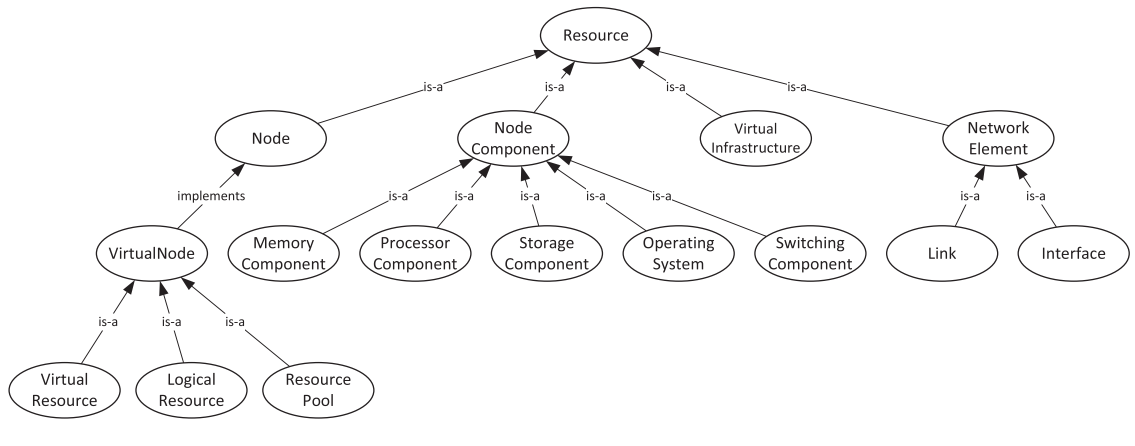

Fig. 1 – Overview of Information Model for Cloud Infrastructure Resources.

3.2. *Attribute-based access control*

ABAC definitions has been described in different forms, but in general access is determined based on matches between specific attribute values of the subject, resource and environment conditions (Hu et al., 2014; Takabi et al., 2010). The policy implemented in ABAC is the mechanism that represents the mapping from the attribute values to authorization decisions and it is only limited by expressiveness of the computational language.

The ABAC concepts from Hu et al. (2014) and Yuan and Tong (2005) will be applied in our approach:

Definition 1 (ABAC concepts). The ABAC has the following concepts:

- *Subject* is the active entity to request access on a resource. In our model, the subject can either be a provider, a tenant, or a user of a tenant. A subject S is characterized by set of attributes. They may include subject's identifier, name, organization, etc. Let $A_{s_1}, A_{s_2}, \dots A_{s_n}$ be value sets (or domains) of subject's attributes, the set of subjects S is defined as the subset of the Cartesian product of k subject's attribute domains:

$$S \subseteq A_{s_1} \times A_{s_2} \times \dots \times A_{s_k} \quad (1)$$

Each subject $s \in S$ is a tuple of subject attribute values: $s = (a_{s_1}, a_{s_2}, \dots a_{s_k}),\ a_{s_i} \in A_{s_i},\ i \in [1, k]$.

- *Resource* is the cloud object that needs to be protected. It could either be in the idle state managed by the provider, or reservation and deployment states and managed by a tenant. A resource is referred by its identifier attribute. However, due to cloud provisioning systems, the identifier of a resource is unknown prior provisioned to the subscribed tenant. In typical ABAC models, actions on a resource often depend on the resource's characteristics. So without loss of generality, action attributes can be seen as attributes of the resources. Similar to the subject, the set of resources is defined as:

$$R \subseteq A_{r_1} \times A_{r_2} \times \dots \times A_{r_l} \quad (2)$$

in which $A_{r_i}, i \in [1, l]$ is the domain of a resource's attribute. A resource $r \in R$ is a tuple of attribute values: $r = (a_{r_1}, a_{r_2}, \dots a_{r_m}), a_{r_i} \in A_{r_i}$.

- *Environment conditions*: attributes such as date, time, system security level, location, etc. are grouped as the environment attributes. The set of environment conditions is defined as:

$$E \subseteq A_{e_1} \times A_{e_2} \times \dots \times A_{e_m} \quad (3)$$

in which $A_{e_i}, i = [1, m]$ is the domain of an environment's attribute. An environment condition $e \in E$ is a tuple of attribute values: $e = (a_{e_1}, a_{e_2}, \dots a_{e_m}),\ a_{e_i} \in A_{e_i}$.

- *Authorization request*: an authorization request x is a tuple of attribute values sent by the subject entity s to the Policy Decision Point (PDP) asking for access to the resource r under environment condition e. The set of requests X is defined as:

$$\begin{aligned} X &= S \times R \times E \\ &= \{(s, r, e) | s \in S, r \in R, e \in E\} \end{aligned} \quad (4)$$

- *Policy*: attribute-based policies are represented by a policy language, that has the semantic as the first-order logic. A policy contains a predicate function mapping from authorization request domain X in Eq. (4) to the decision domain:

$$f : X \mapsto \{Y, N\} \quad (5)$$

with 'Y' and 'N' representing permitted and denied decisions. In this formula, the predicate function f defines a *n-ary* relation of authorization request X.

There are different attribute-based policy languages that can be used in ABAC systems (Amazon, 2013; OASIS, 2013). In our model in Section 4, we propose the authorization statement as the abstraction of policy. The implementation in Section 6 discusses on how to apply XACML to transformation policies into authorization statements.

Proposed model

3.3. *Multi-tenant attribute-based access control model*

In this section, we propose our attribute-based access control model with multi-tenancy properties, (hereinafter referred to as MT-ABAC) for cloud resource management. Compared to the general ABAC, it decouples subjects into providers, tenants and users of tenants, as well as defines constraints for multi-tenant management.

In our model, each cloud provider p is an autonomous system that manages a set of tenants, a set of users and a set of resources. Interactions between multiple autonomous systems (i.e., multi-providers) are discussed in Section 7.

Fig. 2 represents the relationships between subjects in the cloud management systems that supports multi-tenancy properties.

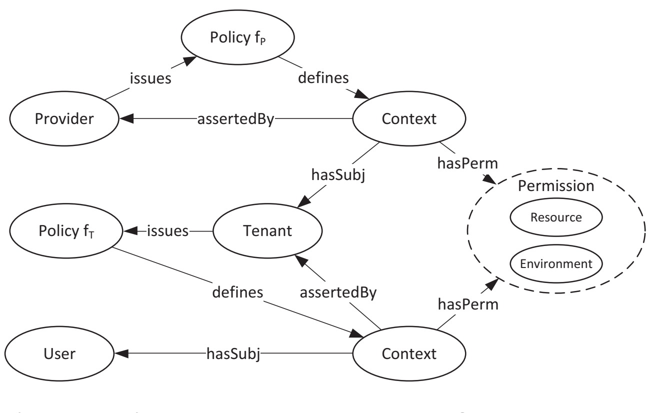

Fig. 2 – Multi-tenant access control model for cloud infrastructure resources.

The multi-tenant access control model has the following concepts:

3.3.1. *Provider*

Let $P \subseteq S$ be the set of providers who can provide resources in the multi-tenant system. A provider $p \in P$ is the subject to assign cloud resources to tenants. It is in charge of provisioning and composing the cloud resources from VR. The cloud provider uses the information model to provision and scale cloud resources to tenants, at the same time regulate tenants' operations according to its information model.

3.3.2. *Tenant*

Denoted $T \subseteq S$ be set of tenants who can subscribe resources in the multi-tenant system. The tenant $t \in T$ is the subject to manage subscribed cloud resources with the following operations:

- Define policies to determine which of its users can access on managed resources.
- Delegate policy management to another tenant on the specified resource via delegation mechanism. This feature is used in the inter-tenant collaboration.

When a tenant subscribes resources from a provider, they have the relation *TenantOf* which is defined as follows:

$$TenantOf \subseteq T \times P \tag{6}$$

With this relation, the set of tenants of the provider $p \in P$ is denoted as $T(p)$ or T_p:

$$T(p) = \{t \in T | t\, TenantOf\ p\}$$

3.3.3. *User*

Let $U \subseteq S$ be the set of all users who consumes resources in the multi-tenant system. In our model, P, T and U have the following properties:

$$S = P \cup T \cup U$$

$$(P \cup T) \cap U = \emptyset$$

The relation *UserOf* defines the set of users of a tenant:

$$UserOf \subseteq U \times T \tag{7}$$

Given a tenant $t \in T$, its users is denoted as $U(t) = \{u \in U | u\, UserOf\ t\}$.

The set of users of a provider p is denoted as:

$$U(p) = \bigcup_{t \in T(p)} U(t) \tag{8}$$

3.3.4. *Resource*

Let R be set of all resources as in Eq. (2). A set of resources owned by either a tenant or provider defined by the relation:

$$ResourceOf \subseteq R \times (T \cup P) \tag{9}$$

A tenant $t \in T$ has its subscribed resource $R(t) = \{r \in R | r\, ResourceOf\ t\}$. A provider $p \in P$ manages their idle resources $R(p) = \{r \in R | r\, ResourceOf\ p\}$. According to the exclusive resource ownership in multi-tenant systems, $\cup x, y \in T \cup P$ we have:

$$R(x) \cap R(y) = \emptyset \Leftrightarrow x \neq y \tag{10}$$

3.3.5. *Permission*

Let $\mathcal{P}$ be the set of permissions defined as:

$$P \subseteq R \times E \tag{11}$$

Each permission is a tuple $(r, e) \in P$ represented by a set of attributes identifying a resource with its action, and specific environment condition attributes.

3.3.6. *Context*

Definition 2 (authorization statement). Authorization statement is the assertion to say that the issuer $i \in S$ authorizes subject $s \in S$ on a permission (r, e). It is denoted as $authz(i, s, (r, e))$.

The authorization statement has the transitive property (Crampton and Khambhammettu, 2006) as follows:

$$authz(s_1, s_2, (r, e)) \wedge authz(s_2, s_3, (r, e)) \rightarrow authz(s_1, s_3, (r, e)) \tag{12}$$

Definition 3 (Context). Context is a statement of the issuer $i \in S$ on the approval of set of permissions to the subject $s \in S$. A statement can be denoted by a tuple (*issuer*, *subject*, {*permissions*}), in which the *issuer* can be either a provider or a tenant and the subject can be either a tenant or a user. The formal definition of the context is described in Eq. (15).

We classify the following contexts:

- Authorization context (AC) is issued by a tenant to a user:

$$AC \subseteq T \times U \times \mathcal{P}^n \tag{13}$$

- Delegation context (DC) is issued by either a provider to its tenant or a tenant to another tenant:

$$DC \subseteq (P \cup T) \times T \times \mathcal{P}^n \tag{14}$$

Formally, the context is defined as:

$$\begin{aligned} C &= AC \cup DC \\ &\subseteq (T \times U \cup (P \cup T) \times T) \times \mathcal{P}^n \end{aligned} \tag{15}$$

Given a context $c \in C$, we denote $\mathcal{I}(c)$ as the issuer of c, $\mathcal{S}(c)$ as the subject of c and $\mathcal{P}(c)$ as the set of permissions in c.

According to definition 2, given a context $c = (i, s, \mathcal{P}(c))$, we have:

$$\forall (r, e) \in \mathcal{P}(c), \quad authz(i, s, (r, e)) \tag{16}$$

Let $\mathcal{R}(c)$ be set of resources referred in the context c. Formally, it is defined as:

$$\mathcal{R}(c) = \{r \mid r \in R, \exists (r, e) \in \mathcal{P}(c)\} \tag{17}$$

3.3.7. *Authorization request*

The request $x = (s, r, e) \in X$ is defined in Eq. (4).

Given a context c, the request x is authorized by a c if and only if:

$$\begin{cases} s = \mathcal{S}(c) \\ (r, e) \in \mathcal{P}(c) \end{cases} \tag{18}$$

According to (16), we see that it is equivalent to the authorization statement $authz(\mathcal{I}(c), s, (r, e))$.

The multi-tenant systems have multiple authorities in which each can define policies freely. According to Eq. (15) in our model, providers and tenants can create contexts (i.e., define policies). So it is possible that some contexts from different issuers may be conflicted and their decisions are contradicted which makes the system inconsistent. In our MT-ABAC, we resolve this problem by defining the delegation model, trusted contexts and constraints. They guarantee that at a given state, the system always returns the consistent decision for an authorization request.

3.4. *Delegations in MT-ABAC*

3.4.1. *Types of delegation contexts*

From Eq. (15), our delegation model defines relations between authorities in the MT-ABAC, including providers, tenants and users of tenants as follows:

- Providers can issue delegation contexts to tenants.
- A tenant can issue delegation contexts to other tenants.
- A tenant can issue authorization contexts to users.

We distinguish delegation contexts crampton2006delegation based on types of issuers and subjects:

- *Transfer context* (*TC*): when a provider provisions its cloud resources to a tenant, it uses the transfer context in which resources are exclusively allocated to only this tenant. The tenant and the provider then have the relationship *TenantOf* as in Eq. (6). The set of transfer contexts is defined as:

$$TC = \{(x, y, \{(r, e)\}) \mid x \in P, y \in T(x), (r, e) \in \mathcal{P}\} \tag{19}$$

- *Grant context* (GC): when a tenant want to share a part of its resources to another tenant, it creates a grant context.

$$GC = \{(x, y, \{(r, e)\}) \mid x, y \in T \setminus P, (r, e) \in \mathcal{P}\} \tag{20}$$

It shows that only tenants without having provider role can create grant contexts.

From Eqs. (19) to (20), we have the following properties:

$$TC \cap GC = \varnothing \tag{21}$$

$$DC = TC \cup GC \tag{22}$$

When transferring a resource $r \in R$ from the provider to a tenant $t \in T$, it creates the relation *ResourceOf* as in (9) between r and t. So the total resource owned by a tenant $t \in T$ is:

$$R(t) = \bigcup_{\forall c \in TC, \mathcal{S}(c)=t} \mathcal{R}(c) \tag{23}$$

Based on the accountable properties in cloud, in which a subscribed cloud resource is exclusively assigned to a tenant during a definite lifetime, we need to define a constraint on transfer contexts so that a resource cannot be provisioned to more than one tenant at a specific environment condition. The isolation constraint will be defined in Section 4.3.

The multi-tenancy system allows tenants to collaborate via the inter-tenant operations, i.e., a resource of a tenant can be accessed by either users of this tenant, or users of another tenant. The inter-tenant is supported by grant contexts as described above. However, to guarantee that a tenant cannot create grant contexts for resources it does not have permissions, we define the grant constraint in Section 4.3.

3.4.2. *Context relationships*

To solve conflicting issues may arise for multiple authorities, we present relationships between issued contexts. At first, they are described for a single provider, then are extended to multiple providers.

Definition 4 (Provider's trust contexts). In a MT-ABAC system of a provider $p \in P$ with a set of its tenants $T(p)$ and their users $U(p) = \bigcup_{t \in T(p)} U(t)$, let $\mathcal{C}(p)$ be set of contexts trusted by p. A context $c = (i, s, \mathcal{P}(c))$ is trusted by $p(a \cdot k \cdot ac \in \mathcal{C}(p))$ if and only if:

$$\forall (r, e) \in \mathcal{P}(c), authz(p, i, (r, e)) \tag{24}$$

Lemma 4.1. If c^* is a transfer context made by p, it is trusted by the provider p:

$$(c^* \in TC) \wedge (\mathcal{I}(c^*) = p) \rightarrow c^* \in \mathcal{C}(p) \tag{25}$$

Proof. Because c^* is the transfer context, we have $\mathcal{I}(c^*) = p$. According to (16):

$$\forall (r, e) \in \mathcal{P}(c^*), authz(p, \mathcal{S}(c^*), (r, e))$$

So by trust context definition (24), we conclude c^* is trusted by p, or $c^* \in C(p)$.

The set of all trusted contexts in the MT-ABAC is denoted as:

$$\mathcal{C} = \bigcup_{\forall p \in P} \mathcal{C}(p) \tag{26}$$

To manage trust contexts of a provider efficiently, we define the partial relationship between two contexts:

Definition 5 (Partial trust relationship). The partial trust relationship $PT \subseteq C \times C$ over two trust contexts of provider p is defined as:

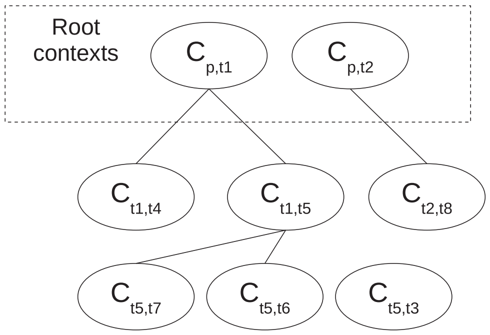

Fig. 3 – An example of context relationships.

$$PT = \{(c_i, c_j) | c_i, c_j \in \mathcal{C}(p), \mathcal{S}(c_i) = \mathcal{I}(c_j) \wedge \mathcal{P}(c_i) \cap \mathcal{P}(c_j) \neq \emptyset\} \quad (27)$$

Thus the function $PT(c)$ returns all contexts in $C(p)$ that c partially trusts. Formally:

$$PT(c) = \{c' \in \mathcal{C}(p) | (c, c') \in PT\} \quad (28)$$

Definition 6 (Tenant privilege scope). The tenant $t \in T$ has its privileges scope:

$$\bar{\mathcal{P}}(t) = \bigcup_{\forall c \in \mathcal{C}, \mathcal{S}(c)=t} \mathcal{P}(c) \quad (29)$$

The MT-ABAC delegations allow a tenant t to share its resources to another one via grant contexts. However, these contexts are not always trusted, because t may grant permissions on unowned resources. To identify if these contexts are trusted, the system need to check if all their permissions belong to the privilege scope of t. If a tenant can grant any contexts, the system runtime overhead on checking their trusts is costly. In Section 4.3, we define constraints at the policy composition stage to prevent such overheads on runtime stage. These constraints can improve the system performance.

Fig. 3 presents context trees of the provider p. Transfer contexts c_{p,t_1}, c_{p,t_2} and c_{p,t_5} are trusted directly by the provider p. Each connection represents the trust relationship between contexts. A context can be trusted by only one context or multiple contexts. In the figure, tenant t_1 share some resources directly from its transfer context to t_4, so $\mathcal{P}(c_{t_1,t_4}) \subseteq \mathcal{P}(c_{p,t_1})$. In the other case, the context c_{t_5,u_1} created by t_5 that combined permissions from different contexts c_{p,t_5}, c_{t_1,t_5} and c_{t_2,t_5} so c_{t_5,u_1} is partially trusted by those contexts.

3.5. *Multi-tenancy constraints*

The system state of a provider $p \in P$ contains the following information: $C(p)$, $T(p)$ and $U(p)$. To simplify definitions, we shorten the system state as $(\mathcal{C}_p, T_p, U_p)$. It evolves over time via administrative operations from the provider and its tenants. In this section, we define constraints to make sure the MT-ABAC system state is consistent.

3.5.1. *Isolation constraint*

The multi-tenancy systems require that tenants should have security isolation (Guo et al., 2007) on different layers. It can be achieved by using implicit filter based on the binding between tenant-id and allocated resources, or with explicit permission-based access control isolation. Our work support the later at the conceptual level by the isolation constraint below.

To guarantee the exclusive resource ownership property in the multi-tenant system, Eq. (10) should be satisfied. Given x, $y \in T$, from (23) and (10), we have:

$$\forall c_1, c_2 \in TC, (\mathcal{S}(c) = x) \wedge (\mathcal{S}(c_2) = y) \rightarrow \mathcal{R}(c_1) \cap \mathcal{R}(c_2) = \emptyset \quad (30)$$

Eq. (30) guarantees that the provider does not transfer a resource to more than one tenant. From this condition, we define the isolation constraint as follows:

Definition 7 (Isolation constraint). Given a system state $(\mathcal{C}_p, T_p, U_p)$, the constraint $canTransfer(c^*)$ on a given transfer context $c^* \in TC$ is valid iff:

$$\forall tc' \in \{tc \in \mathcal{C}_p | \mathcal{I}(tc) = p\} (\mathcal{I}(c^*) = p \wedge \mathcal{S}(c^*) \in T_p \wedge \mathcal{R}(c^*) \cap \mathcal{R}(tc') = \emptyset) \quad (31)$$

in which $\mathcal{R}(c)$ are sets of resources referred in contexts c that is defined in Eq. (17).

3.5.2. *Grant constraints*

In Fig. 3, it is possible that the tenant t_5 issues to t_3 a context containing out-of-scope privileges of t_5 itself (e.g., the write permission on a t_1's folder, while t_1 only allows t_5 to read from it). Therefore, the context c_{t_5,t_3} is untrusted. If there are many untrusted contexts in practical, the authorization process becomes more complex, that affects the system performance.

To prevent this problem, we define the grant constraint applying to the policy composition of tenants (i.e., when a tenant adds, removes or updates its policies) to make sure no context is untrusted. Although this constraint will increase policy composition checking overhead, the authorization evaluation process of the system will improve.

Definition 8 (Grant constraints). Given a system state $(\mathcal{C}_p, T_p, U_p)$ and a context c^* with $i^* = \mathcal{I}(c^*)$, $s^* = \mathcal{S}(c^*)$, the grant constraints are defined as:

- If $c^* \in AC$: the constraint $anGrantAC(c^*)$ is valid iff:

$$(i^* \in T_p) \wedge (s^* \in U(i^*)) \wedge (\mathcal{P}(c^*) \subseteq \bar{\mathcal{P}}(i^*)) \quad (32)$$

- If $c^* \in GC$: the constraint $canGrantGC(c^*)$ is valid iff:

$$(i^* \in T_p) \wedge (s^* \in T_p) \wedge (i^* \neq s^*) \wedge (\mathcal{P}(c^*) \subseteq \bar{\mathcal{P}}(i^*)) \quad (33)$$

For a given context $c^* \in AC \cup GC$, the grant constraint can be written as $canGrant(c^*)$.

The complexity of the algorithm to determine if the context c meets the grant constraint depends on the subset checking operation $\mathcal{P}(c^*) \subseteq \bar{\mathcal{P}}(i^*)$.

Lemma 4.2. Given a system state $(\mathcal{C}_p, T_p, U_p)$ and a context $c^* \in AC \cup GC$. The context c^* is trusted by the provider p if and only if it satisfies the grant constraints.

$$(c^* \in AC \cup GC) \wedge canGrant(c^*) \leftrightarrow c^* \in \mathcal{C}(p) \tag{34}$$

Proof. For the "if" direction: the context c^* can either be an authorization context or grant context. In both cases, we have $\mathcal{P}(c^*) \subseteq \bar{\mathcal{P}}(i^*)$. So that:

$$\forall (r,e) \in \mathcal{P}(c^*) \rightarrow (r,e) \in \bar{\mathcal{P}}(i^*)$$

According to (29):

$$\forall (r,e) \in \mathcal{P}(c^*), \exists c' \in \mathcal{C}(p), ((r,e) \in \mathcal{P}(c')) \wedge (\mathcal{S}(c') = i^*) \tag{35}$$

From the definition of the trust context (24) applied to c', we have:

$$c' \in \mathcal{C}(p) \rightarrow \forall (r,e) \in \mathcal{P}(c'), authz(p, \mathcal{I}(c'), (r,e)) \tag{36}$$

However, based on the definition of the authorization context (16), we have:

$$\forall (r,e) \in \mathcal{P}(c'), authz(\mathcal{I}(c'), \mathcal{S}(c'), (r,e)) \tag{37}$$

Using the transitive property (12), from (36) and (37), we have

$$\forall (r,e) \in \mathcal{P}(c'), authz(p, \mathcal{S}(c'), (r,e)) \tag{38}$$

Replace $i^* = \mathcal{S}(c')$ to (38) and combine with (35):

$$\forall (r,e) \in \mathcal{P}(c^*), authz(p, i^*, (r,e)) \tag{39}$$

Based on the trust context definition (24) applied to (39):

$$\forall (r,e) \in \mathcal{P}(c^*), authz(p, i^*, (r,e)) \rightarrow c^* \in \mathcal{C}(p) \tag{40}$$

For the "only if" direction: according to privilege scope definition (29):

$$\bar{\mathcal{P}}(i^*) = \bigcup_{\forall c \in \mathcal{C}(p), \mathcal{S}(c) = i^*} \mathcal{P}(c)$$

Because $c^* \in C(p)$, we have $\mathcal{P}(c^*) \subseteq \bar{\mathcal{P}}(i^*)$.

If c^* is an authorization context, we have $(i^* \in T_p) \wedge (s^* \in U(i^*))$, so the *canGrantAC*($c^*$) is valid.

If c^* is a grant context, we have $(i^* \in T_p) \wedge (s^* \in T_p)$, so the *canGrantGC*($c^*$) is valid.

In other words, the *canGrant*(c^*) is valid.

3.6. *MT-ABAC operations*

We differentiate two phases in the access control: policy composition phase and authorization evaluation phase. The first occurs when the provider allocates resources to tenants or a tenant composes policies for its users. The later is for authorization request evaluations from users to consume cloud resources.

The system state $(\mathcal{C}_p, T_p, U_p)$ of a provider $p \in P$ evolves over times via administrative commands from the provider p and its tenants T_p. Assume that $(\mathcal{C}'_p, T'_p, U'_p)$ is the new state after an administrative command, Table 1 summaries these commands as follows:

The algorithm 1 is to remove a context c and update C_p.

In the authorization evaluation phase, the MT-ABAC evaluates a request x from a user by finding an authorization context in C so that the request x is authorized by c as defined in Eq. (18).

4. Analysis

In this section, we prove that our system is consistent: given a system state $(\mathcal{C}_p, T_p, U_p)$, after any administrative operations, the new system state $(\mathcal{C}'_p, T'_p, U'_p)$ always maintains the property that all contexts in C'_p are trusted:

$$\forall c \in \mathcal{C}'(p), \forall (r,e) \in \mathcal{P}(c), authz(p, \mathcal{I}(c'), (r,e)) \tag{41}$$

For the operations *addTenant*, *removeTenant*, *addUser*, *removeUser*, the set of contexts does not change: $\mathcal{C}'_p = \mathcal{C}_p$, so (41) is valid.

Table 1 – Administrative commands for MT-ABAC system.

Command	Condition	Update
addTenant(t)	$t \notin T_p$	$T'_p = T_p \cup \{t\}$
removeTenant(t)	$\nexists c \in \mathcal{C}_p(\mathcal{I}(c) = t \vee \mathcal{S}(c) = t)$	$T'_p = T_p \setminus \{t\}$
addUser(t, u)	$t \in T_p \wedge u \notin U(t)$	$U'(t) = U(t) \cup \{u\}$
removeUser(t, u)	$t \in T_p \wedge \nexists c \in \mathcal{C}_p(\mathcal{S}(c) = u)$	$U'(t) = U(t) \setminus \{u\}$
transfer(tc)	$tc \in TC \wedge canTransfer(tc)$	$\mathcal{C}'_p = \mathcal{C}_p \cup \{tc\}$
grant(c)	$(c \in AC \cup GC) \wedge canGrant(c)$	$\mathcal{C}'_p = \mathcal{C}_p \cup \{c\}$
removeContext(c)	$c \in \mathcal{C}_p$	$removeContext(\mathcal{C}_p, c)$

```
proc removeContext($\mathcal{C}_p, c$)
    updateTrustCtxs($\mathcal{C}_p, c$)
    $\mathcal{C}_p \leftarrow \mathcal{C}_p \setminus \{c\}$
    return
proc updateTrustCtxs($\mathcal{C}_p, c$)
    foreach ($c' \in PT(c)$) do
        $perms \leftarrow \mathcal{P}(c') \setminus \mathcal{P}(c)$
        if ($perms \neq \emptyset$) then
            $c' \leftarrow (\mathcal{I}(c'), \mathcal{S}(c'), perms)$
            updateTrustCtxs($\mathcal{C}_p$, c')
        else
            $\mathcal{C}_p \leftarrow \mathcal{C}_p \setminus \{c'\}$
        end
    end
    return
```

Algorithm 1: Remove a context and update $\mathcal{C}_p$ in MT-ABAC

For the *transfer*(*tc*) operation, $\mathcal{C}'_p = \mathcal{C}_p \cup \{tc\}$. Because $tc \in TC$, according to Lemma 4.1, *tc* is trusted by *p*, and (41) is valid.

For the *grant*(*c*) operation, $\mathcal{C}'_p = \mathcal{C}_p \cup \{c\}$. According to Lemma 4.2, *c* is trusted by *p* then (41) is also valid.

For the *removeContext*(c^*) operation, the algorithm 2 is to recalculate all contexts *c* that partially trusted by c^*. It removes the common permissions shared between *c* and c^*, so makes c^* and *c* do not have any relationship: $c \notin PT(c^*)$. The updated context of *c* therefore is still trusted by *p*. The updating process continues recursively until no context in $\mathcal{C}(p)$ has any share permission with c^* or $PT(c^*) = \emptyset$. So this algorithm guarantees that all updates contexts are trusted by *p*.

The consistency of our system guarantees that in the authorization phase, given a system state $(\mathcal{C}_p, T_p, U_p)$ and an authorization request $x = (s, r, e)$, the evaluation process is valid:

$$\exists c^* \in \mathcal{C}_p, authz(\mathcal{I}(c^*), s, (r, e)) \vdash authz(p, s, (r, e))$$

The proof is simple. Because $c^* \in \mathcal{C}_p$, from definition (24), we say that $authz(p, \mathcal{I}(c^*), (r, e))$. Using the transitive property in (12), we conclude:

$$authz(\mathcal{I}(c^*), s, (r, e)) \wedge authz(p, \mathcal{I}(c^*), (r, e)) \rightarrow authz(p, s, (r, e))$$

5. Mechanism to manage contexts in MT-ABAC

In cloud systems with high-scale of resources and tenants, approaches using attribute-based policies such as XACML (OASIS, 2005, 2013) have the advantage of high expressiveness of policy composition. However, there's no efficient mechanism in terms of performance to evaluate and manage these policies. Tang et al. (2013) used traditional XACML implementation in SunXACML (2015) with limited results. Calero et al. (2010) and Bernal Bernabe et al. (2012) used Semantic Web Rule Language (SWRL) and a DL reasoner engine to evaluate policies, which were not mainly designed as an authorization policy engine. Jin et al. (2014) used the PolicyEngine evaluating policies defined in Jin et al. (2012). However they did not describe how to implement the engine. Their experiments did not mention on how complex the policies or random requests, which can significantly affect the evaluation performance.

In Ngo et al. (2015), we have proposed a new mechanism called Multidatatype Interval Decision Diagram (MIDD) to solve such issues on the expressive attribute-based policies, which is proved to have performance advantage. In this section, we apply the MIDD mechanism to manage contexts of the MT-ABAC model.

5.1. *Decision diagrams*

According to the Boole–Shannon expansion, a multi-variable logical function $f : D_1 \times D_2 \ldots \times D_n \rightarrow Boolean$ can be decomposed to partial functions which are free from a variable x_i:

$$f(X) = \bigvee_{P \in \mathcal{P}(D_i)} h_{x_i}(P) \wedge f_{x_i^P} \quad (42)$$

in which $h_{x_i}(P)$ represents a function returning 1 if $x_i \in P$, otherwise 0. P is a partition range of variable x_i. This function can be represented by a decision diagrams as in Fig. 4:

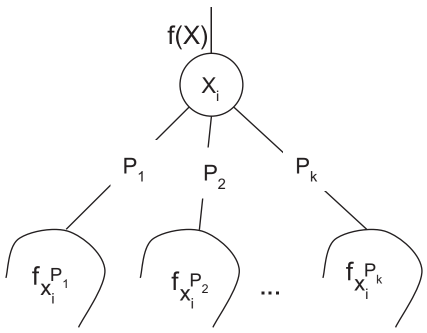

Fig. 4 – A sample Boole–Shannon decision diagram.

Table 2 – XACML evaluation values for elements: Match, AllOf, AnyOf, Target and Condition.

Evaluation values	Annotations
Matched	*T*
No-matched	*F*
Indeterminate	*IN*

However, the XACML language that we apply in MT-ABAC has more complex function signatures. The Match, AllOf, AnyOf and Target elements have the signature in Eq. (43), while Rule, Policy and Policyset elements have the signature in Eq. (44).

$$f: D_1 \times D_2 \ldots \times D_n \rightarrow V_M \quad (43)$$

with $V_M = \{T, F, IN\}$ is the domain of match values in (OASIS, 2013) as in Table 2.

$$f: D_1 \times D_2 \ldots \times D_n \rightarrow V_R \quad (44)$$

The $V_R = \{P, D, N, IN_P, IN_D, IN_{PD}\}$ is the decision rule domain as in Table 3:

Eqs. (43) and (44) can be represented by the graph structure MIDD and Multi-datatype Interval Decision Diagram for XACML (X-MIDD) respectively. A X-MIDD example is illustrated in Fig. 5. In Ngo et al. (2013) and Ngo et al. (2015) we developed algorithms to transform policies into such data structures for evaluations. It's also be used for constraint checking defined in 4.3 in the next section.

Table 3 – XACML decision values for Rule, Policy and Policyset elements.

Decision values	Annotations
Permit	P
Deny	D
NotApplicable	N
Indeterminate{P}	IN_P
Indeterminate{D}	IN_D
Indeterminate{PD}	IN_{PD}

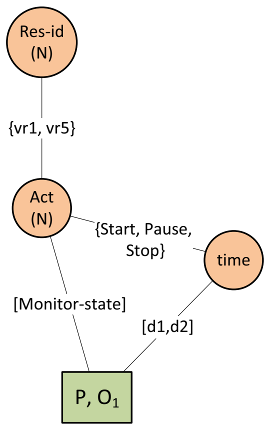

Fig. 5 – X-MIDD representing authorization statements.

5.2. *Context structure*

In Section 4, we propose the context concept representing authorization statements of policies. In this section, we use X-MIDD as the mechanism to implement this concept.

According to definitions in Section 4.1, a context $\langle I, S, \mathcal{P} := \{(X_R, X_E)\}\rangle$ contains the issuer *c#I*, the subject *c!S* and set of permissions *c.P*. The issuer can either be the provider or a tenant, the subject identifies a tenant or a user. Set of permissions *P* contains list of attribute vectors (X_R, X_E) to indicate which resource can be touched (the X_R) in the equivalent condition (X_E). We define a context data structure that have:

- Issuer identifier: is an attribute value or set of attribute values referring to the provider or tenants. In our attribute profile for INDL, the provider and tenants have their unique identifiers, so they can be stored here.
- Subject identifier: contains set of subject attributes (e.g., subject-id, subject-role in XACML attribute profile).
- Permissions: are stored in a X-MIDD structure. It has its variable order, in which subject attributes are at the high level in the tree, resource and environment attributes are at deeper levels, leaf nodes contains *permit* decisions. The X-MIDD has multiple paths from root to its leaf nodes, each path is an authorization statement.

We use XACML as the policy language for tenants, and a subset of XACML as the policy language of the provider. It is also possible to transform and combine a XACML policy-tree of the provider or a tenant into a X-MIDD (Ngo et al., 2013). Then the result X-MIDDs can represent issued contexts. We need to manage these contexts using constraints in Section 4.3.

5.3. *Operations*

According to Section 4, the context has the following operations:

- Function *anTransfer*(*tc*): it implements the Eq. (31)
- Functions *canGrantAC*(*c*), *canGrantGC*(*c*) in Formulas (32) and (33) are illustrated in the algorithm 2.
- Request authorization: given a request *x* and a context *c*, check if *c* authorizes *x*. It can be done by traveling from the root of the context's X-MIDD, if it can reach the leaf node, then the request is authorized.

5.4. *Complexities*

Given a system with |*T*| tenants, each defines two policy-trees, one for its users called intra-tenant policy-tree and the other for tenant sharing called inter-tenant policy-tree. The provider *p* issues policies to tenants, which can be combined to a single policy-tree. Because each policy-tree can be transformed into a X-MIDD (Ngo et al., 2013), the size of the trusted contexts $\mathcal{C}$ is $(2|T|+1)$. The complexity of the algorithm *canGrant* in the worst case is $\mathcal{O}((2|T|+1)\cdot|c|)$ with |c| is the number of paths from the root of X-MIDD in the context *c*.

```
function canGrant($c$)
    foreach $((r, e) \in \mathcal{P}(c)$ do
        $found \leftarrow false$;
        $it \leftarrow \mathcal{C}_p.iterator$;
        while $((\neg found \wedge it.hasNext())$ do
            $c' \leftarrow it.getNext()$;
            if $(\mathcal{S}(c') = \mathcal{I}(c) \wedge (r, e) \in \mathcal{P}(c')$ then
                $found \leftarrow true$;
            end
        end
        if $(\neg found)$ then
            return $false$;
        end
    end
    return $true$
function canGrantAC($c$)
    $i \leftarrow \mathcal{I}(c)$
    return $i \in T_p \wedge \mathcal{S}(c) \in U(t) \wedge$ canGrant($c$)
function canGrantGC($c$)
    $i \leftarrow \mathcal{I}(c)$
    $s \leftarrow \mathcal{S}(c)$
    return $i \in T_p \wedge s \in T_p \wedge i \neq s \wedge$ canGrant($c$)
```

Algorithm 2: Grant constraint functions

6. Extended MT-ABAC for multiple providers

6.1. *Problem statement*

The model MT-ABAC solves multi-tenant access control problems in a single security domain with a cloud provider. For Intercloud scenarios, a provider could play as a tenant of another provider to utilize its cloud resources. This paradigm can be illustrated in GEYSERS (2010) and Ngo et al. (2012) where the Virtual Infrastructure Provider (VIP) can collect VRs from set of Physical Infrastructure Providers (PIPs) to compose the VI. In this section, our model is extended to support Intercloud scenarios by arrange in hierarchy as follows:

- There are set of cloud providers: $p_i \in P$, each of them runs the model in Section 4.
- When a provider p_a subscribes cloud resources from set of providers $p_b = \{p_{b_1}, p_{b_2}, \dots p_{b_k}\}$, p_a becomes the tenant of p_{b_i}, thus can manage these resources with its own policies.
- The p_a can also transfer permissions to its tenants. Permissions either targets to local p_a resources, or the remote resources at a provider $p_{b_i} \in p_b$.
- Any p_{b_i} may be a tenant of other providers, so the chain of providers can be extended.

In such scenarios, we need to solve the challenge of distributed authorization in multiple domains with fine-grained authorization. A request from p_a domain may need to be authorized at two domains, first at p_a domain with relevant p_a tenants' policies, then at p_{b_i} domain with the policy issued by p_{b_i} to p_a. The possible approaches to synchronize and collect decisions are either exchanging tokens or exposing policies between domains. The exchanging token approach needs to deal with token management issues, including storing, synchronization, revocation and overhead of using tokens. In Intercloud paradigm, exchanging policies approach may disclose tenants' Service Level Agreements (SLAs) out of the provider's domain while still has similar issues with token management (Pham et al., 2010). This section proposes a token mechanism that solves token management problems with low overhead on the system performance.

6.2. *Constraints in distributed authorization*

The potential problem in distribute authorization is the conflicting decisions between domains, resulting to the high rejection requests rate at remote domains and increasing system overhead. Preferably, denied requests should be answered as soon as possible at their local domains, rather than at a remote domain in the chain of distributed authorization. It can be solved by establishing grant constraints between the policies at tenant side p_a with policies at the p_b provider sides.

In the above scenario, the provider p_{b_i} issues a policy with the context c_{a_i} for the provider p_a. At the p_a, instead of contexts created by p_a, contexts $c_{a_i|i=\overline{1\dots k}}$ become the root context. All contexts created by p_a must be confined by the root contexts. It can be excepted for local resources X_r physically owned by p_a. To synchronize contexts c_{a_i} between p_{b_i} and p_a, we base on the SLA describing subscribing resources between them. According to the information model (Ghijsen et al., 2013), the SLA request is described by INDL semantic concepts and synchronized upon provisioning and re-planning. We then can use SPARQL and policy generation techniques to extract constrained contexts and update to the trusted root list, which is similar to policy generation in Section 4.

6.3. *Exchanging tokens in Intercloud*

The distributed authorization workflow can be the push sequence as in Fig. 6. It requires that the user needs to have an access token to verify it's allowed to access the resources at remote provider p_{b_i}'s domain. The grant token is initially issued by the first provider in the chain p_a as the consent by p_a to subsequent provider p_{b_i}. The p_{b_i} must validate the token issuer p_a, then evaluate the request attribute embedded inside the token against its policies. If the decision is positive and the target resource is located in its local domain, p_{bi} issues an access token allowing the user to access it. Otherwise, if the target resource is located at another domain, p_b issues another grant token to the user for further distributed authorization process. In this sequence, the communication between authorization services at providers is relayed through the user via exchanging grant tokens and access token.

6.3.1. *Grant token*

The grant token needs to have the following information:

- Request content approved by the issuer, who allows the request to act on behalf of the issuer: it usually is the vector of attributes including issuer's subject attributes.
- The approval proof of the issuer: this proof can be enforced by the digital signature mechanism of the issuer, either based on a digital signature using public cryptography or a message authentication code algorithm using symmetric cryptography.
- The lifetime limitation.
- The proof-of-procession of the user, so the issued access token is not a bearer token and only targets for the user. It's either the user's public key, or the session shared secret key generated by the user.

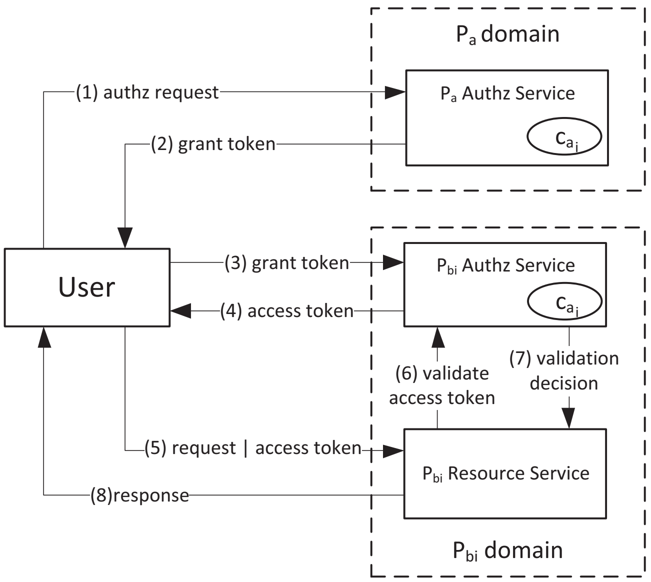

Fig. 6 – Exchanging tokens in Intercloud: grant token and access token.

For the public key cryptography approach, we propose the grant token issued by p_a and returned to the user u as follows:

$$X := \{X_{p_a}, X_r, X_e\}$$
$$m := X|t|pk_u \tag{45}$$
$$granttoken := SK(sk_{p_a}, m)$$

with $SK(sk_{p_a}, m)$ is the annotation that the message m is signed by secret key sk_{p_a} of the provider p_a. The X_{p_a}, X_r, X_e are vectors of attributes of subject, resource and environment respectively. The pk_u is the user's public key, t is the lifetime and X is the vector of attribute request containing p_a's attributes. This grant token allows user to request on behalf of p_a to the remote domain at p_b.

For the symmetric key cryptography approach, the grant token has the following information:

$$m := X|t|k_u$$
$$hmac := MAC(K_{p_a,p_{bi}}, m)$$
$$ek := E(K_{p_a,p_{bi}}, k_u)$$
$$granttoken := \{X|t|ek|hmac\}$$

$$hmac := MAC(K_{p_a,p_{b_i}}, m) \tag{46}$$

with $K_{p_a,p_{b_i}}$ is the shared secret key between the provider p_a and p_{b_i}; MAC is a message authentication code algorithm; k_u is the session key of the user; ek is the encryption of k_u by the $K_{p_a,p_{b_i}}$.

6.3.2. *Access token*

According to the public key approach in Formula (45), the access token issued by p_{b_i} to the user u is constructed as follows:

$$accesstoken := SK(sk_{p_{b_i}}, tid|t) \tag{47}$$

with t is the issuing timestamp, tid is the identifier to the cached authorization session stored at p_{b_i}, which contains access token lifetime, user's associated key pk_u and the involved attributes X.

With symmetric key approach in Formula (46), the access token contains following:

$$accesstoken := E(k_u, tid|t|stoken) \tag{48}$$

$$k_{u,p_{b_i}} = k_u|stoken \tag{49}$$

with $stoken$ is the secret generated value by p_{b_i} shared to the user. The consequent requests from the user to p_{b_i} are signed with the session key $k_{u,p_{b_i}}$.

After having the access token, user accesses the protected resource at p_{b_i}. Upon receiving user's request with access token, the p_b's resource service validates the access token with either pk_{b_i} for public key scheme, or $k_{u,p_{b_i}}$ for symmetric key scheme. If comparison between the request with involved attributes X is positive, the service will serve the request.

7. System design

7.1. DACI architecture and integration

We design the Dynamic Access Control Infrastructure (DACI) as in Fig. 7 in collaboration with an Intercloud architecture in GEYSERS project (GEYSERS, 2010). In this architecture, each PIP runs an instance of the OpenNebula (2013). The VIP plays the role of Intercloud provider that utilize and aggregate computing, storage and network resources from set of PIPs to compose VI services for tenants. To manage cloud resource information between providers, the INDL is used to model and share VRs description.

Each PIP has an instance of OpenNebula (2013) to manage its VRs. PIP runs a system (known as Lower-Logical Infrastructure Composition Layer (LICL)) operating on top the OpenNebula instance via adapters (GEYSERS, 2010) to abstract, control and monitor virtual resource information. This information is synchronized to the Upper-LICL system at VIP. The VI composed by VIP could be distributed across different PIPs' domains, while the tenant ((Virtual Infrastructure Operator (VIO)) of a VI can manage it via the VIP as follows:

- The VIO can send a VI request to the VIP, where the request is analyzed. Based on current available free resources from registered PIPs, the request is broken down into parts which will be provisioned at PIPs. The DACI handles the VI request in reservation phase by the *TenantManagement Service*, that generates provider's delegation policies for a given request. These policies must satisfy the isolation constraint according to Section 4.
- Once the VI is deployed, the VIO can define authorization policies for its end-users via the *TenantAdmin Service*. It also can set which parts of the deployed VI are shared with other tenants via the tenant's delegation policies. These policies are composed in XACML.
- The VIO and its end-users can control VI components via VR Proxies of Upper-LICL, where authorization interceptors extract request attributes and authorize at DACI against local tenant's authorization policies, inter-tenant policies and provider's delegation policies. Depending on returned decisions, the interceptors may reject or permit to process requests.

While the DACI services are designed to integrate with cloud management systems in the GEYSERS (2010), it is also

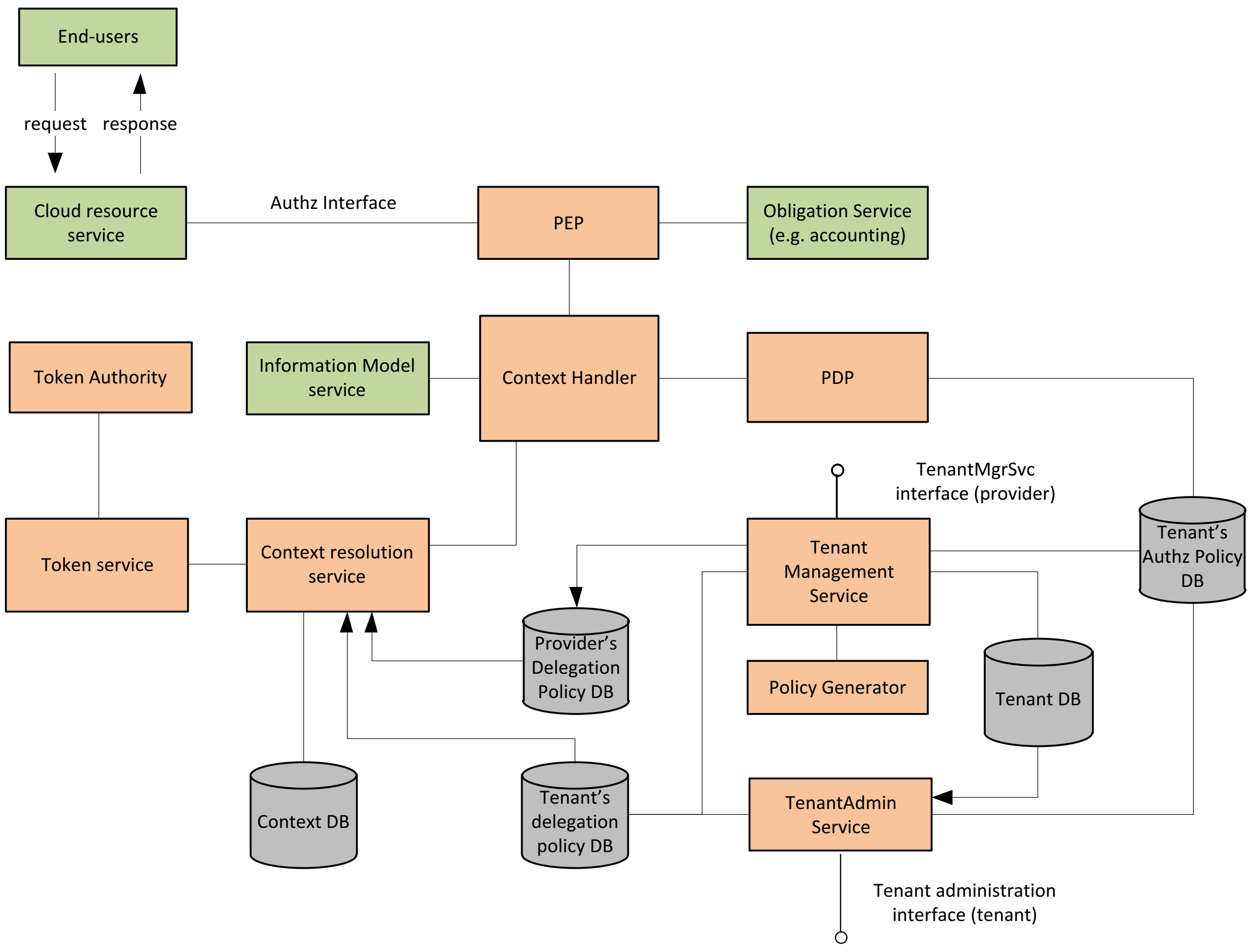
End-users
request
response
Cloud resource service
Authz Interface
PEP
Obligation Service (e.g. accounting)
Token Authority
Information Model service
Context Handler
PDP
TenantMgrSvc interface (provider)
Token service
Context resolution service
Tenant Management Service
Tenant's Authz Policy DB
Provider's Delegation Policy DB
Policy Generator
Tenant DB
Context DB
Tenant's delegation policy DB
TenantAdmin Service
Tenant administration interface (tenant)

Fig. 7 – Dynamic Access Control Infrastructure with Multi-tenant Access Control.

Table 4 – DACI Integration APIs.

Phases	APIs	Description
Reservation	*reserve(tenantId, res_desc)*	Generate provider's policies for given cloud resource description
Deployment	*deploy(tenantId)*	Transform policies into contexts and store to the context DB.
Decommission	*release(tenantId)*	Remove tenant's policies and contexts.

possible to use DACI in other cloud management systems by means of equivalent cloud resource description models. In such cases, the policy generator component allows to parse cloud resource descriptions. The integration APIs with a cloud management system are illustrated in Table 4.

7.2. *Integration MT-ABAC with INDL*

7.2.1. *Attribute-based policy semantic model*

In the MT-ABAC system for clouds, the provider's policies need to be generated automatically based on on-demand resource provisioning, while the tenants' policies can be configured later by customers. Thus, we integrate the MT-ABAC into the cloud information model INDL (Ghijsen et al., 2013) shown in the Fig. 8, which represents provider's policies. Tenants can use other ABAC languages like XACML to compose their policies without limitation.

Any authorization requests to the resource must be checked against these attached policies.

In this INDL extension, the provider's policy is bounded to the resource concept via the relationship *hasPolicy*. Because of the inheritance of resource types, derived resources have all policies of their ancestors, leading conflict may arise. Thus, we define combining operators for joining such

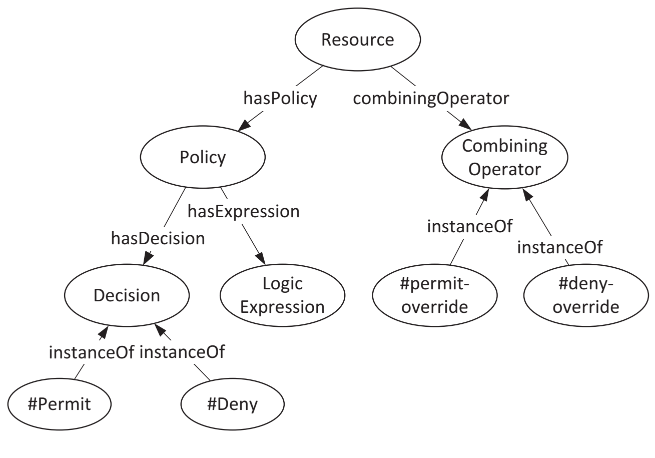

Fig. 8 – Attribute-based policy model integration with INDL.

multiple attached policies. However, rather than many combining operators as in XACML where multiple parties can create conflicting policies, we only need *permit-override* for privilege enrichment and *deny-override* for privilege limitation for inheritance relationships: e.g., the ancestor VirtualNode concept allows to add a new network interface, and its descendant node, a VM, allow to instantiate, so the *permit-override* operator can be used in this case. Thus, it can be seen that the provider's policies in our MT-ABAC utilize a subset of XACML.

In cloud resource life-cycles, the reservation and instantiation of resources occur automatically based on tenants' requests. We need a mechanism to create authorization rules to manage these on-demand resources:

- We define policies for each resource components in a VI, which forms a policy template.
- The policy template is then used to generate the instantiated policies for deployed cloud resources.

7.2.2. *Policy generation from cloud infrastructure descriptions*

In Fig. 2, the provider issues policies for tenants to delegate permissions on their subscribed resources. These operations should be performed automatically in the cloud resource life-cycles, i.e., issuing, updating and revoking policies at the reservation, re-planning and decommissioning phases, respectively. We propose an automatic mechanism to generate policies from cloud infrastructure resource descriptions using INDL as follows:

- Given an infrastructure resource description using INDL implemented by RDF/OWL technologies, we use SPARQL queries to obtain detail resource information of the cloud resources, including resource identifiers, types, owner, as in Listing 1.
- We define the resource policy template using model in Fig. 8, which specifies policies could bind to a resource type, as illustrated in Fig. 9. The policies are retrieved as in Listing 2.
- We create policies with the subject be the tenant, resource identifiers and types along with related actions from the template. They can be either expressed as the generic attribute-based policy notation or a policy language standard like XACML (OASIS, 2013).

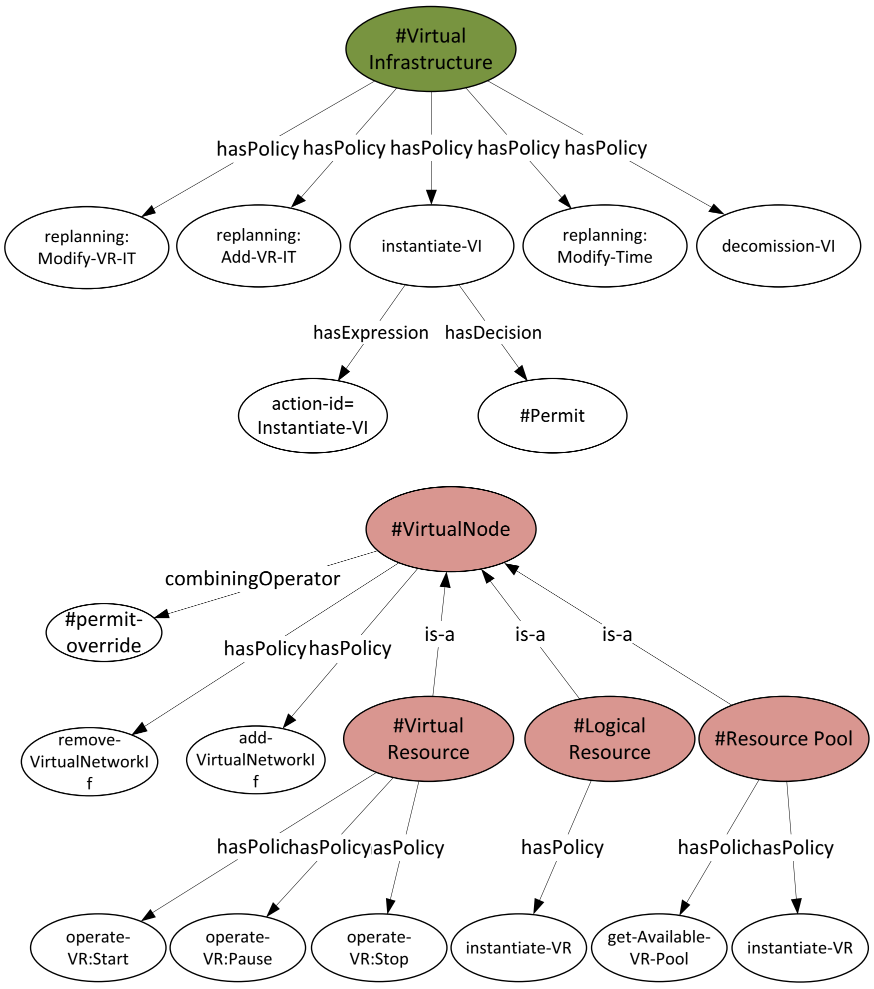

Fig. 9 – Defining policy template sample.

```
PREFIX rdfs: <http://www.w3.org/2000/01/
rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>
PREFIX imf: <http://geysers.eu/imf.owl#>
SELECT ?vi ?r ?rtype
WHERE {
?vi rdf:type imf:VirtualInfrastructure.
?vi imf:hasResource ?r.
?r rdf:type ?rtype.
?rtype rdfs:subClassOf* imf:Resource.
}
```

Listing 1: Query resource components in the virtual infrastructure

```
PREFIX rdf: <http://www.w3.org/1999/02/
22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/
rdf-schema#>
PREFIX imf: <http://geysers.eu/imf.owl#>
PREFIX daci: <http://geysers.eu/imf-daci.owl#>
SELECT ?r ?type ?cop ?d ?expr
WHERE {
#resourcetype# rdfs:subClassOf* ?type.
?r rdf:type ?type.
?r daci:combiningOperator ?cop
?r daci:hasPolicy ?p.
?p daci:hasDecision ?d.
?p daci:hasExpression ?expr
}
```

Listing 2: Retrieve actions for a resource type from its ancestors

In Listing 2, the *resourcetype* parameter is a *Resource* concept in the INDL ontology. Queried result is translated into XACML policies with the equivalent combining operator and logic expressions. These providers' policies are then transformed into the data structure representing root contexts in the next section.

7.3. *High performance PDP for tenants policies*

Tenant's policies are isolatedly stored in the *Tenant Authz Policy DB*. Upon receiving a request from *Context Handler*, the PDP service loads equivalent tenant's policies for evaluation. To gain high performance throughput, we use SNE-XACML engine (Ngo, 2014) to transform regular XACML policies into X-MIDD data structure with much improved throughput compared to other PDP engines.

We create a pool of PDP instances, each for a tenant policy root. By this way, our design is scalable if we plan to extend PDP in different machines.

7.4. *Context resolution and token exchange*

The Context resolution service implements the multi-tenant access control model by extracting delegation policies of providers and tenants to contexts and storing them in the *Context DB*. It finds the trust context for a given request from *Context Handler*. If the trust context has its issuer at a different domain (a PIP), it can either do one of following:

- Proxy method: The VIP creates a VR proxy to send commands to PIP. It's transparent to end-users. This method is implemented in LICL testbeds. This is the direct approach and acceptable with low control and management traffics because they are centralized and routed via VIP to different PIPs' domains.
- Push method: The VIP creates a grant-token and relays via end-users to send commands to PIPs as in Section 7.3. In general Intercloud services, when the control and management traffics from users to underlying providers are high, this approach is more scalable.

7.5. *Tenant policy administration*

A tenant can define its end-users authorization policies as well as inter-tenant delegation policies via policy administration APIs as in Table 5.

Whenever tenants want to add or update policies, the grant constraint in Section 4 is checked to make sure no violation happens. Thus, it reduces the authorization overhead by limiting inconsistent decisions.

Table 5 – Tenant Policy Administration APIs.

Type	APIs	Description
Intra-tenant	*addPolicy(policyId, p)*	Add new policy to the tenant's store.
	updatePolicy(policyId, p)	Update an existing policy
	deletePolicy(policyId)	Delete an existing policy
Inter-tenant	*setTrust(trustee, p)*	Set a new trust relationship between current tenant and *trustee*.
	updateTrust(trustee, p)	Update an existing relationship between current tenant and *trustee*.
	removeTrust(trustee, p)	Remove an existing relationship between current tenant and *trustee*.

Table 6 – VI Datasets.

#VI	#Prov. rules	#Inter-tenant rules	#Intra-tenant rules	Total rules
100	401	394	501	1296
300	1217	394	1517	3128
500	2001	500	2501	5002
800	3211	800	4011	8022
1000	3955	1000	4955	9910

8. Implementation and evaluation

8.1. *Implementation overview*

We develop DACI components as OSGi bundles on Java 1.7. The public DACI interfaces are REST web services based on JAX-RS APIs of the Apache CXF. In our testbed, components in a DACI instance are deployed on the Apache ServiceMix environment (Ser, 2013). Policies for tenants and providers are stored in a Redis key-value database system (Red, 2013) with separated key identifiers for each tenant. It guarantees the isolation of policy management among tenants.

The policy generator module uses Jena OWL engine (Jen, 2013) to parse input VI descriptions in INDL (Ghijsen et al., 2013) and the attribute-based policy template to generate XACML policies as described in Section 8.2.2. They are stored as provider's policies for equivalent VI.

We use SNE-XACML engine (Ngo et al., 2013) as the core PDP to evaluate intra-tenant policies in the *AuthzService* component. For inter-tenant policies and provider's policies, the *ContextService* component transforms into the context objects, each is composed from a MIDD data structure and the policy's issuer attributes. These context objects are stored persistently and used for context validation purposes.

In our testbed, we build a VM representing the VIP role and two VMs for two different PIPs. Each DACI instance in a PIP's VM is registered with the VIP's DACI instance as a tenant. For testing purpose, we generate sample VI datasets with different VI sizes as in Table 6, each VI is one of the following types:

- Type 1: a storage component connected to a VM via a virtual triangular topology network of three virtual routers.
- Type 2: two storage components having network links to a VM.
- Type 3: two VMs having network links to a share storage component.

These VI descriptions are used to generate provider delegation XACML policies. We simulate the inter-tenant operations by generating inter-tenant policies to share resources between tenants, i.e., tenant t_i shares a resource to tenant t_{i+1}. For intra-tenant policies for each tenant, we generate the default policy indicating that the subjects belong to *admin* group of the tenant having all permissions. In practical, inter-tenant and intra-tenant policies are managed by the tenant via the *TenantAdmin* interface.

The DACI for a provider is deployed in a VM with two virtual cores and 4096MB RAM. It runs a ServiceMix instance for DACI and a local Redis server for storing policies.

We use different numbers of Policy Enforcement Points (PEPs) sending requests to the DACI server via the AuthzSvc RESTful interface. They run simultaneously on different machines from the DACI VM, each sends 100 independent requests. The execution times of PEP are measured to calculate the average value.

8.2. *Token implementation*

Our exchanging token approach is implemented in the *TokenService* of the DACI. The service has a public/private key-pair used for issuing and validating tokens. Upon registration, each tenant is bound with a separate public/private key-pair used for Intercloud communication scenario as described in Section 7. In our key management implementation, we choose the RSA algorithm with 2048 bits key length. For digital signature used in issuing tokens, we define the token structure in XML schema and use the XML digital signature standard (XML, 2008) implemented in the Apache XML security library (Apa, 2013). We choose RSASSA-PKCS1-v1.5 signature scheme with SHA-1 algorithm (RSA, 2003). DACI uses Bouncy Castle v1.49 (Bou, 2013) as the Java cryptographic provider.

We deploy DACI instances with TokenService in separate VMs having two virtual cores and 4096 MB RAM. Each VM represents a cloud provider running DACI with the sample datasets in 7 as in Section 4.

In our inter-provider test scenarios, we have two DACIs for P_a and P_b providers, in which P_a subscribes resources of the P_b as described in Section 7.1. PEPs at the user side of the P_a send requests to access to the resource at P_b, so DACI of the P_a needs to evaluate its local policies prior issuing grant-tokens for further authorization at P_b. Compared to the intra-provider scenario in the previous section, the token issues and validations increase overhead of the original DACI system.

8.3. *Evaluation results*

Fig. 10 shows the performance result for the single provider scenario, where the AuthzService on DACI performs authorization evaluation on provider local resources and does not issue tokens. We observe that throughputs are affected by the number

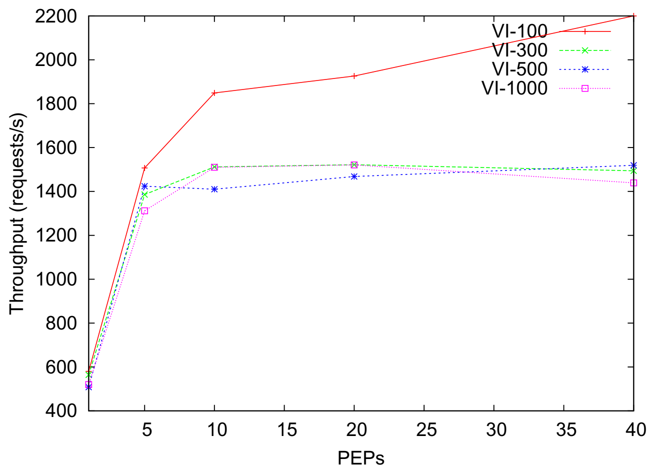

Fig. 10 – Single Cloud provider performance evaluation.

of managed VIs differently. The throughput in VI-100 scenario is higher 12%–40% comparing to other scenarios. For scenarios VI-300, VI-500 and VI-1000, with the same number of PEPs, their throughputs are stable. It means that our prototype is scalable for number of resources.

In other aspect, the result also shows that using high number of PEP for a given dataset in VI-300, VI-500 or VI-1000 can generate enough requests to saturate the AuthzSvc message queue. We measure the AuthzSvc service can handle from 1400 to 1600 requests/s. In our prototype, we do not apply enterprise service patterns like distributed load balancing for web services or decision caching, so with these techniques, the performance will be expected to be better.

From our experiments, the performance tests show that on average, the response time for an authorization request with issuing grant-token is 320 ms, which is significantly slower than the response time in the intra-provider scenario. The overhead here mostly comes from the digital signing tokens with RSA 2048 bits key-length for every issued token and the XML messages serialization/deserialization. Therefore, we are developing a hybrid key management scheme in which tenants and providers use shared secret keys in communications, which are established and refreshed periodically based on the public/private key-pairs. The symmetric key scheme using message authentication code could improve the system performance significantly comparing to the public key scheme.

9. Conclusion

In this paper, we presented a multi-tenant attribute-based access control model for cloud services in which the access control model is integrated with the cloud infrastructure information description model. Our approach not only can generate provider delegation policy automatically from cloud resource descriptions but also can support multiple levels of delegations with high flexibility for inter-tenant collaborations. Constraints were defined to guarantee consistent and correctness of semantic policy management. We utilized decision diagram mechanisms to attribute-based policy evaluation, which also facilitated the implementation of the proposed context in our model. The prototype was developed, tested and integrated into the GEYSERS project. The evaluation results demonstrated that our prototype has good performance in terms of number of cloud resources, clients and policies.

We also extended the MT-ABAC for distributed, multiple collaborative cloud providers in hierarchy to support Intercloud scenarios with exchanging tokens approach. In future work, we will improve key management model for Intercloud using combining public-key and symmetric cryptography, which could improve the system performance in the Intercloud communications using tokens. We are planning to develop adapter layers between our DACI system using INDL with popular cloud management systems like OpenStack, CloudStack or Eucalyptus, thus could integrate the DACI with these systems. Regarding authorization policy language, beside XACML in XML profile, we plan to support others as well as supporting our DACI with legacy on-premise authorization systems.

Acknowledgments

This work is supported by the FP7 EU funded projects The Generalised Architecture for Dynamic Infrastructure Services (GEYSERS, FP7-ICT-248657), GN3plus (FP7-RI-605243) and the Dutch national research program COMMIT.

REFERENCES

Amazon. AWS Identity and Access Management (IAM), <http://aws.amazon.com/iam/>; 2013 [accessed 14.10.15].
ANSI. American national standard for information technology role based access control. Technical Report ANSI INCITS 359–2004 ANSI; 2004.
Apa. Apache Santuario project: XML Signature and Encryption Processing, <http://santuario.apache.org/>; 2013 [accessed 14.10.15].
Bernal Bernabe J, Marin Perez JM, Alcaraz Calero JM, Garcia Clemente FJ, Martinez Perez G, Gomez Skarmeta AF. Semantic-aware multi-tenancy authorization system for cloud architectures. Future Gener Comput Syst 2012;32:154–67.
Bethencourt J, Sahai A, Waters B. Ciphertext-policy attribute-based encryption. In Security and Privacy, 2007. SP'07. IEEE Symposium on (pp. 321–334). IEEE; 2007.
Bou. Bouncy Castle Java Crypto APIs v1.49, <http://bouncycastle.org/java.html>; 2013 [accessed 14.10.15].
Brakerski Z, Vaikuntanathan V. Efficient fully homomorphic encryption from (standard) lwe. In Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on (pp. 97–106). IEEE; 2011.
Calero JMA, Edwards N, Kirschnick J, Wilcock L, Wray M. Toward a multi-tenancy authorization system for cloud services. IEEE Secur Priv 2010;8:48–55.
Chong F, Carraro G, Wolter R. Multi-tenant data architecture, <http://msdn2.microsoft.com/en-us/library/aa479086.aspx>; 2006 [accessed 14.10.15].
Crampton J, Khambhammettu H. Delegation in role-based access control. In: Computer security — ESORICS 2006. Springer; 2006. p. 174–91.
D. Hardt E, Recordon D. The OAuth 2.0 Authorization Framework, draft-ietf-oauth-v2-30. Technical Report. <http://tools.ietf.org/html/draft-ietf-oauth-v2>; 2012 [accessed 14.10.15].
Dillon T, Wu C, Chang E. Cloud computing: issues and challenges. In Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on (pp. 27–33). Ieee; 2010.
Ferraiolo DF, Sandhu R, Gavrila S, Kuhn DR, Chandramouli R. Proposed nist standard for role-based access control. ACM Trans Inf Syst Secur 2001;4:224–74.
Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing 360-degree compared. In Grid Computing Environments Workshop, 2008. GCE'08 (pp. 1–10). Ieee; 2008.
Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, et al. Above the clouds: A berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 28, 13; 2009.
Franqueira V, Wieringa R. Role-based access control in retrospect. Computer 2012;45:81–8. doi:10.1109/MC.2012.38.
Garcia-Espin JA, Riera JF, Figuerola S, Lopez E. A multi-tenancy model based on resource capabilities and ownership for infrastructure management. In Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on (pp. 682–686). IEEE; 2012.

GEANT. GEANT project, <http://www.geant.net/>; 2010 [accessed 14.10.15].

GEYSERS. GEYSERS — Generalised Architecture for Dynamic Infrastructure Services, Technical Report The GEYSERS Project (FP7-ICT-248657). <http://cordis.europa.eu/project/rcn/93786_en.html>; 2010 [accessed 14.10.15].

Ghijsen M, Van Der Ham J, Grosso P, Dumitru C, Zhu H, Zhao Z, et al. A semantic-web approach for modeling computing infrastructures. Comput Elec Eng 2013;39:2553–65.

Goyal V, Pandey O, Sahai A, Waters B. Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on computer and communications security. ACM; 2006. p. 89–98.

Guo CJ, Sun W, Huang Y, Wang ZH, Gao B. A framework for native multi-tenancy application development and management. In E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on (pp. 551–558). IEEE; 2007.

Hogan MD, Liu F, Sokol AW, Jin T. Cloud computing standards roadmap. Technical Report SP 500-291 NIST. <http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909024>; 2011 [accessed 14.10.15].

Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M. SWRL: a semantic web rule language combining OWL and RuleML. W3C Member Submission World Wide Web Consortium. <http://www.w3.org/Submission/SWRL>; 2004 [accessed 14.10.15].

Höfer C, Karagiannis G. Cloud computing services: taxonomy and comparison. J Internet Serv Appl 2011;2:81–94.

Hu VC, Ferraiolo D, Kuhn R, Schnitzer A, Sandlin K, Miller R, et al. Guide to Attribute Based Access Control (ABAC) definition and considerations. Technical Report. <http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf>; 2014 [accessed 14.10.15].

ISO. Security frameworks for open systems: Access control framework. 1996.

Jen. Jena Semantic Web Framework, v2.11.0, <http://jena.apache.org/>; 2013 [accessed 14.10.15].

Jin X, Krishnan R, Sandhu RS. A unified attribute-based access control model covering dac, mac and rbac. DBSec 2012;12:41–55.

Jin X, Krishnan R, Sandhu R. Role and attribute based collaborative administration of intra-tenant cloud iaas. In Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on (pp. 261–274). IEEE; 2014.

Kuhn DR, Coyne EJ, Weil TR. Adding attributes to role-based access control. IEEE Comput 2010;43:79–81.

Li M, Yu S, Zheng Y, Ren K, Lou W. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Trans Parallel Distrib Syst 2013;24:131–43.

Mell PM, Grance T. SP 800-145. The NIST Definition of Cloud Computing. Technical Report Gaithersburg, MD, United States; 2011.

Naehrig M, Lauter K, Vaikuntanathan V. Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. ACM; 2011. p. 113–24.

Ngo C. SNE-XACML project, LGPL v3, <https://github.com/canhnt/sne-xacml>; 2014 [accessed 14.10.15].

Ngo C, Membrey P, Demchenko Y, De Laat C. Security framework for virtualised infrastructure services provisioned on-demand. 2011 IEEE Third Int Conf Cloud Comput Technol Sci 2011;698–704. doi:10.1109/CloudCom.2011.108.

Ngo C, Membrey P, Demchenko Y, de Laat C. Policy and context management in dynamically provisioned access control service for virtualized cloud infrastructures. In Availability, Reliability and Security (ARES), 2012 Seventh International Conference on (pp. 343–349). IEEE; 2012.

Ngo C, Makkes MX, Demchenko Y, de Laat C. Multi-data-types interval decision diagrams for XACML evaluation engine. In Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on (pp. 257– 266). IEEE; 2013.

Ngo C, Demchenko Y, de Laat C. Decision diagrams for xacml policy evaluation and management. Comput Secur 2015;49:1–16.

OASIS. eXtensible Access Control Markup Language XACML version 2.0, <http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf>; 2005 [accessed 14.10.15].

OASIS. XACML v3.0: Core specification, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf>; 2013 [accessed 14.10.15].

OpenNebula. OpenNebula Toolkit, <http://opennebula.org/>; 2013 [accessed 14.10.15].

Pham Q, Reid J, McCullagh A, Dawson E. On a taxonomy of delegation. Comput Secur 2010;29:565–79.

Red. Redis key-value data store, <http://redis.io/>; 2013 [accessed 14.10.15].

RSA. RFC3447: Public-Key Cryptography Standards (PKCS1): RSA Cryptography Specs v2.1, <http://tools.ietf.org/html/rfc3447>; 2003 [accessed 14.10.15].

Sahai A, Waters B. Fuzzy identity-based encryption. In: Advances in cryptology—EUROCRYPT 2005. Springer; 2005. p. 457–73.

Sandhu R, Coyne E, Feinstein H, Youman C. Role-based access control models. IEEE Comput 1996;29:38–47. doi:10.1109/2.485845.

Sandhu R, Bhamidipati V, Munawer Q. The arbac97 model for role-based administration of roles. ACM Trans Inf Syst Secur 1999;2:105–35.

Ser. Apache ServiceMix v4.5.3, <http://servicemix.apache.org/>; 2013 [accessed 14.10.15].

Smart NP, Vercauteren F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Public key cryptography–PKC 2010. Springer; 2010. p. 420–43.

SunXACML. Sun's XACML implementation, <http://sunxacml.sourceforge.net/>. 2015 [accessed 14.10.15].

Takabi H, Joshi JB, Ahn G-J. Securecloud: Towards a comprehensive security framework for cloud computing environments. In Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual (pp. 393–398). IEEE; 2010.

Tang B, Li Q, Sandhu R. A multi-tenant rbac model for collaborative cloud services. In Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on (pp. 229–238). IEEE; 2013.

XML. XML signature syntax and processing, 2nd. ed. <http://www.w3.org/TR/xmldsig-core/>; 2008 [accessed 14.10.15].

Yu S, Wang C, Ren K, Lou W. Achieving secure, scalable, and fine-grained data access control in cloud computing. In INFOCOM, 2010 Proceedings IEEE (pp. 1–9). Ieee; 2010.

Yuan E, Tong J. Attributed based access control (ABAC) for web services. In Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. IEEE; 2005.