2013 IEEE International Conference on Cloud Computing Technology and Science

Access Control and Security Properties Requirements Specification for Clouds' SecLAs

Asma Guesmi
Univ. Orléans, LIFO
EA 4022, F-45067, Orléans, France
Email: asma.guesmi@univ-orleans.fr

Patrice Clemente
ENSI de Bourges, Univ. Orléans, LIFO
EA 4022, F-18020, Bourges, France
Email: patrice.clemente@ensi-bourges.fr

***Abstract*—Current Cloud Service Level Agreements (SLAs) do not cover security requirements. Some consortiums have proposed standards for the evaluation of security offered by the Cloud Providers (CP). Cloud Brokers (CB) can then generate Security Level Agreement (SecLA) contracts between customers and providers to fit users' requirements. However, the SecLAs do not provide enough details for complex customers' situations, such as sharing resources with other users/companies, or set up specific Access Controls and Security Properties (ACSP).**

In this paper, we tackle this issue, by introducing a general Requirement Specification Language (ACSP-RSL) to allow the customers to express their needs in term of ACSP. The underlying formal model, on which is based RSL, is partially presented. The global SecLA definition and negotiation process is thus extended with our proposal. RSL indeed also allows to express Security Requirements currently existing in SecLAs. The negotiation phase between CB and the CPs is discussed. We show how the RSL specifications expressed by the customer can be projected into a generic detection/protection policy expressed as an extension of RSL. A complete use-case for a healthcare system with multi-tenancy for users and services deployed is given. Its security requirements are analyzed, modeled, expressed and discussed.

***Keywords*—Security Level Agreement, Security Requirements, Access Control, Security Properties, Cloud Computing, Cloud Broker.**

I. Introduction

A. Service Level Agreements for Clouds

Cloud computing has become a major computing paradigm in most domains where computing can be used as an on-demand resource. Actually, every computing resources that can be provided and billed as a service, is a potential candidate for a Cloud offering. The services billed to the customer are contracted between the Cloud vendor and the customer under the *Service Level Agreement* (SLA).

As Cloud computing evolves, the integration of multiple Cloud services can be too complex for Cloud consumers to manage. Thus, a Cloud consumer may request Cloud services from a *Cloud broker* (CB), instead of contacting a CP directly. A CB is an entity managing the use, performance and delivery of Cloud services. Generally, a CB provides three categories of services [11]: 1) *intermediation*: improving Cloud capabilities and services for consumers ; 2) *aggregation*: combining multiple services into one or more new ones, between the Cloud consumer and multiple CPs ; 3) *arbitrage*: (elastic aggregation) similar to service aggregation but the services being aggregated are not fixed.

The CB also negotiates the SLAs between CPs and Cloud consumers. The SLA mainly covers the Quality of Service (QoS) and may include specific resources/services. However, most of the time, SLAs do not cover security concerns. At best, they address availability purposes.

B. Security in SLAs

Nevertheless, many security problems arise in Clouds, even at well known and strong providers such as Google and Amazon. The customers on Clouds need to understand what are the risks when migrating their data and services to the Cloud and how the security of their data and services will be maintained. The CPs make efforts to secure their services and they should be able to describe what they can provide in term of security and safety as in term of service, which helps the vendors to better commercialize their Cloud products and convince the customers. But, it remains difficult to know what really the security assurance level guaranteed by each provider is. As a response, the US General Services Administration has set the Federal Risk and Authorization Management Program (FedRAMP). FedRAMP provides a standard framework for assessing and authorizing Cloud service vendor using a standardized approach to security assessment, authorization, and continuous monitoring. Starting from June 2014, governmental agencies will be required to accept only CPs that have been assessed and authorized through the FedRAMP.

Although some previous norms and standards, such as ISO 27000 [13] or the National Institute of Standards and Technology (NIST) -FISMA (Federal Information Security Management Act) [21], that were not intended to be used in the context of Clouds, some authors have tried to adapt them in that objective. Almorsy *et al.* [3] have identified that multi-tenancy on Cloud requires maintaining different security profiles for each tenant on the same service instance. They have proposed a cloud security management framework based on aligning the NIST-FISMA standard [21], as one of the main security management standards, to fit with the cloud architectural model. However, they only consider basic security profiles, such as high-level qualitative evaluation of integrity, confidentiality and availability. Nevertheless, like FedRAMP, those norms and methodologies also lack of end users implication.

Recently, the NIST has proposed a Cloud Computing Security Reference Architecture [23], providing a comprehensive formal model to serve as security overlay to the NIST Cloud Computing Reference Architecture [22]. It also describes a methodology for applying a Cloud-adapted Risk Management Framework using the formal model and an associated set of Security Components (derived from the capabilities identified in the Cloud Security Alliances Trusted Cloud Initiative Reference Architecture [8]) to orchestrate a secure cloud Ecosystem.

As Clouds are becoming the major computing paradigm not only for governmental agencies but for every kind of company, the Cloud Security Alliance [1], has indeed provided standards to question and to evaluate the security of CPs, in order to be able to choose accordingly to the user security requirements. As classical SLAs never contain security requirements, those recent efforts have led to the specification of security statements in Service Level Agreements, also known as Security Level Agreements or SecLAs. SecLAs force the customers and the providers to respect the contract, which decreases significantly security violation and vulnerabilities. SecLA is an important area, several industrial and academic are developing methods to build, specify and quantify Cloud SecLA. Some CPs are creating and storing their SecLAs in public repositories e.g., the CSA STAR [1]. Its Consensus Assessments Initiative Questionnaire (CAIQ) [1] provides industry-accepted ways to document what security controls exist in IaaS, PaaS, and SaaS. It contains 197 questions that a Cloud customer or Cloud auditor would ask, distributed on 11 fields: Compliance (CO), Data Governance (DG), Facility Security (FS), Human Resources Security (HR), Information Security (IS), Legal (LG), Operations Management (OP), Risk Management (RI), Release Management (RM), Resiliency (RS) and Security Architecture (SA).

Many Cloud vendors (Amazon AWS, Red Hat OpenShift, HP Enterprise Cloud Services, Microsoft Office 365, Microsoft Windows Azure and others) have completed this questionnaire to present their security assurance and how to achieve it [2]. The SecLA has the advantage, beyond the legal one, to make understand how security is accomplished. The concurrence between vendors forces them to improve the security of their services and protect customers data.

After the work of Almorsy *et al.* [3], Luna *et al.* [16] proposed metrics to quantify various CPs SecLAs. In [17] Luna *et al.* extends their work by proposing a methodology to provide security benchmarks at different levels of SecLA-granularity (e.g., overall Cloud SecLA or individual security provisions). The benchmarks evaluate the SecLA of one or more CPs with respect to a user-defined requirement. Their work is based on the Cloud Security Alliance "Security, Trust and Assurance Registry" (CSA STAR) repository [1]. Authors in [12] propose a trust management system for Cloud computing. They extract trust information from the STAR repository [2] and identify the trustworthy CPs.

As the current Cloud SLAs are written in natural language and they do not cover security requirements, authors in [19] discuss the possibility to express the security requirements in seven different specification languages, considering a CB that negotiates SLA between customers and providers.

C. Motivations

All work presented above focus on providing a set of well established security assets and provisions for the Cloud customers, at best regarding very high level security needs. For example, the first question of the CSA questionnaire ([1]) (i.e. Question #IS-01.1) of the IS domain (under the control group IS *Management Program*) is the following: "*Do you provide tenants with documentation describing your Information Security Management Program (ISMP)?*". All the security concerns remain at this very high level. Precisely, no attention is given to the specification of precise security properties.

We propose a process in which customers can express their security requirements in terms of Access Control and Security Properties (ACSP), while the CPs can express their related security offers. The paper focuses of the formal model and the ACSP specification language it is based on. It is outside the scope of the paper to present in practice how those specifications are negotiated between customers/brokers and providers. The assurance and the monitoring of those security requirements is also outside the scope of the paper.

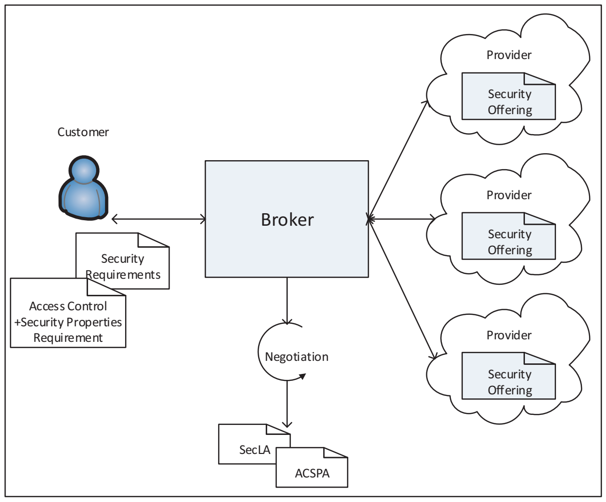

Fig. 1. Negotiation of security requirements

To allow the expression of the security requirements, we model the Cloud computing architectures in order to build a specification language designed to express ACSP requirements. This Requirement Specification Language (ACSP-RSL) can be used to express both Access Control requirements and Security Properties requirements, but also classical security requirements such as those already covered in the CSA STAR questionnaire [2].

Precisely, we use ACSP such as the following. AC refers to precise Access Control constraints (e.g. read access) to resources while SP refers to Security Properties (e.g. integrity, separation of duties, trusted (path) execution, etc.).

As it is difficult for customers to discover by themselves all providers offers, it should be better to perform the discovery and negotiation of service and ACSP requirements by an automatic process like a CB. In the rest of the paper, we thus implicitly consider a CB that negotiates and creates the SecLAs between the Cloud customers and the CPs, considering the customer security requirements and the provider offers.

Considering both customer requirements and providers offers, the CB tries to find the most adequate provider that satisfies the customer needs. The negotiation step ends and the broker creates the Security Level Agreement (SecLA) and (or) the AC and SP Agreements (ACSPA) (Figure 1).

This paper is organized as follows. In section II we will present our use-case: a healthcare system to be deployed on the Cloud. A complete multi-tenancy scenario will be given, along with all required security constraints, including AC requirements, required SP, and also classical SecLA requirements.

In section III, we present how the Cloud customer can express ACSP requirements. We firstly introduce a formal description of the Cloud architecture, compliant with any classical level of Clouds (i.e. IaaS, PaaS, SaaS). We thus describe the RSL and exemplify each AC rule or SP presented on the use-case. We also show how to express classical CSA

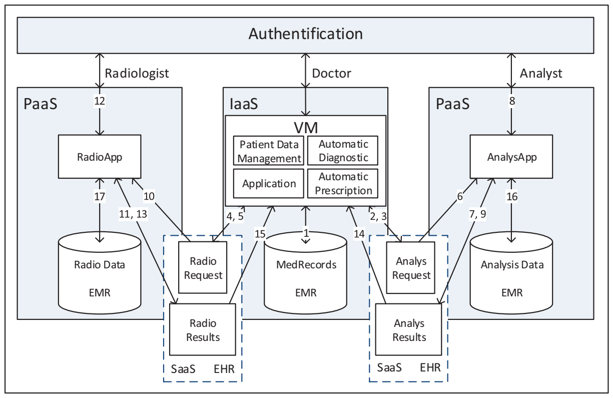

Fig. 2. Use case: healthcare architecture

security requirements with RSL.

Section IV discusses the negotiation steps (conflicts resolution, and effective negotiation), reviews all types of conflicts, regarding the use-case proposed.

Section V describes how to create and enforce the SecLA and the ACSP policy.

II. Use Case: Healthcare architecture

To present our approach, we consider a healthcare system that has to be deployed in a Cloud architecture. The healthcare services and data have thus to be stored and executed from, in and out of the Cloud. The use-case architecture in Figure 2 is an illustrative one, inspired from the LAB Interoperability HUB[1]

The customer needs a VM to manage patients medical records and automatic diagnostic and prescriptions, and 3 separate storage spaces to store medical records, analysis data and radiology data. It also needs to deploy and use 2 PaaS applications, one for the analysis and researches and the other one for the radiology and imaging.

A. Actors in the use case

We consider in this use case:

- 3 user roles: a *doctor*, an *analyst* and a *radiologist*.
- Objects:
  - Storage-Databases: *medRecord* (*Db1* is an EMR[2] to store patient medical records), *Db2* (EMR to store analysis data), *Db3* (EMR to store radio data).
  - Storage: an EHR[3] that contains Radio requests "*radioFolder*" and radio results "*radioFolderRes*", another EHR that contains analysis requests "*analysFolder*" and analysis results "*analysFolderRes*".
  - Applications: *radioApp* (Application for imaging and radiology), *analysApp* (Application for analysis results treatments).
  - Virtual Machine: *VM1* to manage patients medical records and automatic diagnostic and prescriptions.

  See Figure 2.
- Actions: create, read, write, execute, share.

B. Detailed Scenario

When a patient visits a doctor, the doctor creates a *medRecord* for this patient (1) than he writes the patient's personal informations, the diagnostic and prescriptions into the *medRecord* (1). The doctor submit the patient to the analysis service, so he creates an *analysFolder* (2), prescribes the required tests (2) and shares it with the analyst (3). The doctor can also submit the patient to the radiology service so the doctor creates a *radioFolder* (4) and shares it with the radiologist (5).The analyst reads the *analysFolder* to determine the required analysis (6), creates an *analysFolderRes* (7). He performs the analysis using the *analysApp* application (8), writes results into the *analysFolderRes* (9) and shares it with the doctor (9). The radiologist reads the *radioFolder* to determine the required radios (10), creates a *radioFolderRes* (11). He performs and analyzes the radios using the *radioApp* application (12), writes results into the *radioFolderRes* (13) and shares it with the doctor (13). The doctor can read analysis results (14) and radio results (15). The analyst and the radiologist can stock information about what they have performed (analysis (16) and radios (17)).

III. Customer requirements

When the customers decide to deploy their services and store their data on the Cloud, they need to express their requirements in term of services, performances, security requirements, and ACSP because security of data and the access to resources will be completely or partially managed by the provider. We defined an ACSP Requirements Specification Language for Clouds (*ASCP-RSL*, or more simply *RSL*).

A. Basics of ACSP-RSL

Let introduce some formal basics of RSL for Clouds. We refer to each Cloud level as the "system". RSL is based on our generic model[4] of the Cloud, partially presented here, including:

- the set $\mathcal{S}$ of active entities (called **subjects**) on the Cloud composed of:
  1) the set of users $\mathcal{U}$,
  2) the set $\mathcal{A}$ of applications/services executed in the name of any entity $\forall u \in \mathcal{U}$.
     Thus $\mathcal{S} \supseteq \mathcal{U} \cup \mathcal{A}$
- the set $\mathcal{O}b$ of passive entities (i.e. **objects**) on the Cloud, including:
  1) the set of resources $\mathcal{R}es$,
  2) the set $\mathcal{A}$ seen above, as services can also be accessed from other subjects.
     Thus $\mathcal{O}b \supseteq \mathcal{R}es \cup \mathcal{A}$
- the set $\mathcal{R}$ of roles which are attributed to users according to job competency, authority and responsibility.

[1]LAB Interoperability HUB from ViSolve http://www.visolve.com/products/lih/lih-for-radiology.php is an integration platform that enables Laboratories and Imaging Centers to receive orders electronically from multiple physicians using different EMR systems.

[2]Electronic medical records (EMRs) are defined as health-related information on a patient, maintained within a single health care organization (hospital or physician office) [10].

[3]Electronic health records (EHRs) are defined as health-related information on a patient, gathered cumulatively across multiple health care organizations (hospitals, physician offices, clinics, laboratories) [10].

[4]The model of the system finds its roots in the paper of Clemente *et al.* [7], where operations (including binary execution) between subjects and objects can be all abstracted in terms of direct or transitive information flows. Complex SP (e.g. Separation of Duties) are then build on these elementary operations and flows.

For example: the physician, the analyst and the nurse roles.

- the set $\mathcal{A}ct$ of elementary operations (i.e. **actions**) that subjects can perform on objects, i.e. *read like* operations (r), *write like* operations (w), and *execution like* operations (x).
- the set $\mathcal{SP}$ of Security Properties, that are complex operations between $\mathcal{S}$ and $\mathcal{O}b$, built upon $\mathcal{A}ct$, such as: separation of duties, confidentiality (e.g. data confidentiality, Bell&LaPadula), trusted subjects and objects, trusted (path) execution, dynamic SP (e.g., chinese wall, dynamic data confinement[5]), integrity (e.g. integrity of data, integrity of subject, integrity of objects, integrity of domains, non interference, no race condition), etc. For example, the "data confidentiality" SP is the protection against direct and indirect information flow from the confidential object to the unauthorized subject. If the object is an active entity, that means it must not be able to send information to the subject.
- the set $\mathcal{M}$ of management operations, such as granting operations (grant), create objects, share objects, delegate (grant) his own rights, etc. This set $\mathcal{M}$ of complex operations is also built upon $\mathcal{A}ct$. For example, the sharing operations can be seen as mix of *read* and *write* like operation permissions between sets of subjects and objects.
- the set $\mathcal{P}$ of Permissions and directives applied to AC and SP requests, including the following values {allow ,deny, not applicable, enforce, ...}.
- the set $\mathcal{CL}$ of the considered Cloud levels: including the following values {IaaS, PaaS, SaaS, ...}.
- the set $\mathcal{C}txt$ of attribute-value pairs representing the considered situation context. Those pairs are composed of the type of the context and its value. Those types include: {duration, location, historic, ...}. For example, possible values for the duration type could be: { infinite : [] } or {daily : [09:00,11:00]} or {today : [09:00,11:00]}.

B. ACSP Requirements

1) Access Control: The Customers express their AC requirements in *RSL* using the following request:
req(roles, objects, actions, cloudLevels, targetRoles, targetActions, contexts, permissions),
where:
roles,targetRoles $\subset \mathcal{R}$,
objects $\subset \mathcal{O}b$,
actions,targetActions $\subset \mathcal{A}ct \cup \mathcal{SP} \cup \mathcal{M}$,
cloudLevels $\subset \mathcal{CL}$,
contexts $\subset \mathcal{C}txt$,
permissions $\subset \mathcal{P}$.

That means the customer requires that the users endorsing one of the given "roles" must be "allow"ed to perform given "actions" on given "objects" when a given "context" is true, at a given service level. The other attributes are used when requesting a management action, for example: users having "roles" request to "share" "objects" with the users having "targetRoles" with "targetActions" and "permissions".
When an attribute is not applicable we use the "_" symbol in the request (throughout the document) and we use the symbol "\" to express the choice between several possible arguments, and "⋆" stands for all possible values.

The customer who wants to deploy its healthcare system on the Cloud, expresses the following AC requirements:

```
req(doctor, medRecord, create/read/write, IaaS, _, _, _, allow)
req(doctor, radioFolder, create/read/write, SaaS, _, _, _, allow)
req(doctor, radioFolder, share, SaaS, radiologist, read, _, allow)
req(doctor, radioFolderRes, read, SaaS, _, _, _, allow)
req(doctor, analysFolder, create/read/write, SaaS, _, _, _, allow)
req(doctor, analysFolder, share, SaaS, analyst, read, _, allow)
req(doctor, analysFolderRes, read, SaaS, _, _, _, allow)
req(radiologist, radioFolder, read, SaaS, _, _, _, allow)
req(radiologist, radioFolderRes, create/read/write, SaaS, _, _, _,
    allow)
req(radiologist, radioFolderRes, share, SaaS, doctor, read, _, allow)
req(analyst, analysFolder, read, SaaS, _, _, _, allow)
req(analyst, analysFolderRes, create/read/write, SaaS, _, _, _,
    allow)
req(analyst, analysFolderRes, share, SaaS, doctor, read, _, allow)
```

It may be very difficult for some customers to express their security requirements for the access and the use of services/resources. That is why we propose to use more powerful and usable expressions: the SP, which are described below.

2) Security Properties: The AC requirements above could be also expressed in terms of confidentiality and integrity properties. More than that, expressing SP allows the customer to not being a security expert able to precise every required AC rules to ensure its security requirements. For example, in our healthcare system, the customer may desire that nurses should not have "w" access permission to patient's medical records (integrity property of EMRs against nurses). However, if doctors have delegation grant, they could allow nurses to modify the patient's EMR. Thus guaranteeing this property is not feasible only using AC rules. The last arguments for the use of both AC rules and security properties is that they are complementary. One single SP could replace hundred or thousands of classical AC rules. But, it may also go beyond by expressing additional security requirements. For example, AC rules only describe direct accesses, whereas SP intrinsically describe also indirect accesses. For example, using AC rules, it would be impossible to prevent the doctor to add to the patient's EMR an electronic notice written into the system by the nurse. Using SP, this could be prevented.

a) Confidentiality: to protect the information from disclosure to unauthorized parties.

For example, the confidentiality of the object *analysFolderRes* toward *doctor*, expressed in line #7 of the listing above, could be written as the following: req(doctor, analysFolderRes, no-object_confidentiality, SaaS, _, _, ⋆, enforce). That means that the *object_confidentiality* SP is not enforced for the role *doctor*, thus the doctor can read *analysFolderRes*.

By default, all SP are enforced to any object for any subject req(⋆, ⋆, ⋆, ⋆, _, _, ⋆, enforce).

b) Integrity of information: refers to protecting information from being modified by unauthorized parties. For example, the fifth line of the ACSP-RSL listing above req(doctor, analysFolder, create/read/write, SaaS, _, _, ⋆, allow) would result in the following SP request: req(doctor, analysFolder, no-integrity, SaaS, _, _, ⋆, enforce)

To have a more secure architecture for our use-case, we also define the following SPs.

c) Race conditions: when operations must be done in the proper sequence, and a write operation is performed on an object between two other operations. For example, the doctor can not write on *radioFolder* between the operation where he writes on the *radioFolder* and the operation where

[5]Those SP are fully described following the model in [7] in [24].

the radiologist reads from the same resource. Moreover, no one should be allowed to modify *analysFolder* between a modification by the doctor and a read operation by the analyst. Identically, the analyst (radiologist) can not write on the *analysFolderRes* (*radioFolderRes*) between he writes on and the doctor reads from the *analysFolderRes* (*radioFolderRes*):

```
req(*,radioFolder,no_race_condition,SaaS,_,_,_,enforce)
req(*,radioFolderRes,no_race_condition,SaaS,_,_,_,enforce)
req(*,analysFolder,no_race_condition,SaaS,_,_,_,enforce)
req(*,analysFolderRes,no_race_condition,SaaS,_,_,_,enforce)
```

d) Privilege separation: a program is separated into parts with different privileges. For example: separation between the write privilege of *analysApp* and the execution privilege of the same application. Thus we have:

```
req(*,analysApp,privilege_separation,PaaS,_,_,_,enforce)
```

e) Domain Integrity: isolate some entities that share information only between them. We can consider the integrity of a domain constituted by a doctor, a radiologist, an analyst for each patient, all shared data concern only this patient. For example: for the patient "patient#p1", we generate the domain constituted by the doctor "doctor#d1", the analyst "analyst#a1" and the radiologist "radiologist#r1" and the resources "medRecord#p1", "radioFolder#R1", "radioFolderRes#RF1", "analysFolder#A1" and "analysFolderRes#AF1" that are shared only between the member of this domain, the shared information must not disclose outside the domain:

```
req({doctor#d1,analyst#a1,radiologist#r1},{medRecord#p1,
    radioFolder#R1,radioFolderRes#RF1,analysisFolder#A1,
    analysisFolderRes#AF1},domain_integrity,{IaaS,PaaS,
    SaaS},_,_,_,enforce)
```

f) Trusted path execution (TPE): users can execute only trusted applications. For our example, the doctor can execute only *patientDataManagement*, *automaticDiagnostic* and *automaticPrescription* applications, the radiologist can use only the *radioApp* application and the analyst can use only the *analysApp* application. Thus we have:

```
req(doctor,patientDataManagement,tpe,IaaS,_,_,_,enforce)
req(doctor,automaticDiagnostic,tpe,IaaS,_,_,_,enforce)
req(doctor,automaticPrescription,tpe,IaaS,_,_,_,enforce)
req(radiologist,radioApp,tpe,PaaS,_,_,_,enforce)
req(analyst,analysApp,tpe,PaaS,_,_,_,enforce)
```

C. CSA Security requirements

In addition to the ACSP Requirements presented above, we also want to keep the existing state of the art and work on SecLAs and related high-level security requirements. Thus, we extend our model with the set $\mathcal{SR}$ of Security Requirements, which is a set of attribute-value pairs representing the considered SR types and their values (both are strings). For example the SR attribute types set contains: `Identification, Authentication, Secure transport, Encryption, Confidentiality, Integrity, Data leakage prevention, Auditing, Non_repudiation, Localization, Intrusion detection, Separated environments, ...`.

Customers choose their security requirements from the CAIQ [1] and express them using the format: `reqSec(objects, requirements, values)`. The attribute *values* is optional.

For examples, the request `reqSec(Vm, Data Leakage Prevention ,_)`: the customers requires to provide *data leakage prevention* on the Vm. The request `reqSec(Db, Localization, France)` means that *Db* must be stored at the France area. The request `reqSec(*, Encryption,_)`: the customer requests the encryption for all deployed resources and data.

1) Example: According to [9], a healthcare infrastructure requires the following functional and technical requirements:

```
reqSec(*,Identification,_)
reqSec(*,Authentication,_)
reqSec(*,Secure transport,_)
reqSec(*,Encryption,_)
reqSec(*,Confidentiality,_)
reqSec(*,Integrity,_)
reqSec(*,Data leakage prevention,_)
reqSec(*,Auditing,_)
reqSec(*,Non_repudiation,_)
reqSec(vm1/db1/db2/db3,Localization,France)
reqSec(vm1,Intrusion detection,_)
reqSec(radioApp/analysApp,Separate environments,_)
```

IV. NEGOTIATION

Expressing the ACSP requirements in the proposed specification language simplifies the automation of the negotiation process. First and before negotiating services, conflicts between customer requirements are detected automatically by the broker and some corrections are attributed and communicated to the customer who validates these corrections or modifies its requirements. These conflicts can concern the ACSP needs as well as the general security needs.

A. ACSP conflicts

AC rules and SP can be checked for consistency, completeness, redundancy and determinism [18], [15].

- *Modality Conflicts*: When opposite decisions are attributed for the same subjects, objects and actions [20]. To prohibit a role to access to a resource and to permit to a sub-role to access to the same resource.
  To permit a role to access to a folder and to prohibit this role to access to a sub-folder.
- *Redundancy Conflicts*: Assign priorities to access control rules can lead to policies that never apply [20]. For example: if we add the request `req(analyst,analysApp ,no-tpe,PaaS,_,_,daily:[12:00,14:00],enforce)` and we give high priority to positive rules, then the analyst will have the permission to execute *analysApp* any time because the rule related to the request `req(analyst, analysApp,tpe,PaaS,_,_,_,enforce)` is a higher priority than the rule related to the other request. The existence of the positive AC request leads to the non-use of the negative request.
  Another issue is that SP may factorize many AC rules. Thus redundancy is more complex when mixing, as AC rules and SP have to be considered at the same time and for the same purpose.
- *Context* (temporal, spatial and historical) conflicts: `req(doc,medRecord,read,IaaS,_,_,daily:[09:00,11:00], allow)` and `req(doc,medRecord,read,IaaS,_,_,daily :[08:00,11:00],allow)`
  `req(doc,medRecord,read,IaaS,_,_,Paris,allow)` and `req(doc,medRecord,read,IaaS,_,_,France,allow)`
  If we add a system admin (sysAdmin) role and the request `req(sysAdmin,analysApp,write,PaaS,_, _,daily:[08:00,10:00],allow)`, this contradicts the request `req(analyst,analysApp,tpe,PaaS,_,_,_,allow)` because it is possible that the sysAdmin modifies the *analysApp* at the same time when the analyst uses it.
- *Complete absence of rules* for some resources or some roles. The customer must define rules to access all resources.

- *Conflict caused by propagation and role delegation* [14]: When a rule is covered by one or many other rules. If we permit users to share resources and delegate their roles to other users, this may contradict other requirements. Example: req(doctor, radioFolder, share, SaaS, analyst, read, _, _, allow) and req (analyst, radioFolder, read, SaaS, _, _, _, deny). The doctor share radioFolder with analyst with read only permission, but this leads to contradiction with the second request where the analyst is prohibited to read the radioFolder.

B. CSA Security conflicts

The security requirements defined by customers may contain conflicts. As conflicts between security requirements exist, there can also exist conflicts between service requirements and security requirements. For examples:

- A customer that requests a limited memory, disk or instances number and on the other hand requests the elasticity and availability of resources that can extend beyond the threshold.
- A customer requests that its data must be stored at a specific geographical area and requests the disaster recovery of the data which is provided by replication of data to an off-site location.
- A customer requests non-production or the retention of its data and requests a high availability by storing data in different locations.
- A customer requests total privacy of its data witch means that the provider can not access these data and requests that the provider has to detect information leak or intrusions.

All conflicts should be first detected, and then corrected. After the broker and customer agree, the negotiation step starts.

C. Negotiation process

After that all requirements have been corrected and validated by the customer and the broker, the broker starts negotiating service, ACSP and CSA security requirements between the customer and the providers, considering different providers' offers. The technics, methods and algorithms to be developed and used in the negotiation process are also outside the scope of this paper, and may be the main focus of a future paper. Let us just give some short cues about this step.

The negotiation is mainly based on the comparison of the consumer needs and the vendors offers. The negotiation process can end in various (successful or unsuccessful) ways:

- The broker finds a provider that meets all requirements and the client agrees.
- The broker does not find any provider satisfying the command, the negotiation is stopped by a time-out error.
- The negotiation is skipped by the customer.

In the first case, the customer and the provider agree about the command, the requested resources are provided. In the next step, the ACSP policy will be created by the provider and the SecLA by the broker.

V. ACSP policy and SecLA

A. SecLA

The Security Level Agreement is a contract created by the broker, using an XML schema, WS-agreement [4] or any other formalism. It describes what the provider offers in term of ACSP assurance. The SecLA must be respected by the customer and provider, and each violation of this contract leads to penalty which is also described in the same contract.

B. ACSP policy

The chosen provider creates the ACSP policy based on customer requirements. This process can be automated, from the parsing of the "requirements language" introduced below to the production of a generic ACSP policy that can be in turn projected into real world IDS or IPS policies, such as the work presented in [5]. We use the following syntax for expressing access permissions: allow/deny(role, object, action, cloudLevel ,targetRole, targetAction, context)
We use the following syntax to express the SP rules: enforce(role, object, SP/no-SP, cloudLevel,_, _, context).
context is an optional attribute to designate temporal, spatial or historical constraints.

1) Example: For the healthcare use case, the CP may automate the creation of the ACSP policy. Depending of his security facilities and mechanisms, the CP may have to produce a classical AC policy, or a SP policy, or a mix of the two. The easier way would to directly transfer the ACSP requirement described in RSL, into a general ACSP security policy. That way, the first line of the previous listing for AC requirement would simply become the following security rule: allow(doctor, medRecord, create/read/write, IaaS, _, _, _)

But in case of a customer expressing only SP, and the case of the provider having only AC security mechanisms, the ACSP-RSL request: req(doctor, analysFolderRes, no-object_confidentiality, SaaS, _, *, enforce) would be projected into the following AC rule: allow(doctor, analysFolderRes, read, SaaS, _, _, *).

C. Security enforcement

The guarantee, enforcement and monitoring of the security requirements expressed by customers using our ACSP-RSL is outside the scope of this paper. However, we can give at this points some cues about things to be done. The most important thing is that AC rules can bothly be applied by classical host and network intrusion detection systems (resp. HIDS, NIDS). Besides, SP enforcement is quite more difficult, as no much systems are able to handle this. For example, a system such as the one described in [7] could be a good candidate. Thus, further work will deal with that precisely. Also, for other security requirements already identified by the CSA, many can already be handled using existing security devices, mechanisms, algorithms and appliances.

VI. Concluding Remarks

In this paper, we have introduced a specification language, RSL, designed to express AC and SP Requirements for Clouds. RSL has to be used in the general brokering process of contracting Cloud providers for customers regarding their ACSP requirements, expressed with RSL. After having introduced a use-case for a healthcare system on the Cloud, we have presented all the related ACSP requirements in RSL, and shown how they can be used during the negotiation process. Finally, we have seen how to project the agreed RSL specifications onto the providers, as effective security policies. Future works will focus on the writing of the verification algorithms applied on generic SP, and to the verification and conflict resolution of the ACSP and CSA security specifications. The projection of RSL into effective policies could be done, following the process used in [6]. Other work will have to deal with the effective enforcement of the ACSP requirements.

Acknowledgements

The funds of the PhD of Asma Guesmi come from the French Ministry of High Education and Scientific Research. We would like to thank Frédéric Loulergue and Pascal Berthomé for their useful comments and corrections on the preliminary versions of this document.

References

[1] C. S. Alliance. Consensus assessments initiative questionnaire, 2011. https://cloudsecurityalliance.org/research/cai/, accessed on august 2013.

[2] C. S. Alliance. Star registry entries, 2011. https://cloudsecurityalliance.org/star/registry/, accessed august 5th, 2013.

[3] M. Almorsy, J. Grundy, and A. S. Ibrahim. Collaboration-based cloud computing security management framework. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 364–371. IEEE, 2011.

[4] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement), 2003. https://forge.gridforum.org/projects/graap-wg/, accessed on august 2013.

[5] M. Blanc, J. Briffaut, P. Clemente, M. Gad El Rab, and C. Toinard. A Collaborative Approach for Access Control, Intrusion Detection and Security Testing. In W. W. Smari and W. McQuay, editors, *The 2006 International Symposium on Collaborative Technologies and Systems, Special Session on Multi Agent Systems and Collaboration*, ISBN: 0-9785699-0-3, pages 270–278. IEEE Computer Society, 2006.

[6] M. Blanc, J. Briffaut, P. Clemente, M. Gad El Rab, and C. Toinard. A Multi-Agent and Multi-Level Architecture to Secure Distributed Systems. In *First International Workshop on Privacy and Security in Agent-based Collaborative Environments*, page ., Hakodate, Japon, 2006. The Fifth International Joint Conference on Autonomous Agents and Multiagent Systems.

[7] P. Clemente, J. Rouzaud-Cornabas, and C. Toinard. From a generic framework for expressing integrity properties to a dynamic mac enforcement for operating systems. In M. Gavrilova, C. Tan, and E. Moreno, editors, *Transactions on Computational Science XI*, volume 6480 of *Lecture Notes in Computer Science*, pages 131–161. Springer Berlin Heidelberg, 2010.

[8] Cloud Security Alliance. Trusted cloud initiative reference architecture, 2011. https://cloudsecurityalliance.org/wp-content/uploads/2011/10/TCI-Reference-Architecture-v1.1.pdf, accessed on august 2013.

[9] B. Filkins and H. Khiabani. Incident handling in the healthcare cloud: Liquid data and the need for adaptive patient consent management. SANS Institute, 2012.

[10] D. Garets and M. Davis. Electronic patient records: Emrs and ehrs. Healthcare Informatics, 2005.

[11] Gartner. Gartner says cloud consumers need brokerages to unlock the potential of cloud services, 2013. http://www.gartner.com/it/page.jsp?id=1064712, accessed on august 2013.

[12] S. M. Habib, S. Ries, M. Mühlhäuser, and P. Varikkattu. Towards a Trust Management System for Cloud Computing Marketplaces: Using CAIQ as a Trust Information Source. *Security and Communication Networks*, 2013.

[13] International Organization for Standardization (ISO). "ISO/IEC 27000 - Information technology - Security techniques - Information security management systems - Overview and vocabulary," ISO/IEC 27001:2005(E), 2009.

[14] J.-C. Jeon and K.-Y. Yoo. Conflict detection in role-based access control using multiple-attractor cellular automata. In *Intelligent Computing*, pages 533–541. Springer, 2006.

[15] H. Kamoda, M. Yamaoka, S. Matsuda, K. Broda, and M. Sloman. Policy conflict analysis using free variable tableaux for access control in web services environments. In *Proceedings of the Policy Management for the Web Workshop at the 14th International World Wide Web Conference (WWW)*, 2005.

[16] J. Luna, H. Ghani, T. Vateva, and N. Suri. Quantitative assessment of cloud security level agreements: A case study. *Proceedings of the International Conference on Security and Cryptography*, 2012.

[17] J. Luna, R. Langenberg, and N. Suri. Benchmarking cloud security level agreements using quantitative policy trees. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, pages 103–112. ACM, 2012.

[18] S. Mankovskii. Typing for conflict detection in access control policies. *E-Technologies: Innovation in an Open World*, 26:212, 2009.

[19] P. H. Meland, K. Bernsmed, M. G. Jaatun, A. Undheim, and H. Castejon. Expressing cloud security requirements in deontic contract languages. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science, CLOSER*, 2012.

[20] J. Moura. Characterising access control conflicts. In *NMR 2012 - 14th International Workshop on Non-Monotonic Reasoning*, 2012.

[21] NIST. The federal information security management act (fisma), 2002. http://csrc.nist.gov/drivers/documents/FISMA-final.pdf, accessed on august 2013.

[22] NIST. Nist cloud computing reference architecture, 2011. http://www.cloudcredential.org/images/pdf_files/nist%20reference%20architecture.pdf, accessed on august 2013.

[23] NIST. Nist cloud computing security reference architecture, 2013. http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/CloudSecurity/NIST_Security_Reference_Architecture_2013.05.15_v1.0.pdf, accessed on august 2013.

[24] J. Rouzaud-Cornabas. *(in French) Formalisation de propriétés de sécurité pour la protection des systèmes d'exploitation*. PhD Thesis, Université d'Orléans, France, Dec. 2010.