A Cloud Brokerage Solution: Formal Methods Meet Security in Cloud Federations

Salwa Souaf
SICCS
Northern Arizona University
Flagstaff, AZ, USA
salwa.souaf@nau.edu

Pascal Berthomé
INSA Centre Val-de-Loire
LIFO
Bourges, France
pascal.berthome@insa-cvl.fr

Frédéric Loulergue
SICCS
Northern Arizona University
Flagstaff, AZ, USA
frederic.Loulergue@nau.edu

***Abstract*—Cloud Computing has known in the past few years a fast development, leading to a spike in the number of companies competing on providing the best Cloud services. This makes it harder for potential Cloud customers to chose the adequate provider. Despite its wide adoption, many are still hesitant due to the security issues Cloud Computing poses. In this paper, we propose a brokerage solution that formalizes security properties under the form of inter-VM relations, and gives the possibility of setting these security requirements to its customers from the first steps. This solution uses formal methods and the finite model finder KodKod to verify the consistency of the customer's requirements, and to find a placement for his deployment model.**

***Index Terms*—Cloud Brokerage, Cloud federation, Cloud Security, Formal Methods**

I. Introduction

The use of Cloud computing in the past few years have spread widely, attracting many companies to the market who now provide various Cloud services. Many definitions of Cloud computing exist, majority reference the one given by the NIST [1], which defines Cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud computing offers different service models, the three highlighted so far have been: (1) *SaaS*: Software as a Service, where the user buys access to (right to use) an online software ; (2) *PaaS*: Platform as a Service, which offers an environment for the user to develop, test and run his software products ; (3) *IaaS*: Infrastructure as a Service, provides the user with computing, networking and storage resources, it is considered as the most established Cloud service model [2] and it will be the service that we will be focusing on in this work.

In spite of all the advantages, security in Cloud computing [3] is still the main reason stopping more users from integrating the Cloud. More precisely customers are looking for Cloud security *assurance*. Pearson [4] gives several requirements to accountability approaches. First organizations should establish policies that follow some recognized external criteria and that allow for external enforcement, monitoring and auditing. Such a requirement calls for Cloud services certification [5] that would enhance the trust of customers in Cloud solutions. Secondly, organizations should provide *transparency* and mechanisms for customer participation. Moreover the mechanisms used to implement the policies should be documented. Finally, organizations should offer mechanisms for remediation. Techniques to support accountability range from verifiable evidence collection [6] to formal proved mechanism with proof certificates that a security property holds [7].

Another reason discouraging potential customers is the overwhelming number of Cloud service providers. Customers either cannot chose between providers or wish to combine different offers to do so. As an answer to these demands we leverage two concepts: *Cloud brokerage* and *Cloud federation*. These are presented as Cloud interoperability scenarios in [8], where Toosi et al. present in detail the motivations and benefits of Cloud Interoperability. A Cloud *broker* is "an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers [1]." Many brokers have been introduced in the past few years, with the role of matchmaking the customers' functional requirements to the appropriate provider. Very few have tackled the security aspect and none, as far as we know, but [9] have managed to take into consideration personalized security requirements.

A Cloud federation on the other hand, is a collaboration between multiple providers. Many benefits arise from the aspect of Cloud federation, such as: QoS improvement, reduction of Service Level Agreement (SLA) violations, cost efficiency and more, all discussed in detail in [10]. Despite the benefits of using multiple providers, it is not a common practice in current Cloud solutions due to the many challenges Cloud interoperability is still facing. A spectrum of the obstacles and challenges facing the Inter-Cloud realization is presented in [8].

Our work integrates these two aspects: cloud security assurance and the variety of cloud offers. We present a cloud broker that will guide a customer through the whole process of integrating the Cloud. Our broker takes into consideration the functional (resources) and non-functional (security properties) requirements. Our solution is *transparent* as the users describe their security requirements. Moreover the mechanism we propose can be easily documented as its foundations rely on an easy to understand formalism: first order relational logic. Our

978-1-5386-7879-4/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCS.2018.00113

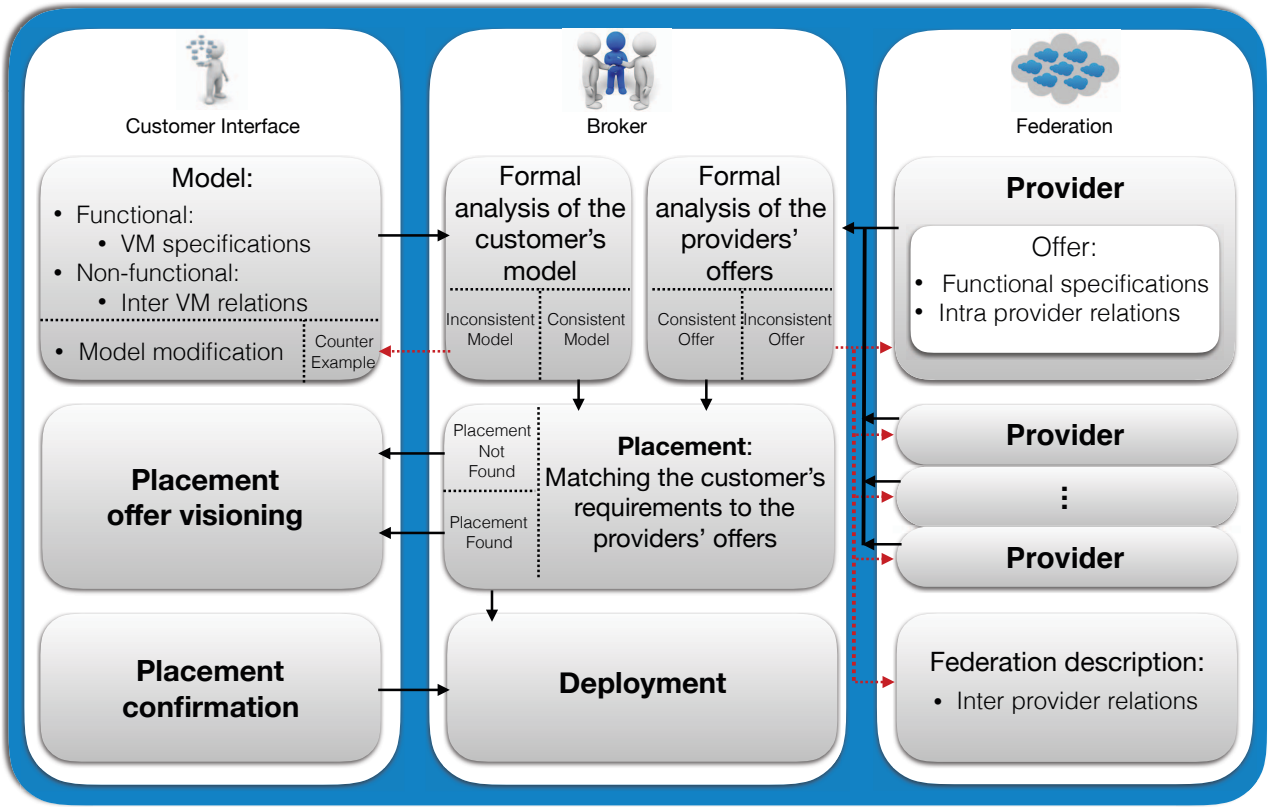


Figure 1: General Architecture of the Brokerage Solution

system verifies the coherence of the user's demand and sends a counter example in case of an inconsistency. We use formal methods to find the appropriate placement in a federation of clouds. After communicating the placement found, if any, to the customer, the latter will decide if he wants to go through with this offer, if so, the broker will deploy the customer's model. Finally, we give the possibility to the customer to modify or update his architecture even after its deployment.

The rest the paper is organized as follows. Section II introduces some of the security issues related to cloud computing. In Section III we present the formal model and our approach to some of the security issues mentioned before. Section IV describes our brokerage solution, its general architecture and the functionalities of the broker. Section V is devoted to the current prototype implementation. A comparison of our solution with related work is presented in Section VI. We conclude in Section VII.

II. Security Issues

Although Cloud computing has brought up many benefits to the market, many security and privacy concerns hinder it from being fully adopted. Cloud computing utilizes many traditional as well as novel technologies, each causes some specific Cloud security issues. There are different Cloud concepts, to mention: virtualization, Cloud platforms, data outsourcing, data storage standardization and trust management. All these concepts were discussed in [3], [11]. In this section we focus on the security issues related to IaaS.

Multiple IaaS security challenges and threats exist [2], [12]. The issues that we try to cover are those related to virtualization properties. In [13] many security vulnerabilities, attacks and threats in virtualization security have been pointed out. The authors categorize these threats and attacks against virtualized systems according to the different layers (i.e. hardware, virtualization, OS and application). They also present some of the known attacks against virtualized environments at the different levels. In our project we try to limit the damages of the attacks against the virtualization layer, to mention:

- Cross-VM Attacks: attacks between virtual machines (VMs),
- VM Escape and VM Hopping: where attackers run a malicious code on the VM that would break the operating system giving them direct access to the hypervisor,
- VM Detection: when attackers detect that the targeted system is running inside a VM and start targeting their attacks on the virtualization layer instead of the application or OS.

In the survey [13], a framework for threat models categorization has been proposed. When trying to propose a new protection solution, this framework can be useful for a more precise definition of the security and trust assumptions. Our work will especially help limit the damage of the *Cross-*

$PNCPU$	
$node_1$	Intel-1-2.0Ghz
$node_2$	Intel-1-2.0Ghz
$node_3$	Intel-1-2.4Ghz
$node_4$	Intel-1-2.4Ghz

$PhysicalNode$
$node_1$
$node_6$

(a) Relation and Set Examples

$$\forall (a,b) \in \mathbb{N} \times \mathbb{N},\ b > 0 \implies$$
$$\exists! (q,r) \in \mathbb{N} \times \mathbb{N},\ a = b \times q + r \wedge r < b$$

(b) Formula Example

Figure 2: First Order Relation Logic Examples

VM Attacks and the exploits taking advantage from inter-VM communications. By giving the customers the possibility to define their security properties under the form of inter-VM relations, they will be able to control where they put their most critical data as well as the type of connections between their VMs. Thus, with a robust security requirements definition, a malicious attacker won't be able to access the critical data. The providers will also be defining the security properties they offer in the form of relations linking their clusters internally and other relations linking their clusters with other providers' clusters. Next section provides a detailed presentation of our approach.

III. The Model

One of the aspects of our solution consists in considering the personalizing of the security of the customers' models from its first description. Our security approach consists in formalizing the security properties under the form of communication relations, from both the customer and provider sides.

a) A Model of IaaS Architecture: IaaS is a cloud computing model where the cloud provider manages server hardware, virtualization layers, storage, networks. The way we chose to simplify the presentation of an IaaS provider's architecture is by considering the provider to be presented by a set of *clusters*, each cluster contains a set of *physical nodes*. Physical nodes host the customers' virtual machines, and are defined by their geographic location and a set of VM templates.

In our system the providers will describe their offers in two steps. First step will consist of describing the architecture: the number of *clusters*, the number and characteristics of *physical nodes* in each of them. Second step will be defining the relations between the clusters, intra-provider and inter-provider. We have defined three relations as follow: *Link*: two clusters related by a link or a information flow permitting data interchange; *Guardian*: two clusters having a intrusion protection system and filtering systems, such as network firewalls, relating them; *Conflict*: conflicted clusters, are clusters that cannot have any communication flow relating them.

b) Customer Security Requirements: The customer describes the architecture she would like to deploy. This architecture first includes functional requirements: the number of virtual machines and their characteristics (e.g., OS, memory). It also includes the non-functional/security ones, the relationships between the nodes (VM).

The relations that have been implemented so far in our solution are the following, but we are considering adding more relations in order to have better specifications further down the line: (a) Isolation: an isolated VM can not communicate with the others; (b) Collaboration: collaborating VMs are allowed to share data between them; (c) Concurrence: two concurrent VMs are unauthorized to have a communication flow direct or indirect relating them; (d) Unidirectional Flow: one-way flow between two VMs, meaning one has the right to send data to the other one but not read any.

c) Formal Notations: Our brokerage solution is based on using formal methods. More specifically we use the KodKod model finder that is provided as a Java API. We will give some excerpts of the implementation of our solution using KodKod in Section V. However for presenting how we formalized the broker offers and the customers requirements, a more abstract and concise notation is needed. The formal foundation of KodKod is first-order relational logic. Alloy [14] proposes a syntax for such a logic, and the notations given below are very close to Alloy syntax: the only difference is that we use usual mathematical notations.

First we consider that all the objects we use are sets. In particular an element is identified with the singleton set that contains only this element. $\in$ therefore denotes *set inclusion* as well as set membership.

For sets A_i, $1 < i \leq n$, $A_1 \times \ldots \times A_n$ denotes the set of tuples of arity n. We write such a tuple $(a_1, \ldots, a_n)$ where for all i with $0 < i \leq n$, a_i belongs to A_i.

Subsets of $A_1 \times \ldots \times A_n$ are relations of arity n. For example $<$ is a binary relation on natural numbers and can be seen as the subset of $\mathbb{N} \times \mathbb{N}$ defined as $\{(x,y) \in \mathbb{N} \times \mathbb{N} \mid x < y\}$.

For $n = 1$, a unary relation is just a subset. As it may not be a strict subset, any set is a unary relation. Thus from now on we consider that all the objects we manipulate are *relations*.

To state properties about relations, we write logical formulas. Such formulas are built using basic predicates such as $\in$ or $=$, and using usual logical connectors: $\wedge$ is logical conjunction, $\vee$ is logical disjunction, $\implies$ is logical implication, $\iff$ is logical equivalence, as well as logical quantifiers: $\forall$ is universal quantification, $\exists$ is existential quantification, $\exists!$ is existential quantification with a unique element. For example the formula if Figure 2b states the existence of a unique quotient and reminder for the division of a number a by a non-zero natural number b.

There are several operations to manipulate relations: relation union is denoted by $\cup$, relation intersection is denoted by $\cap$, relation difference is denoted by $-$, relation navigation is denoted by ".". Union, intersection and difference have their usual meaning.

Navigation is similar to a relational join, but the relation join operator joins the relations by column and the matching column is kept. The navigation operation joins the last column of the first relation with the first column of the second relation and drops the matching column.

$CPU = \{\ CPU_2,\ CPU_{24},\ CPU_3\ \}$ $Location = \{\ \text{EU},\ \text{USA}\ \}$ $Customer = \{\ Customer_1\ \}$	$OS = \{\ \text{Linux},\ \text{Windows}\ \}$ $VM = \{\ Vm_1,\ Vm_2,\ Vm_3,\ Vm_4,\ Vm_5\ \}$
$PhysicalNode = \{Pn_1^1,\ Pn_1^2,\ Pn_2^1\ \}$ $Cluster = \{\ Cluster_1,\ Cluster_2,\ C_1^1,\ C_1^2,\ C_1^3,\ C_2^1,\ C_1^2,\ C_3^1,\ C_3^2,\ C_3^3\ \}$ $Provider = \{\ Provider_x,\ Provider_1,\ Provider_2,\ Provider_3\ \}$	

Figure 3: Example Sets

For example if $PNCPU$ is a relation between physical machine identifiers and CPU description and $PhysicalNode$ is a set of physical machine identifiers as shown in Figure 2a, using navigation, the expression $PhysicalNode.PNCPU$ denotes the the relation containing only the value Intel-1-2.0Ghz.

Finally, the transitive closure of R is denoted by R^+, i.e, if $(x, y) \in R$ then $(x, y) \in R^+$; and if $(x, y) \in R$ and $(y, z) \in R^+$ then $(x, z) \in R^+$.

IV. A Brokerage Solution

A. Broker General Architecture

The general architecture of our brokerage solution is presented in Figure 1. It can be described as three phases.

The customer describes his model using a special interface and the provider describes his offer, as mentioned in Section III.

We use formal methods to verify the consistency of both the customer's demand, the provider's offer and the coherence of the federation. In case of an inconsistency, the broker returns a counterexample, highlighting the reason of inconsistency.

Once the customer's model is consistent, the broker will then try to find one possible placement, satisfying both the functional and non-functional requirements. Once a placement found, if any, the broker would forward the suggestion to the customer, and after his approval the broker would deploy the architecture.

a) Basic Sets: The following relations are some of the basic sets and relations that will be used further in this section. These are defined in KodKod as unary relations:

- CPU: contains all the values of CPU characteristics (in our current setting only speed) which means it is a unary relation that has the name "CPU" and that will be seen as a set of values,
- OS: contains all the possible operating systems,
- $Location$: contains all the possible locations,
- VM: contains all the virtual machines identifiers present in the system,
- $Customer$: contains a set of all customer identifiers,
- $PhysicalNode$: contains the identifiers of all the providers' physical nodes,
- $Cluster$: contains the identifiers of all the providers' cluster,
- $Provider$: contains a set of all the provider identifiers,

If we combine the examples given in Figures 4a, 4b and 4c, the previous unary relations would be defined as in Figure 3.

B. Provider Offer and Federation Verification

1) Provider Offer Description: A provider offers a set of clusters, each cluster is made of physical nodes. For the sake of conciseness, we do not formally define the relations between $Provider$and $Cluster$, and $Cluster$and $PhysicalNode$.

A physical node is characterized by its locations and the different characteristics of VM that it is able to host (basically OS and the virtual CPU characteristics):

- $PNOS$: A binary relation between the sets $PhysicalNode$ and OS,
- $PNCPU$: A binary relation between the sets $PhysicalNode$ and CPU,
- $PNLocation$: A binary relation between the sets $PhysicalNode$ and OS.

The placement of a virtual machines on physical nodes is modeled as $PNVM$, a binary relation between $PhysicalNode$ and VM.

The provider describes the relations, $Link$, $Guardian$ and $Conflict$, between his own clusters as well as between his clusters and the other providers' clusters within the same federation. $Link$, $Guardian$ and $Conflict$ are binary relations, subsets of $Cluster \times Cluster$. Ours system makes sure that when a provider adds a new pair (c, c') in one of these relations, then at least one of c and c' is one of his own cluster.

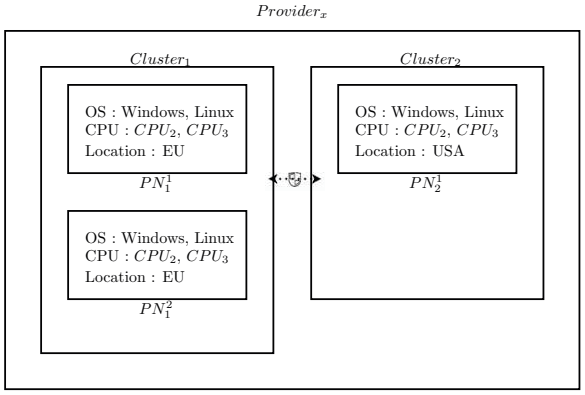
An example of a provider model is presented in Figure 4a. In this case only one relation was defines as follows: $Guardian = \{\ (Cluster_1,\ Cluster_2)\ \}$.

2) Federation description: The accumulated provider descriptions form a model of a federation. The aim of the latter is to have more interoperability between cloud providers, and prevent the issue of provider lock-in (i.e. a customer depending on one and only provider). Figure 4b presents an example of a small federation containing three providers: $\{\ Provider_1,\ Provider_2,\ Provider_3\ \}$. In this example the relations grouping the different clusters are as follow:

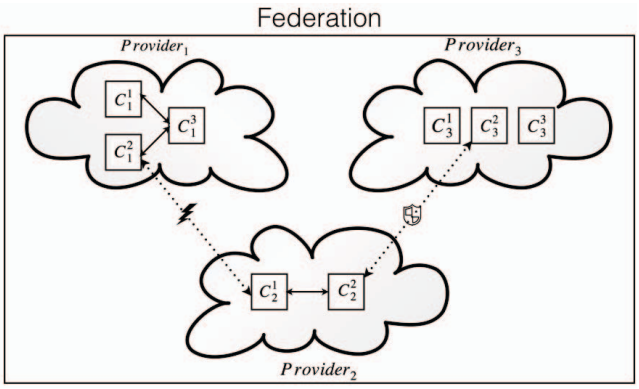
- $Conflict = \{\ (C_1^2, C_2^1)\ \}$,
- $Link = \{\ (C_1^1, C_1^3),\ (C_1^2, C_1^3),\ (C_2^1, C_2^2)\ \}$,
- $Guardian = \{\ (C_2^2, C_3^2)\ \}$.

3) Offer and Federation Consistency Verification: In order for an offer to be consistent or a federation to be coherent, one major rule should be respected.

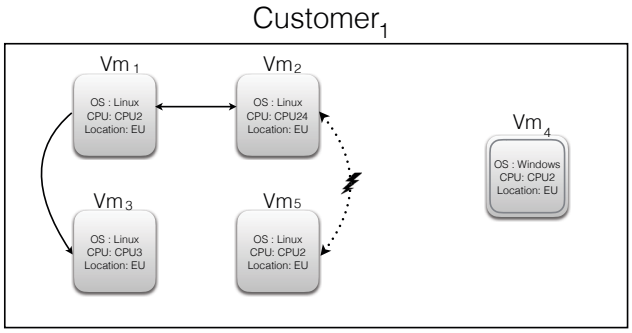
a) Rule 1: Two conflicted clusters can not have a link of any sort relating them (i.e. if two clusters are identified as conflicted the virtual machines hosted on them cannot



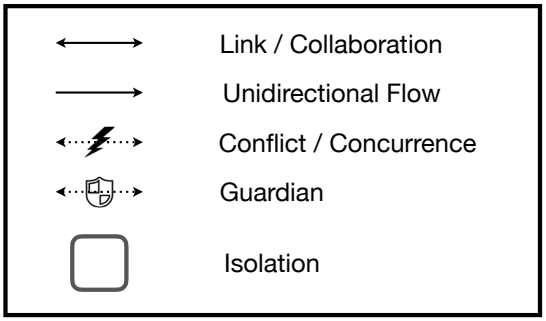
(a) Provider Model Example



(b) Federation Model Example



(c) Customer Model Example



(d) Legend

Figure 4: Model Example

communicate, meaning that no communication tunnel, direct or transitive, can connect these two clusters):

$$\forall c_i, c_j \in Cluster, (c_i, c_j) \in Conflict \implies ((c_i, c_j) \notin (Link \cup Guardian)^+ \wedge (c_j, c_i) \notin (Link \cup Guardian)^+)$$

A counterexample is sent to the provider in case of an inconsistency. Once the offer is consistent, the latter will be added to the federation.

C. Customer Model Description and Verification

1) Customer Model Description: In our system, a customer is identified by an element of $Customer$, and a relation $CustomerVM$ between $Customer$ and VM defines the virtual machines she owns.

The customer's requirements for virtual machines are expressed as their CPU speed, OS and location. This is formalized by three relations: $VMCPU$ a binary relation between VM and CPU, $VMOS$ a binary relation between VM and OS, and $VMLocation$ a binary relation between VM and $Location$.

To define her applications, a customer then defines the relations $Isolation$, $Collaboration$, $Concurrence$, and $Flow$, informally described in Section III. $Isolation$ is a subset (unary relation) of VM, while $Collaboration$, $Concurrence$, and $Flow$ are binary relations between VM and VM. The system ensures that a customer only defines or modifies relations involving her own virtual machines.

An example of a customer model is presented in Figure 4c. The customer's non-functional requirements are:

$$\begin{aligned} Isolation &= \{\ Vm_4\ \} \\ Flow &= \{\ (Vm_1, Vm_3)\ \} \\ Collaboration &= \{\ (Vm_1, Vm_2)\ \} \\ Concurrence &= \{\ (Vm_2, Vm_3)\ \}. \end{aligned}$$

2) Consistency Verification: Verifying the consistency of the customer's requirements is verifying that it respects a set of rules related to the security relations.

a) Rule 2: An *isolated* VM cannot have any information flows coming into/out from it:

$$\forall v_i \in Customer.CustomerVM, v_i \in Isolation \implies \nexists v_j \in Customer.CustomerVM, (v_i, v_j) \in Flow \vee (v_j, v_i) \in Flow$$

b) Rule 3: Two *collaborating* VMs have a bidirectional information flow, they can read and write data from one to another:

$$\forall v_i, v_j \in Customer.CustomerVM, (v_i, v_j) \in Collaboration \implies ((v_i, v_j) \in Flow \wedge (v_j, v_i) \in Flow)$$

$$
\begin{array}{lcl}
CustomerVM & = & \{(Customer_1, Vm_1), (Customer_1, Vm_2), (Customer_1, Vm_3)\} \\
VMOS & = & \{(Vm_1, Linux), (Vm_2, Windows), (Vm_3, Linux)\} \\
VMCPU & = & \{(Vm_1, CPU_3), (Vm_2, CPU_2), (Vm_3, CPU_3)\} \\
PNCPU & = & \{(Pn_1^1, CPU_2), (Pn_1^1, CPU_{24})\} \\
PNOS & = & \{(Pn_1^1, Linux), (Pn_1^1, Windows)\} \\
PNLocation & = & \{(Pn_1^1, USA)\} \\
VMLocation & = & \{(Vm_1, EU), (Vm_2, USA), (Vm_3, EU)\} \\
PNVM & = & \{(Pn_1^1, Vm_2)\}
\end{array}
$$

Figure 5: Example: Relations Values

c) Rule 4: Two *concurrent* VMs cannot have any direct or transitive information flow relating them:

$$\forall v_i, v_j \in Customer.CustomerVM, (v_i, v_j) \in Concurrence \Rightarrow (v_i, v_j) \notin Flow^+ \wedge (v_j, v_i) \notin Flow^+$$

Once the brokerage system receives the customer's model, it verifies that her requirements respects the rules defined. In case of an inconsistency of the demand, a counter-example is sent for the customer to rectify her model. Otherwise the broker proceeds to the next step: finding a placement which is detailed in Section IV-D.

D. Placement Strategy

The brokerage system searches for a placement that satisfies the customer's functional and non-functional requirements, taking into consideration the relations defined between the clusters. It tries to find a placement respecting the following rules.

1) Rules related to the functional requirements:

a) Rule 5: A VM should be placed in a physical node offering the adequate VM characteristics and that is located in the requested location:

$$\begin{array}{c}
\forall v \in VM, \\
\exists p \in PhysicalNode, v \in Customer.CustomerVM \\
\wedge v \in p.PNVM \iff v.VMOS \in p.PNOS \wedge \\
v.VMCPU \in p.PNCPU \wedge \\
v.VMLocation = p.PNLocation
\end{array}$$

For a better understanding of the formula, we present a concrete example. Let us consider the values, given in 5, for the relations mentioned in the formula above, along with those given in Figure 3.

Considering we have only one customer $Customer_1$ and one physical node Pn_1^1, the navigations will contain the following values: $Customer_1.CustomerVM = \{ Vm_1, Vm_2, Vm_3 \}$ and $Pn_1^1.PNVM = \{ Vm_2 \}$.

Vm_2 is both in $Customer.CustomerVM$ and $PhysicalNode.PNVM$, which means that the virtual machine Vm_2 can be placed in the physical node Pn_1^1, if and only if additionally Vm_2's characteristics are offered by this physical node. As we have:

$$
\begin{array}{lcl}
Vm_2.VMOS & = & \{Windows\} \\
Pn_1^1.VMOS & = & \{Linux, Windows\} \\
Vm_2.VMCPU & = & \{CPU_2\} \\
Pn_1^1.VMCPU & = & \{CPU_2, CPU_{24}\} \\
Vm_2.VMLocation & = & \{USA\} \\
Pn_1^1.PNLocation & = & \{USA\}
\end{array}
$$

all Vm_2 characteristics are handled by Pn_1^1.

b) Rule 6: A VM should be placed in one and only $PhysicalNode$, all the customer's *VMs* should be placed:

$$\begin{array}{c}
\forall v \in VM, \exists! p \in PhysicalNode, \\
v \in Customer.CustomerVM \wedge v \in p.PNVM
\end{array}$$

2) Rules related to the non-functional requirements:

a) Rule 7: A customer's VMs cannot be placed on *conflicting* clusters:

$$\begin{array}{c}
\forall v_i, v_j \in VM, \exists c \in Customer, \exists c_i, c_j \in Cluster, \\
\exists p_i, p_j \in PhysicalNode, v_i \in c.CustomerVM \wedge \\
v_j \in c.CustomerVM \wedge v_i \in c_i.p_i.PNVM \wedge \\
v_{ij} \in c_j.p_j.PNVM \Longrightarrow (c_i, c_j) \notin Conflict
\end{array}$$

b) Rule 8: An *isolated* VM should be placed in a separate physical node:

$$\begin{array}{c}
\forall v_i, v_j \in VM, \exists c \in Customer, \exists p \in PhysicalNode, \\
v_i \in c.CustomerVM \wedge v_j \in c.CustomerVM \wedge \\
v_i \in p.PNVM \wedge v_i \in Isolation \iff v_j \notin p.PNVM
\end{array}$$

Currently a customer's isolated VM will only be separated from his own VMs, but could be placed with other customers' VMs in the same physical node. We aim to improve our current program by giving the customers the possibility to specify, for each of their VMs, whether they want place a VM in a same physical node as other customers or not.

c) Rule 9: Two *collaborating* VMs should be on the same cluster:

$$\begin{array}{c}
\forall v_i, v_j \in VM, \exists c_i, c_j \in Cluster, \\
\exists p_i, p_j \in PhysicalNode, v_i \in c_i.p_i.PNVM \wedge \\
v_{ij} \in c_j.p_j.PNVM \wedge (v_i, v_j) \in Collaboration \Longrightarrow c_i = c_j
\end{array}$$

```
public Formula Isolation_rules(){
  final Variable vm_i = Variable.unary("vmi");
  final Variable vm_j = Variable.unary("vmj");
  final Formula F1 = Customer.c_relation
    .product(vm_i).in(isolation).not()
    .and(Customer.c_relation.product(vm_j)
    .in(isolation).not());
  final Formula iso = Customer.c_relation
    .product(vm_i).product(vm_j)
    .in(vm_flow).implies(F1).forAll(vm_i
    .oneOf(VM.vm).and(vm_j.oneOf(VM.vm)));
  return iso;
}
```

Figure 6: KokKod Example: Formula

d) Rule 10: Two *concurrent* VMs should be placed on non-communicating clusters, or clusters linked by a *guardian*:

$$\forall v_i, v_j \in VM, \exists c_i, c_j \in Cluster, \exists p_i, p_j \in PhysicalNode,$$
$$v_i \in c_i.p_i.PNVM \wedge v_{ij} \in c_j.p_j.PNVM \wedge$$
$$(v_i, v_j) \in Concurrence \implies ((c_i, c_j) \notin Conflict \wedge$$
$$(c_i, c_j) \notin Link \wedge (c_i, c_j) \notin Guardian) \vee (c_i, c_j) \in Guardian$$

e) Rule 11: Two VMs related by a *unidirectional flow* should be placed on clusters linked by a *guardian*:

$$\forall v_i, v_j \in VM, \exists c_i, c_j \in Cluster, \exists p_i, p_j \in PhysicalNode,$$
$$v_i \in c_i.p_i.PNVM \wedge v_{ij} \in c_j.p_j.PNVM \wedge (v_i, v_j) \in Flow$$
$$\implies (c_i, c_j) \in Guardian$$

V. Implementation

We have managed to develop a Cloud broker performing all the described functionalities in one JAVA application, using the KodKod API for formal analysis. The workflow of the application follows the general architecture of the broker described in Section IV-A.

KodKod [15] is a finite model finder, that we used for both the consistency verification and the search for placement strategies. These functionalities were transformed into KodKod *problems*, where the *relations* group our system's actors and relations previously mentioned, the *universe* consisted of all the submitted values, the rules previously described were translated into KodKod *formulas*.

a) Relational variables: As previously mentioned everything is a relation in KodKod, and so all the actors of our system are defined as relations. In this section we show some of the steps for the customer's model verification example using the KodKod API. We start by defining the relational variables, using the different methods of the class *Relation* in order to precise the *arity* of the relation and the *name* we want to be assigned to it:

```
Relation Customer = Relation.unary("Customer");
Relation c_vms = Relation.binary("c_vms");
```

```
TupleFactory f = U.factory();
Bounds b = new Bounds(U);
TupleSet vms = f.noneOf(1);
for(String id: VMIds)
      vms.add(f.tuple("Vm"+C+"-"+id));
b.boundExactly(vm, vms);
```

Figure 7: KokKod Example: Relation Bounding

b) Universe: In this case the *atoms* would be the different identifiers of: the customer's, the set of his virtual machines and a VM relation, relating the customer to the relations he have described. And so we use the constructor of the class *Universe* giving it as argument a list of the values in string format (i.e. `Set<String>`): `Universe U = new Universe(atoms);`

c) Formula: The consistency rules previously described are written in a KodKod syntax using the class *Formula*. An example of the *contraposite* of the isolation rule (Rule 2) in KodKod syntax is given in Figure 6.

Then we bound the relational variables with tuples created from the universe, using a tuple factory, and then we just bound the VM relation as an example as shown in Figure 7. In this case we use the method *boundExactly* of the class *bound* and so we are only giving the *lower bounds* (i.e. the relation should contain all of these values). We could assign both the *upper* and *lower bounds* using the method: `bound(Rel, low, up)` or just the *upper bounds* using: `bound(Rel, up)`.

In the case of verification we have tightly *bound* our *relational variables* (i.e. both lower and upper bounds) creating a *model* and used KodKod to insure that the model created satisfied the *formulas* (consistency rules) wanted. In case of an inconsistency KodKod returns the *unsatisfied formula*. In order to return a helpful *counter example* to the customer (resp. provider), we try to find a *solution model* for the *contraposite* of the returned formula, this way we know the source of the inconsistency and forward it to the customer (resp. provider).

On the other hand, for finding a placement strategy, we have loosely *bound* the *relational variables* (i.e. we only give the upper bounds) and asked KodKod to find a *model* that satisfies the placement rules (*formulas*).

KodKod's *partial solution* is an interesting feature that have permitted us to add a dynamic aspect to our project. The collection of all the variable *lower bounds* is a *partial solution* of the problem. Our broker allows the customer to modify his architecture in time: he could either add virtual machines or remove them, modify the virtual machines relations as he desires, and the broker will modify the placement strategy of the concerned virtual machines respecting the old placement of the other ones. Same thing for the provider's offer. In order to do so, we use the existing placement strategy as lower bounds of *modified problem*. Finally we choose the solver to use to solve the conjunction of all the formulas to verify.

d) Example: In order to have a concrete idea of the time spent to verify the consistency of a customer's architecture

and find an appropriate placement in a federation, we have used the following test as a proof of concept. Meanwhile, we are still working on the scalability of our solution in order for it to be used in real life scenarios. We started by using a federation grouping four providers, respectively, composed of $\{11,\ 12,\ 32,\ 308\}$ physical node. We first tried to find placements for three different customers in this federation. The customers are: C_1 with 12 virtual machine and 10 security properties, C_2 with 30 virtual machine and 15 security properties and C_3 with 100 virtual machine and 40 security properties. The consistency of C_1's architecture was verified in 128 ms and the placement problem was solved in 20 ms. The consistency of C_2's architecture was verified in 188 ms and the placement problem was solved in 175 ms. The consistency of C_3's architecture was verified in 682 ms and the placement problem was solved in 22.5 s. Then we added two providers composed respectively of 535 and 957 physical nodes. We found the placement for a customer with 100 VMs in 2.5 min and another with 200 VMs in 5.6 min. Which showed the limitation of using KodKod due to the time spent in the translation, from KodKod problem to CNF, step.

VI. Related Work

Many works have tackled the subjects *Cloud brokerage* and *Cloud federations*, but very few have worked on the combination of the two. On one hand, many of the existing brokers [16]–[18] tend to consider: cost, performance, storage capacity, *etc...* Few, such as CloudSurfer [19] and some SLA based brokers mentioned in detail in [20] that consider security requirements but only at a higher, non formal, level (i.e. the security features that providers offer). On the other hand, there are projects that have showed interest in the cloud brokering concept, to mention MODAClouds [21]. The main goal of this project is to provide cloud customers with tools for developing and deploying applications on multi-Clouds. In fact, such interoperability is becoming more and more possible with the emergence of new standards such as the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [22]. But, as far as we know, no broker but [9] considers personalized and formal security properties.

Having formal requirements can be the basis of a strongly verified system for cloud brokering. In particular we will consider proving the correctness of the principles of our brokering system using the Coq proof assistant [23] based in part on an automated translation [24] of the formula described in this paper. It can be the basis of a fully verified broker implementation using Coq: this kind of verified software is one of the most reliable form of software. For example the full functional correctness of the kernel of an operating system [25], or a C compiler [26] have been established. The authors of a study on compiler testing [27] were indeed unable to find any bug in formally proved component of the CompCert compiler, while it found several "release blocking" bugs in mainstream compilers such as GCC or clang.

The work in [9] is a good starting point, but it has many limitations as well. On one hand, the authors have managed to consider the non-functional properties and verify the consistency from the early steps, in order to prevent any problems when trying to find the placement. They used the language and analyzer [14] for the formal analysis. On the other hand, the sizes of the problems they could handle were very limited, but the major set back was that it was impossible to handle the dynamicity in the customer's demand, once it was submitted to the broker the customer could not modify or upgrade his architecture, and same goes for the provider's offer.

First, we found that the previous model was lacking an explicit definition of the *provider* and *federation* models, and so, we have introduced these new models with an additional step of verifying the consistency of the providers' offers and coherence of the enclosing federation. Secondly, we had to modify the *customer*'s model because of the way it was modeled made it impossible to create multiple *instances*. The use of the ALLOY language and analyzer was the main reason for the lack of dynamicity in the old prototype. In fact, the concept of overriding an object does not exist in ALLOY, and so we have decided to change the tool to KODKOD [15] that thanks to its partial solution, helped us create a more dynamic prototype.

Our customer system modeling approach is different from the existing Cloud Modeling Languages (CMLs), a comparison and classification of existing CMLs in presented in detail in [28]. While CMLs are used in the process of developing or migrating specific applications, we try to model a system in on a higher level.

In our verification approach, we recover the customer's architecture and we try to check if it respects the properties described in the brokering system. In a similar fashion, but purely theoretical and without any implementation, Chong and Meyden have proved in [29], that an architecture can provide sufficient structure to prove that a given information security property holds in all machines that comply with this architecture.

On a different note, not related to security but using formal methods in Cloud Computing. In [30] Challita et al. tackle the semantic differences between cloud providers by proposing a formal-based framework for semantic interoperability in multi-clouds, FCLOUDS. In [31] Mezni and Sellami rely on the mathematical foundation of Formal Concept Analysis to provide a solution to the problem of multi-cloud service composition/ selection. They aim to offer optimal service compositions with a minimal number of providers, in order to reduce communication costs and financial charges, and a short execution time.

VII. Conclusion and Future Work

In this paper, we present an improvement of the cloud brokerage solution presented in [9]. Our system takes into account the functional and non-functional requirements of a cloud customer. It verifies the consistency of both the customers' demand and providers' offers, and finds, if possible, an adequate placement of the customer's model in a cloud

federation. We have managed to introduce a dynamic aspect to the solution by using a finite model finder called KODKOD.

However, even though our solution is more dynamic, in the way that we can create multiple customers and providers with the possibility of modifying their architectures when wanted, we still have some paths to explore in order to improve our solution and add more dynamic features. We are planing on improving the scalability of the solution in order to support larger problems, as well as making it more robust and user friendly.

There are several ways in which the proposed solution could be improved: the security issues related to the centralization of information using a broker, the relation between the broker, customers and providers and the guarantees that they are being truthful. While we aim at providing guarantees before the users' applications are deployed and run in the Cloud, using runtime assertion checking [32] techniques based on the formal assertions described in this paper, we may also provide monitoring to check that the user security requirements are still satisfied and it could be a way to detect if the Cloud infrastructure has been compromised.

REFERENCES

[1] J. T. Fang Liu, R. B. B. Jian Mao, M. L. B. John V. Messina, and D. M. Leaf, "SP 500-292. NIST cloud computing reference architecture," Gaithersburg, MD, United States, Tech. Rep., 2011.

[2] L. M. Vaquero, L. Rodero-Merino, and D. Morán, "Locking the sky: a survey on IaaS cloud security," *Computing*, vol. 91, no. 1, pp. 93–118, Jan 2011.

[3] D. A. B. Fernandes, L. F. B. Soares, J. V. Gomes, M. M. Freire, and P. R. M. Inácio, "Security issues in cloud environments: a survey," *International Journal of Information Security*, vol. 13, no. 2, pp. 113–170, Apr 2014.

[4] S. Pearson, "Toward accountability in the cloud," *IEEE Internet Computing*, vol. 15, no. 4, pp. 64–69, July 2011.

[5] A. Sunyaev and S. Schneider, "Cloud services certification," *Commun. ACM*, vol. 56, no. 2, pp. 33–36, Feb. 2013.

[6] M. Anisetti, C. A. Ardagna, F. Gaudenzi, and E. Damiani, "A certification framework for cloud-based services," in *ACM Symposium on Applied Computing (SAC)*, S. Ossowski, Ed. ACM, 2016, pp. 440–447.

[7] M. van Dijk, A. Juels, A. Oprea, R. L. Rivest, E. Stefanov, and N. Triandopoulos, "Hourglass Schemes: How to Prove That Cloud Files Are Encrypted," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 265–280.

[8] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 7:1–7:47, May 2014.

[9] A. Guesmi, P. Clemente, F. Loulergue, and P. Berthomé, "Cloud resources placement based on functional and non-functional requirements," in *International Conference on Security and Cryptography (SECRYPT)*. ScitePress, 2015, pp. 335–324.

[10] M. Liaqat, V. Chang, A. Gani, S. H. A. Hamid, M. Toseef, U. Shoaib, and R. L. Ali, "Federated cloud resource management: Review and discussion ," *Journal of Network and Computer Applications*, vol. 77, pp. 87 – 105, 2017.

[11] V. Amandeep and K. Sakshi, "Cloud Computing Security Issues and Challenges: A Survey." in *ACC (4)*, ser. Communications in Computer and Information Science, A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, Eds., vol. 193. Springer, 2011, pp. 445–454.

[12] W. Dawoud, I. Takouna, and C. Meinel, "Infrastructure as a service security: Challenges and solutions," in *2010 The 7th International Conference on Informatics and Systems (INFOS)*, March 2010, pp. 1–8.

[13] D. Sgandurra and E. Lupu, "Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems," *ACM Comput. Surv.*, vol. 48, no. 3, pp. 46:1–46:38, Feb. 2016.

[14] D. Jackson, *Software Abstractions*, revised ed. MIT Press, 2012.

[15] T. Emina and J. Daniel, "Kodkod: A Relational Model Finder," in *Tools and Algorithms for the Construction and Analysis of Systems: (TACAS)*, G. Orna and H. Michael, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 632–647.

[16] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14.

[17] É. Michon, J. Gossa, S. Genaud, L. Unbekandt, and V. Kherbache, "Schlouder: A broker for IaaS clouds," *Future Generation Computer Systems*, vol. 69, pp. 11–23, 4 2017.

[18] S. K. Garg, S. Versteeg, and R. Buyya, "SMICloud: A Framework for Comparing and Ranking Cloud Services," in *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, ser. UCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 210–218.

[19] M. Frtunic, F. Jovanovic, M. Gligorijvic, L. Dordevic, S. Janicijevic, P. H. Meland, K. Bernsmed, and H. Castejon, "Cloudsurfer - a cloud broker application for security concerns," in *Proceedings of the 3rd International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, INSTICC. SciTePress, 2013, pp. 199–206.

[20] E. Mostajeran, B. I. Ismail, M. F. Khalid, and H. Ong, "A survey on sla-based brokering for inter-cloud computing," in *2015 Second International Conference on Computing Technology and Information Management (ICCTIM)*, April 2015, pp. 25–31.

[21] MODAClouds Team, "MODAClouds, Underpinning the Leap to DevOps Movement on Cloud Environments," http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds-WhitePaper.pdf, 2013.

[22] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, *Advanced Web Services*. New York, NY: Springer New York, 2014, ch. TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549.

[23] The Coq Development Team, "The Coq Proof Assistant," http://coq.inria.fr.

[24] S. Souaf and F. Loulergue, "Strong security guarantees: from Alloy to Coq," in *International Conference on High Performance Computing and Simulation (HPCS)*. Orléans, France: IEEE, 2018.

[25] T. Sewell, S. Winwood, P. Gammie, T. C. Murray, J. Andronick, and G. Klein, "seL4 Enforces Integrity," in *Interactive Theorem Proving (ITP)*, ser. LNCS, vol. 6898. Springer, 2011, pp. 325–340.

[26] X. Leroy, "Formal verification of a realistic compiler," *Commun. ACM*, vol. 52, no. 7, pp. 107–115, 2009.

[27] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and Understanding Bugs in C Compilers," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2011, pp. 283–294.

[28] A. Bergmayr, U. Breitenbücher, N. Ferry, A. Rossini, A. Solberg, M. Wimmer, G. Kappel, and F. Leymann, "A systematic review of cloud modeling languages," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 22:1–22:38, Feb. 2018.

[29] S. Chong and R. V. D. Meyden, "Using Architecture to Reason About Information Security," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 2, pp. 8:1–8:30, Dec. 2015.

[30] S. Challita, F. Paraiso, and P. Merle, "Towards Formal-Based Semantic Interoperability in Multi-Clouds: The FCLOUDS Framework," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, June 2017, pp. 710–713.

[31] H. Mezni and M. Sellami, "Multi-cloud service composition using formal concept analysis," *Journal of Systems and Software*, vol. 134, pp. 138 – 152, 2017.

[32] L. A. Clarke and D. S. Rosenblum, "A historical perspective on runtime assertion checking in software development," *SIGSOFT Softw. Eng. Notes*, vol. 31, no. 3, pp. 25–37, May 2006. [Online]. Available: http://doi.acm.org/10.1145/1127878.1127900