Model-Checking Cloud Systems Using BigMC

Hamza Sahli Faiza Belala Chafia Bouanaka
LIRE Laboratory, University of Constantine II
Constantine, Algeria

sahli.hamza.glsd@gmail.com faiza.belala@univ-constantine2.dz@hotmail.com c.bouanaka@umc.edu.dz

Cloud computing is a promising concept in the IT evolution that has increasingly attracted attention from both industry and academic sectors. However, it has introduced new security problems and obstacles. Since formal methods provide a reliable mathematical basis giving rise to safely analysable and easily verifiable models, we aim in this paper to propose a formal framework to specify cloud system architectures and verify their inherent proprieties. Bigraphical Reactive Systems are adopted as a semantic framework for their graphical aspect and rigorous basis. We argue that the proposed models are useful for simulation and analysis of cloud systems proprieties as elasticity and plasticity, while using a given model checker tool dedicated to BRS.

Cloud Computing, Reconfiguration, Formal Methods, Bigraphical Reactive System, Model Checking.

1. INTRODUCTION

In recent years, cloud computing (Mell et al., 2011) has emerged as a new and promising concept in the IT evolution; it has increasingly attracted attention from both industry and academic sectors. The basic idea beyond cloud computing is to provide a poll of computing resources as on demand services (e.g. servers, storage, applications, and services). These resources are consumed by users according to their needs and by paying only their real consumption. Such flexibility and cost effectiveness is what makes cloud computing models very attractive. Albeit cloud computing offers numerous benefits, it has raised new obstacles (Michael et al., 2009) and security concerns (Vic (J.R.), 2011).The fact that the cloud is accessible from everywhere makes it vulnerable to various types of attacks, like distributed denial of service attacks (DDoS). Such attacks could heavily affect the cloud quality of service (QoS) properties as high service availability.

Cloud service availability introduces a very important concept that distinguishes cloud computing paradigm from the other ones, which is rapid elasticity (Guilherme et al., 2012) (Dustdar et al., 2011). Elasticity goes beyond a simple flexible and dynamic allocation and deallocation of resources on the fly. It implies a permanent reconfiguration of the underlying network and its associated controls. Elasticity has many forms of violation such as plasticity (Gambi et al., 2013) (i.e. the inability to spontaneously return back to the original configuration after an adaptation process).

To ensure cloud systems reliability and consistency, these concepts need to maintain a formal model that supports specification and analysis of such properties. Until now, there are only few formal models for cloud systems.

Bigraphs (Milner, 2008) enriched with a set of meta-reaction rules, giving rise to Bigraphical Reactive Systems, are a good candidate to formalise cloud computing fundamental architectural aspects and their reconfiguration. Indeed, bigraphs differ from traditional formalisms in their expressive power getting designers a great flexibility to specify their own reaction rules. In overall terms, our contribution is two-fold:

- We argue that Milner's BRS, through their graphical aspect, are capable of representing both locality and connectivity that constitute main concepts of cloud computing architecture.
- We propose a bigraph-based model for cloud system composed of two independent regions (physical or logical). For instance, the client and service provider may represent these regions. Then, interactions between these two regions are defined via reaction rules. Since BRS have an executable algebraic language, the obtained formal model serves to model-check some proprieties that are inherent to cloud systems as the elasticity and some forms of its violation (plasticity).

The rest of the paper is organised as follows. In section 2, we present related work. In section 3, we give a brief overview on Bigraphical Reactive Systems (BRS) and their dedicated model checker. Section 4 presents our bigraphical specification of cloud systems. In, section 5 we formally verify the elasticity and plasticity proprieties. Finally, some concluding remarks and ongoing work rounds up the paper.

2. RELEATED WORK

There is a significant body of work on defining and analysing cloud systems, but we are unaware of approaches involving formal and rigorous mathematical models. For instance, authors in (Grandison et al., 2010) and (Dong et al., 2010) try to tackle the lack of consensus or base comprehension on technical constituents of a cloud by presenting an initial definition of cloud computing. They particularly provide some discussion on the relationship between cloud computing and virtualization.

Formal models for managing the complexity of evolving cloud system behaviour while it is executing is a recent area of interest. Existing approaches in this context are based on various formalisms.

(Freitas et al., 2012) present an abstract formalisation of federated cloud workflows using the Z notation. In addition, a process algebra framework for the specification of virtual machines migration and the associated security policies in the cloud is given in (Jarraya et al., 2012). On the other hand, author in (Rady, 2013) proposes a formal definition of service availability in cloud computing using the web ontology language OWL. Authors in (Klai et al., 2013) propose a formal model adopting Petri nets for describing service-based business processes in cloud environments.

While our adopted formalism (BRS) is different and more appropriate, our work has a similar goal in that it is reasoning about cloud systems. We note a related bigraphical modelling approach taken on by (Benzadri et al., 2014) to model-check configurations of a cloud system. In this work, only some functional properties are verified using LTL Maude tool. Besides, all these research studies do not explicitly tackle the formal analysis of elasticity property which is inherent to cloud systems, expect those defining a systematic model-based test generation framework for testing the elastic properties of cloud systems (Gambi et al., 2013) and (Amziani et al., 2013).

3. OVERVIEW OF BRS

Bigraphical reactive systems (BRS) were initially introduced by (Milner, 2008) to provide a graphical intuitive formal model capable of representing at the same time connectivity and locality of distributed entities. Thus, it coincides strongly with cloud computing concepts. A bigraph is composed of two graphs: a place graph for entities locality and hierarchy representation and a link graph for interconnectivity representation. Bigraphs structural dynamics is formalised by reaction rules that express their eventual reconfigurations. Hence, bigraphs can be used for representing system possible configurations, and reaction rules for specifying how these configurations may evolve (i.e. relations between bigraphs).

3.1 Structural Aspects

A bigraph is the combination of two independent structures: the place and link graphs. The place graph represents system entities geographical distribution. The Link graph is a hypergraph representing interconnections between these entities. Within a BRS, system entities are represented by nodes and interactions between them are represented by edges (see Figure 1).

A node can be dotted with ports representing connection points to edges or inner/outer names. A control is also associated to each node; consisting of node type identifier that belongs to a set called signature. Each control indicates the number of ports of each node (i.e. arity), which controls are atomic for empty nodes and which of the non-atomic controls are active (i.e. subject to reactions) or passive. The inner/outer names of a bigraph indicate connectors to other elements. Such interconnection is only possible if the outer name of a bigraph or root corresponds to the inner name of another bigraph. Sites represent holes into which a root or node can be nested, they are considered as an abstraction indicating the presence of other elements.

Definition (Milner, 2009): a bigraph is formally defined by

$G = (V, E, ctrl, G^P, G^L): I \rightarrow J, I = < m, x >, J = < n, y >$, *where:*

- *V* and *E* represent finite sets of nodes and edges respectively.
- $ctrl : V \rightarrow K$ is a control map that assigns a control to each node. The signature *K* is a set of controls.
- G^P *and* G^L are Place and Link graphs respectively.
- *I and J* represent inner and outer names (interfaces) respectively, of the bigraph *G*. *m* and *n* are the number of sites and roots respectively.

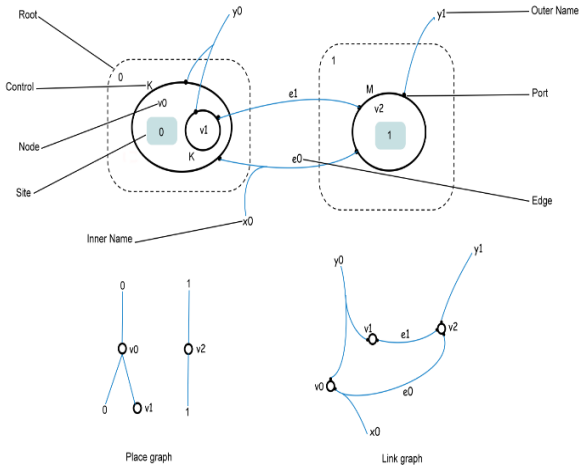

Figure 1: *The anatomy of bigraphs.*

In addition to the graphical representation, a term algebraic language is defined to specify bigraphs, language primary operations and elements are summarized in Table 1.

Table 1: *Terms language for bigraphs*

Term	Signification
$U \mid\mid V$	Juxtaposition of roots.
$U \mid V$	Juxtaposition of nodes.
$U \circ V$	Composition.
$U . V$	Nesting (U contains V).
$/x . U$	U with outer name x replaced by an edge.
x/y	Connection inner names y to outer name x.

3.2 Dynamical aspects

Bigraphs structural dynamics is expressed via reaction rules; each one defines a redex bigraph to be transformed to a reactum one.

As an example, Figure 2 represents a reaction rule that allows a person *P* in the same root, as a room *R,* to leave the room. This rule is purely a placing reconfiguration. A linking reconfiguration represents any possible connectivity; reaction rule of Figure 3 represents a person *P* connecting to a *pc* in the same root through the edge *e0*.

Formally, a reaction rule takes the form *(R,R',η)* where *R:m⟶J* is a redex, *R':m'⟶J* is a reactum and *η:m⟶m'* is a map of ordinals (Milner, 2008). The category of all bigraphs and their reaction rules constitute a BRS.

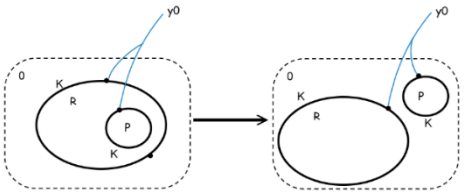

Figure 2: *Placing reaction rule.*

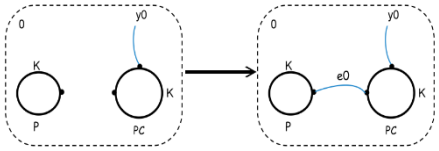

Figure 3: *Linking reaction rule.*

3.3 A Model Checker for BRS

Few tools for verifying BRS-based distributed systems inherent properties exist as BigMC (Perrone et al., 2012) model checker.

BigMC (Bigraphical Model Checker) is a model-checker designed to operate on Bigraphical Reactive Systems, where model checking is accomplished through an exhaustive search of all possible states of the bigraphical model that satisfy the property to be verified.

One of the main benefits of a model checking approach is the ability to provide a counter-example whenever the desired property does not hold in the actual system model. In our case, this means showing system configuration that violates the specified property, and the transition system path by which this configuration was reached. The full grammar of BigMC bigraph terms is summarized in Table 2 (Perrone et al., 2012).

Table 2: *BigMC terms language*

$M ::= E; M \mid E;$

$E ::= \%passive\ k : arity$

$E ::= \%active\ k : arity$

$E ::= \%rule\ n\ T \rightarrow T$

$E ::= \%property\ n\ P$

$E ::= T \rightarrow T \mid T$

$T ::= K{:}T \mid T \mid T \mid T \mid\mid T \mid \$n \mid K \mid nil$

$K ::= k[names] \mid k$

$names ::= n, names \mid n$

$n ::= [a-zA-Z][a-zA-Z0-9]* \mid -$

$P ::= matches(T) \mid terminal() \mid !P$

Using this grammar, we can specify all bigraphical elements. A BigMC model (designated by M) can be

composed using other models and/or expressions (designated by E). An expression E can be a node declaration, a reaction rule, a term (T), or a property (P). A term T can represent a single node, site, region or a combination of all these elements. Terms of the form T -> T are considered reaction rules. A property P represents a state definition to be checked with BigMC tool.

We will use this grammar to specify our cloud model. Then, we will use the BigMC tool to verify some of the cloud system proprieties.

4. CLOUD SYSTEMS BIGRAPH-BASED SPECIFICATION

At a high level of abstraction, a cloud computing system is considered as a set of computing resources (e.g. data centers, servers, services) that are distributed across multiple computing sites, and are often referred to as nodes. These resources are provided as on demand services that users (clients) can consume. Thus, two types of entities are identified in cloud computing: the front-end entity and the back-end entity that are interacting via the Internet.

The front-end represents the client interface, used to access the cloud. Clients are classified into two kinds: end users (i.e. simple cloud service consumers) and developers (i.e., costumers exploiting cloud as for Google Apps, Codeita to host their applications).

The back-end is the cloud service provider. It offers a complete system for allocating the required resources to execute user applications and managing the entire system flow.

Many types of resources can exist in a cloud as:

- Data centers: physical facilities used to gather cloud computing resources and components.
- Load balancers: devices responsible of service requests rooting and resources provision.
- Servers: infrastructures for calculation and execution.
- Virtual machines (VMs): abstractions of the underlying infrastructure.

Example 1: In order to ease the understanding of our proposed cloud system formalisation, we will first introduce the following generic example illustrating important features that will be considered.

Figure 4 depicts the architecture of a simplified cloud system (back-end) interacting with a set of end users (front-end) via the internet.

The cloud system is composed of a unique instance of the following cloud components (data center, load balancer, server and virtual machine) and offers two different services for the end users (end user 1, end user 2, end user 3).

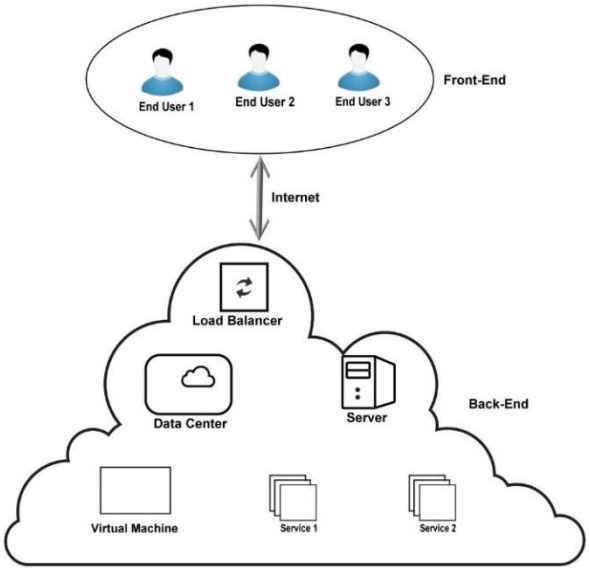

Figure 4: *Architectural elements of a cloud system.*

4.1 Modelling Cloud System Architecture

In a previous work (Sahli et al., 2014) we have shown that bigraphs constitute a suitable mathematical model allowing the formalisation of the two parts (back-end and front-end) of cloud architecture using two distinct regions of bigraph.

This is achieved thanks to a formal mapping based on correspondence rules between the cloud system elements and bigraph concepts (see Table 3).

Table 3: *Correspondence table Cloud / Bigraph Concepts*

Cloud architecture element	Bigraph element
Client, Data center, Load balancer, Server, Service, Virtual machine.	Node
Physical or logical Location of the Client and the Cloud.	Root
Various types of Links between the different elements.	Edge/Hyper Edge
Abstract elements.	Site

Let us return to our running example and apply the bigraph based formalisation approach (Sahli et al., 2014) to give a well-defined semantics of its architectural aspect.

Assuming that the two services are deployed in the same virtual machine and an end user is connected to the first service while the others are connected to the second service. The corresponding bigraphical model is shown in Figure 5.

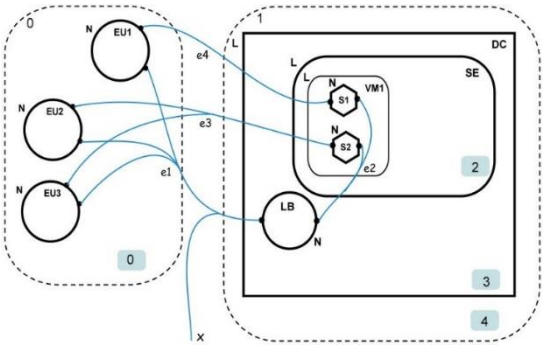

Figure 5: *The example's Bigraphical model.*

As we can notice in Figure 5, there are two services actually deployed in the virtual machine VM1 and two end users connected to S2 service through the edge e3. Thus, the S2 service cannot be allocated to another end user and the virtual machine VM1 cannot deploy any other service instance. This is expressed in our model by the absence of a site in that virtual machine.

The signature associated to a cloud bigraph is as follows: K = {L: (0, active), N :(2, atomic)}, L and N represent controls associated to different nodes. The different nodes types used in the model and their associated controls are summarized in Table 4.

Table 4: *Nodes types of cloud architecture*

Node	Control	Attribute	Arity	Meaning
EU	N	Atomic	2	End User
DC	L	Active	0	Data center
LB	N	Atomic	2	Load balancer
SE	L	Active	0	Server
VM	L	Active	0	Virtual machine
S	N	Atomic	2	Service

The cloud system initial configuration expression in BigMC tool appropriate grammar is shown in Figure 6.

```
# Nodes                          # Links
%active DataCenter : 0;          %name e1;
%active Server : 0;              %name e2;
%active VirtualMachine1 : 0;     %name e3;
%active Service1 : 2;            %name e4;
%active EndUser1 : 2;            %name e5;
%.....................;         %name x;

# Cloud System Model
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3]
| $1 | DataCenter.(LoadBalancer[e1,e2,x] | Server.(
VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3])));
```

Figure 6: *Implementing Bigraphical model in BigMC.*

We notice that each concept involved in the cloud system has a precise semantics. The conceived bigraphs do not specify just the graphical representation, but also the intended mathematical models. Furthermore, the proposed formalisation approach is general enough; it remains valid for any cloud architecture examples.

To deal with the dynamic behaviour of cloud system at runtime, we enrich the proposed bigraph-based model with reaction rules. Hence, a set of reaction rules defining system configurations and their evolution at runtime is specified.

4.2 Modelling Cloud System Reconfiguration

Albeit, bigraphs are sufficient to formally specify cloud systems static structure, they do not represent their dynamic behaviour. Our main contribution is to extend the proposed bigraph-based model for cloud system by a set of reaction rules expressing its possible reconfiguration.

Table 5 illustrates how we graft behavioural models, based on reaction rules, to graph transformation ones, to deal in this case with Bigraphical Reactive Systems (BRS).

Table 5: *Modelling cloud system dynamics*

Cloud system	**BRS**
Configuration CS.	Bigraph : $G_{cs} = (V_{cs}, E_{cs}, crtl_{cs}, G^{p}_{cs}, G^{L}_{cs})$
Reconfiguration from CS *to* CS'.	Meta reaction rule: $RL = (CS, CS', m' \rightarrow m)$
Example *RL1*: Service deployment	
$x/EU_{xe0e1}\|D_{xe0}\|\|DC.(LB_{xe0e2}\|SE.(VM.(S_{e1e2}\|d1)\|d2)\|d3)\|d4 \rightarrow$ $x/EU_{xe0e1}\|D_{xe0}\|\|DC.(LB_{xe0e2}\|SE.(VM.(S_{e1e2}\|S1_{e2e3})\|d2)\|d3)\|d4$	
Example *RL2*: Service migration	
$x/D_{xe0e1}\|\|DC.(LB_{xe0e2}\|SE.(VM.(S_{e1e2})\|d2)\|d3)\|d4 \rightarrow$ $x/D_{xe0e1}\|\|DC.(LB_{xe0e2}\|SE.(VM\|VM1.(S_{e1e2}\|d1)\|d2)\|d3)\|d4$	
Example *RL3*: Virtual machine migration	
$x/D_{xe0e1}\|\|DC.(LB_{xe0e2}\|SE.(VM.(S_{e1e2}))\|d3)\|d4 \rightarrow$ $x/D_{xe0e1}\|\|DC.(LB_{xe0e2}\|SE\|SE1.(VM.(S_{e1e2}\|d1)\|d2)\|d3)\|d4$	

Therefore, Cloud system dynamics is formalised as bigraphical reactive system. Its configuration transition is performed through a series of meta-reaction rules.

Thus, the meta-reaction rule examples cited above can be instantiated to express cloud system changes in terms of shape shifting or elasticity, while preserving cloud architectural constraints. For a better comprehension of our model, we will illustrate more these reaction rules examples in what follows.

4.2.1 Deploying a new service meta-reaction rule

It specifies a developer client type (denoted by D) being connected to the cloud in order to deploy a

new service. As we can see in the reactum of the reaction rule in Figure 7, a new service S1 is created within the virtual machine (denoted VM) along with a communication link between the developer and the service he deployed. In the redex, presence of site 1 means that VM virtual machine is able to deploy other services while its disappearance in the reactum means that the virtual machine has reached its limits (saturation), we suppose here that virtual machine capacity is of two services. We can also note the existence of another service that is already loaded in virtual machine VM and is actually exploited by a client of type end user (denoted by EU).

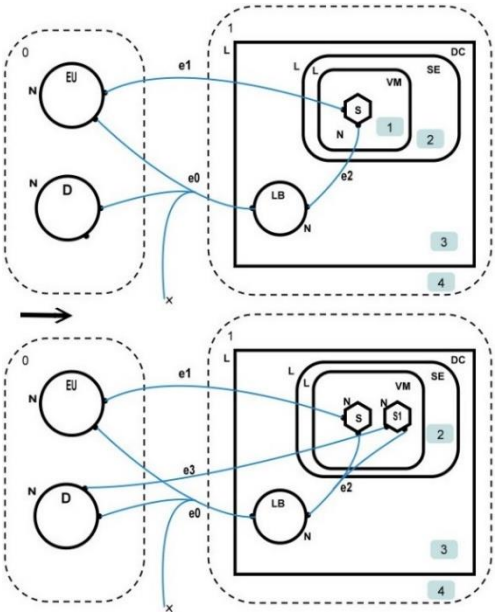

Figure 7: *Service deployment meta-reaction rule.*

4.2.2 Service migration meta-reaction rule

This rule expresses the fact that a service may migrate from one virtual machine to another for many reasons, as degraded virtual machine performance or overloaded virtual machine, expressed in our model by the absence of a site in the virtual machine. As shown in Service migration reaction rule of Figure 8, service S changes its placing from virtual machine VM to a virtual machine VM1.

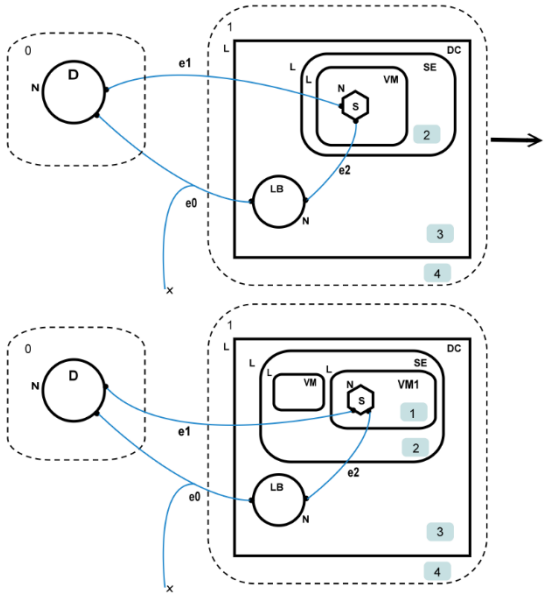

Figure 8: *Service migration meta-reaction rule.*

4.2.3 Virtual machine migration meta-reaction rule

The virtual machine migration reaction rule (see Figure 9) expresses the fact that a virtual machine may migrate from an excessively loaded host server to a less loaded server. A loaded server is expressed in our model by the absence of a site in this server. In the redex below, we can see an excessively loaded server (denoted SE).

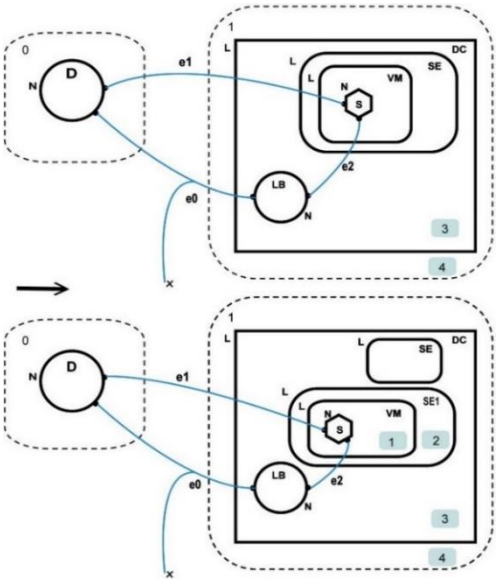

Figure 9: Virtual machine meta-reaction rule.

The presented rules are just few examples of many other reaction rules that can be expressed through our model (e.g. new client, service allocation, virtual machine replication, service instance replication, new service instance).

Example 2: Let us head back again to our previous example, a new end user wishes to connect to the second service. Assuming that a service instance can handle at most two end users simultaneously, a new instance of the service has to be created within the virtual machine to treat the new request. We also assume that a virtual machine can handle at most two service instances at the same time. Hence, the virtual machine might be replicated in order to deploy the new service instance.

To express our model dynamics, we have defined two meta-reaction rules sequences written in BigMC appropriate grammar. The first reaction rules sequence expresses the scaling up of our running example cloud architecture when the workload rises (see Figure 10).

```
# Scalling up Reaction Rules
# 1   New End User
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] | $1
| DataCenter.(LoadBalancer[e1,e2,x] | Server.(
VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3]))) ->
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
EndUser4[e1,-] | DataCenter.(LoadBalancer[e1,e2,x] |
Server.(VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3]))) ;
# 2   Virtual Machine Replication
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
EndUser4[e1,-] |DataCenter.(LoadBalancer[e1,e2,x] | Server.(
VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3]))) ->
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
EndUser4[e1,-] | DataCenter.(LoadBalancer[e1,e2,x] | Server.(
VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3]) |
VirtualMachine1_1.($2))) ;
# 3   New Service Instance
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
EndUser4[e1,-] | DataCenter.(LoadBalancer[e1,e2,x] |
Server.(VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3])
| VirtualMachine1_1))  ->
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
EndUser4[e1,-] | DataCenter.(
LoadBalancer[e1,e2,x] | Server.(VirtualMachine1.(
Service1[e2,e4] | Service2[e2,e3]) | VirtualMachine1_1.(
Service2_1[e2,-] | $2))) ;
# 4   Service Allocation
EndUser4[e1,-] || Service2_1[e2,-]  ->
EndUser4[e1,e5] || Service2 1[e2,e5] ;
```

Figure 10: *Scaling up reaction rules sequence.*

By applying this first reaction rules sequence (new end user, virtual machine replication, new service instance, service allocation), we expect the cloud system configuration shown in Figure 11. We can notice the appearance of new service S2 instance denoted S2_1, deployed in a new virtual machine VM1_1 and connected to a new end user denoted EU4.

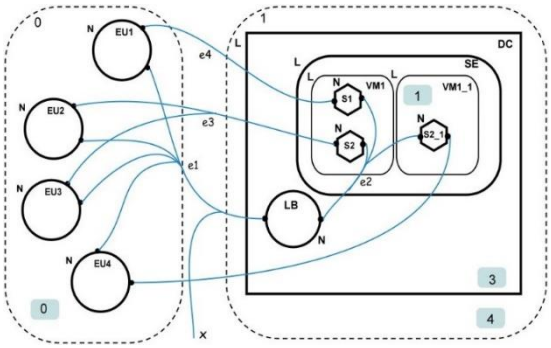

Figure 11: The resulting Cloud system configuration.

After the workload dropping (e.g. deallocation of service, disconnection of client), the cloud system has to go back to its original configuration shown in Figure 5 (scale down). Hence, the defined second reaction rules sequence provide to our model the ability to scale down (see Figure 12).

```
# Scalling down Reaction Rules
# 5  Service Deallocation
EndUser4[e1,e5] || Service2_1[e2,e5]  ->
EndUser4[e1,-] || Service2_1[e2,e5] ;
# 6  End User Disconnection
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
EndUser4[e1,-] | DataCenter.(LoadBalancer[e1,e2,x] |
Server.(VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3]) |
VirtualMachine1_1.(Service2_1[e2,e5] | $2))) ->
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
DataCenter.(LoadBalancer[e1,e2,x] | Server.(
VirtualMachine1.(Service1[e2,e4] | Service2[e2,e3] ) |
VirtualMachine1_1.(Service2_1[e2,-] | $2 )));
# 7  Service Instance Destruction
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
DataCenter.(LoadBalancer[e1,e2,x] | Server.(VirtualMachine1.(
Service1[e2,e4] | Service2[e2,e3] ) | VirtualMachine1_1.(
Service2_1[e2,-] | $2 )))->
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
DataCenter.(LoadBalancer[e1,e2,x] | Server.(VirtualMachine1.(
Service1[e2,e4] | Service2[e2,e3] ) | VirtualMachine1_1));
#8  Virtual Machine Destruction
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
DataCenter.(LoadBalancer[e1,e2,x] | Server.(VirtualMachine1.(
Service1[e2,e4] | Service2[e2,e3] ) | VirtualMachine1_1)) ->
EndUser1[e1,e4] | EndUser2[e1,e3] | EndUser3[e1,e3] |
DataCenter.(LoadBalancer[e1,e2,x] | Server.(VirtualMachine1.(
Service1[e2,e4] | Service2[e2,e3] )));
```

Figure 12: *Scaling down reaction rules sequence.*

5. FORMAL ANALYSIS OF PROPERTIES

Model checking is a fully automatic and fast verification technique, which makes it very effective one. Thus, for its ability to express and check safety and liveness properties we use BigMC tool, a model checker designed to operate on Bigraphical Reactive Systems, in order to verify some cloud systems inherent properties.

Through the following example, we identify some properties that we intend to verify using BigMC model checker as elasticity property and its dual one, i.e. a form of elasticity violation (the absence of plasticity).

While the elasticity property consists of checking that the cloud system is scaling up, when the workload rises and scaling down when, it drops. Verifying the absence of plasticity is to ensure that a cloud configuration returns to its initial state after an adaptation process.

Example 3: Cloud computing service availability is key quality that can be affected by many threats as distributed denial of service (DDoS) attacks conducting to ensure the elasticity property, which plays a significant role in keeping a high level of service availability. To illustrate elasticity and its verification technique, we return back to our running example of a cloud system and its possible reconfiguration.

Elasticity property: in (Herbst et al., 2013) elasticity is defined as ability degree of the cloud system to adapt to workload changes by provisioning/deprovisioning computing resources in an autonomic manner.

Using BigMC tool grammar (see Figure 13), we express in our BRS-based analysis technique, this property by the following formula.

```
%property elasticity size() >= $pred->size() || size() <=
$pred->size();
```

Figure 13: *Elasticity property.*

BigMC model checker tries to apply the two reaction rules sequences (see Figure 10 and Figure 12) to verify that the elasticity property holds. This means checking whether cloud configuration size is growing and shrinking.

The result of the model checking is shown in Figure 14; we can notice that BigMC model checker reaches the state *9* starting from initial state *1* without reporting any property violation (existence of counter example).This means that the cloud system is scaling up/down.

```
Welcome to BigMC!
> C:\Progra~1\BigMC/bin/bigmc -m 1000 -r 50 -p
C:\Users\hp\AppData\Local\Temp\bigmc_model6225568433644825304.bgm
1: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
$1 | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil)))
2: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
EndUser4[e1,-].nil | DataCenter.(LoadBalancer[e1,e2,x].nil |
Server.VirtualMachine1.(Service1[e2,e4].nil | Service2[e2,e3].nil)))
3: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
EndUser4[e1,-].nil | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(
VirtualMachine1.(Service1[e2,e4].nil | Service2[e2,e3].nil) |
VirtualMachine1_1.nil)))
4: (EndUser4[e1,-].nil | EndUser3[e1,e3].nil | EndUser2[e1,e3].nil |
EndUser1[e1,e4].nil | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(
VirtualMachine1_1.(Service2_1[e2,-].nil) | VirtualMachine1.(
Service2[e2,e3].nil |Service1[e2,e4].nil))))
5: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
EndUser4[e1,e5].nil | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(
VirtualMachine1_1.(Service2_1[e2,e5].nil) | VirtualMachine1.(
Service2[e2,e3].nil | Service1[e2,e4].nil))))
6: (EndUser4[e1,-].nil | EndUser3[e1,e3].nil | EndUser2[e1,e3].nil |
EndUser1[e1,e4].nil | DataCenter.(Server.(VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil) | VirtualMachine1_1.(
Service2_1[e2,e5].nil)) | LoadBalancer[e1,e2,x].nil))
7: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil) | VirtualMachine1_1.(
Service2_1[e2,-].nil))))
8: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil) | VirtualMachine1_1.nil)))
9: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
DataCenter.(LoadBalancer[e1,e2,x].nil | Server.VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil)))
[mc::step] Complete!
[mc::report] [q: 0 / g: 9] @ 10
```

Cheking end with succes

Figure 14: Elasticity checking result.

Plasticity property: this property can be seen as a cloud system stuck in a giving configuration while being incapable to return to its initial one. This time, we have associated the clause of Figure 15, to express the plasticity property, and the BigMC tool checks its absence, which means reaching the state specified in the property clause.

```
%property plasticity_absence (EndUser1[e1,e4]  | EndUser3[e1,e3]
| DataCenter.(LoadBalancer[e1,e2,x] | Server.(VirtualMachine1.(
Service1[e2,e4] | Service2[e2,e3] ))));
```

Figure 15: *Plasticity property.*

We can notice that the final state 9 corresponds to the initial state (see Figure 16), meaning that the cloud system has returned to its original configuration at some point.

```
Welcome to BigMC!
> C:\Progra~1\BigMC/bin/bigmc -m 1000 -r 50 -p
C:\Users\hp\AppData\Local\Temp\bigmc_model6225568433644825304.bgm
1: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
$1 | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil)))
2: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
EndUser4[e1,-].nil | DataCenter.(LoadBalancer[e1,e2,x].nil |
Server.VirtualMachine1.(Service1[e2,e4].nil | Service2[e2,e3].nil)))
3: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
EndUser4[e1,-].nil | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(
VirtualMachine1.(Service1[e2,e4].nil | Service2[e2,e3].nil) |
VirtualMachine1_1.nil)))
4: (EndUser4[e1,-].nil | EndUser3[e1,e3].nil | EndUser2[e1,e3].nil |
EndUser1[e1,e4].nil | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(
VirtualMachine1_1.(Service2_1[e2,-].nil) | VirtualMachine1.(
Service2[e2,e3].nil |Service1[e2,e4].nil))))
5: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
EndUser4[e1,e5].nil | DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(
VirtualMachine1_1.(Service2_1[e2,e5].nil) | VirtualMachine1.(
Service2[e2,e3].nil | Service1[e2,e4].nil))))
6: (EndUser4[e1,-].nil | EndUser3[e1,e3].nil | EndUser2[e1,e3].nil |
EndUser1[e1,e4].nil | DataCenter.(Server.(VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil) | VirtualMachine1_1.(
Service2_1[e2,e5].nil)) | LoadBalancer[e1,e2,x].nil))
7: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil) | VirtualMachine1_1.(
Service2_1[e2,-].nil))))
8: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
DataCenter.(LoadBalancer[e1,e2,x].nil | Server.(VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil) | VirtualMachine1_1.nil)))
9: (EndUser1[e1,e4].nil | EndUser2[e1,e3].nil | EndUser3[e1,e3].nil |
DataCenter.(LoadBalancer[e1,e2,x].nil | Server.VirtualMachine1.(
Service1[e2,e4].nil | Service2[e2,e3].nil)))
[mc::step] Complete!
[mc::report] [q: 0 / g: 9] @ 10
```

Figure 16: *Plasticity checking result.*

6. CONCLUSION

Elasticity is a key aspect in cloud computing that plays a significant role in keeping a high level of service availability in cloud-based systems. In this paper, we have proposed a formal framework to specify cloud-based systems and verify their proprieties as the elasticity property and some forms of its violation (e.g. plasticity). First, Bigraphical Reactive Systems were adopted as a formal framework for designing and reconfiguring cloud architectures. Cloud Bigraphs graphical and formal basis simplify considerably cloud systems readability. Then, to formally analyse the elasticity and plasticity proprieties, we have extended our proposed cloud model by defining associated clauses that are integrated in the model checker BigMC, designed to operate on Bigraphical Reactive Systems.

BigMC model checker is a very interesting tool that enables executing bigraphical models and checking some of their proprieties. As future work, we intend to evaluate our proposed bigraphical model with other complementing model-checkers.

REFERENCES

Mell, P. and Grance, T. (2011) The NIST definition of Cloud Computing. Technical Report, National Institute of Standards and Technology (NIST), Gaithersburg, MD, p. 800-145.

Michael, A., Armando, F., Rean, G. and Anthony, D. J. (2009) A Berkeley View of Cloud, A Berkeley View of Cloud: s.n.

Vic (J.R.), W. (2011) Securing the Cloud: Cloud Computer Security Techniques and Tactics. Elsevier.

Guilherme, G. and Luis Carlos, E.D. (2012) A Survey on Cloud Computing Elasticity. In Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing (UCC '12), IEEE Computer Society, Washington, USA, p. 263-270.

Dustdar, S., Yike, G., Satzger, B. and Hong-Linh, T. (2011) Principles of Elastic Processes. IEEE Internet Computing, vol. 15, no. 5, p. 66–71.

Gambi, A., Filieri, A. and Dustdar, S. (2013) Iterative Test suites refinement for elastic computing systems. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013), ACM, New York, USA, p. 635-638.

Milner, R. (2008) Bigraphs and their algebra. In Proceedings of the LIX Colloquium on Emerging Trends in Concurrency Theory (LIX 2006), Electronic Notes in Theoretical Computer Science, Volume 209, Elsevier, p. 5-19.

Grandison, T., Maximilien, E. M., Thorpe, S. and Alba, A. (2010) Towards a Formal Definition of a Computing Cloud. 6th World Congress on Services, Miami, p. 191-192.

Dong, H., Hao, Q., Zhang, T. and Zhang B. (2010) Formal discussion on relationship between virtualization and cloud computing. In Parallel and Distributed Computing, Applications and Technologies (PDCAT), International Conference on, p. 448-453.

Freitas, L. and Watson, P. (2012) Formalising workflows partitioning over federated clouds: Multi-level security and costs. In Services (SERVICES), 2012 IEEE Eighth World Congress on, p. 219-226.

Jarraya, Y., Eghtesadi, A., Debbabi, M., Zhang, Y. and Pourzandi, M. (2012) Cloud calculus: Security verification in elastic cloud computing platform. In International Symposium on Security in Collaboration Technologies and Systems (SECOTS2012), IEEE Press, p. 447-454.

Rady, M. (2013) Formal definition of service availability in cloud computing using owl. In Computer Aided Systems Theory-EUROCAST 2013, Springer, p. 189-194.

Klai, K. and Tata, S. (2013) Formal Modelling of Elastic Service-Based Business Processes. Services Computing (SCC), 2013IEEE International Conference on, p. 424-431.

Benzadri, Z., Bouanaka, C., Belala, F. (2014) Verifying Cloud Systems using A Bigraphical Maude-Based Model Checker. The 4th International Conference on Cloud Computing and Services Science, CLOSER, Barcelona, Spain.

Amziani, M., Melliti, T. and Tata, S. (2013) Formal Modeling and Evaluation of Service-Based Business Process Elasticity in the Cloud. 22nd IEEE International Conference on Collaboration Technologies and Infrastructure (WETICE 2013), Hammamet, Tunisia, p. 284–291.

Amziani, M., Melliti, T. and Tata, S. (2013) Formal Modeling and Evaluation of Stateful Service-Based Business Process Elasticity in the Cloud. On the Move to Meaningful Internet Systems OTM 2013 Conferences, Springer.

Milner, R. (2009) The Space and Motion of Communicating Agents. Cambridge University Press.

Perrone, G., Debois, S. and Hildebrandt, T. (2012) A Model Checker for Bigraphs. In proceedings of the 27th ACM Sym, in Applied Computing ACM-SAC'12, p. 1320-1325.

Sahli, H., Bouanaka, C. and Dib, A.T.E. (2014) Towards a Formal Model for Cloud Computing Elasticity. To appear in the 23rd IEEE International Conference on Collaboration Technologies and Infrastructure (WETICE 2014), Parma, Italy.

Herbst, N.R., Kounev, S. and Reussner, R. (2013) Elasticity in Cloud Computing: What It Is, and What It Is Not. Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13), San Jose, CA, USENIX, p. 23-27.