CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds

Glauco Gonçalves, Patricia Endo, Marcelo Santos, Djamel Sadok, Judith Kelner
Federal University of Pernambuco (UFPE)
Research Group in Computer Network and Telecommunication (GPRT)
Recife, Pernambuco, Brazil
{glauco,patricia,marcelo,jamel,jk}@gpert.ufpe.br

Bob Melander, Jan-Erik Mångs
Ericsson Research
Stockholm, Sweden
{bob.melander, jan-erik.mangs}@ericsson.com

Abstract— Distributed Clouds, or just D-Clouds, can be seen as a paradigm that is able to exploit the potential of sharing resources across geographic boundaries and provide latency-bound allocation of resources to third-party developers. The representation of D-Cloud resources is a challenge that involves the careful choice of characteristics that drive the mapping of requests on the substrate resources. Regarding these problems, this paper introduces the Cloud Modeling Language (CloudML), a vendor-neutral XML-based language intended to integrate the description of different cloud related aspects such as computational and network resources, services profiles, and developers’ requests in an integrated way. Furthermore, the CloudML provides a way to describe geographical location-aware services, seen particularly indispensable in D-Cloud scenarios.

Keywords— Distributed Cloud; resource description; modeling language

I. INTRODUCTION

Current Cloud Computing setups involve a huge amount of investments in datacenters, which are the underlying infrastructure of Clouds. These massive-size datacenters bring many well-known challenges such as the need for resource over-provisioning, the high cost of heat dissipation and temperature control, and power losses due to the distribution systems [10]. In contrast to this, Distributed Clouds, or just D-Clouds [1], can reduce these problems through the use of smaller datacenters sharing resources across geographic boundaries. Also, D-Clouds can reduce communication costs by provisioning servers and data close to end-users.

However, the creation of such distributed and interconnected datacenters raises resource allocation issues. Some of these have been investigated in a previous work [1]. Overall, they fall into four fundamental challenges: a) resource modeling; b) resource offering and treatment; c) resource discovery and monitoring; and d) resource selection.

Figure 1 shows how these four challenges are related to each others. First, the Cloud provider faces problems grouped together in the Conception Phase. The provider should model resources according to the type of service it will supply and the type of resources that it will offer. The next two challenges are faced in the scope of the Operational Phase. When allocation requests arrive, a Resource Allocation System (RAS) should be aware of the current status of resources in order to determine if there are sufficient available resources in the D-Cloud to attend the request. Then, if this is the case, the system may select and enforce their allocation to serve the present request.

This paper introduces the Cloud Modeling Language (CloudML), a XML-based language extension intended to cope with the aforementioned required representations. It is known that there are many description languages ([2],[3],[4],[5],[6],[7]). However, each of these was developed to cope with some specific problem domain. CloudML is proposed to model service profiles and developer’s requirements, while at the same time, to represent physical and virtual resource status in D-Clouds. CloudML is a modeling language that is able to represent different levels of abstraction,
such as computational and network resources, provider
services, and developers’ requirements.

The rest of this paper is organized as in the following:
Section II presents and details the CloudML language and its
XML Schemas; Section III is dedicated to illustrate the use of
CloudML in a simple scenario; a qualitative comparison and
discussion between CloudML and existent languages are given
in Section IV; finally, conclusions and future works are traced
in Section V.

II. THE CLOUDML

Considering the previous approaches for the representation
of resources and their limitations, it was decided to propose the
Cloud Modeling Language (CloudML) to ensure three clear
objectives: a) the language must represent all physical and
virtual resources in a D-Cloud, including their current state; b)
the proposed language must be able to model the service
supported by the provider; and c) the language must represent
developer’s requirements while also relating them to provider’s
services.

In order to grasp how the CloudML offers the integration of
such objectives, please consider Figure 2. This figure depicts a
scenario with three actors: the application developer, the Cloud
provider, and an automated Cloud management system. Further,
the figure shows the interactions between these actors
through the use of a respective description in CloudML.

First, the Cloud provider should describe all services
offered by the D-Cloud, generating a service description
document. Next, a Cloud developer may use these descriptions
to verify if its requests are attended by the concerned D-Cloud.
Note that the CloudML allows different D-Cloud providers to
comply with their respective service descriptions. In this way, a
developer may choose between different providers according to
its own criteria and convenience.

Once a Cloud provider is selected, the developer composes
a document describing its requirements and may then submit its
requests to the D-Cloud, more specifically represented by the
Cloud System Management, which will ultimately allocate
resources according to the requested resources and the current
status of the D-Cloud.

At any time, the provider may asynchronously request
status or the description of resources (the Infrastructure
Description in the figure) of the D-Cloud. Also, this same
information may be used by the Cloud Management System for
many purposes, such as load balancing, usage optimization, or
energy saving.

An important design aspect of CloudML is the abstraction
level in which the resources are modeled and offered. Given
that D-Clouds are closely related to the network virtualization
field [1], a straightforward approach is to use CloudML to
model virtualized infrastructures. Thus, in CloudML the
developers’ applications are treated as virtual networks and the
provider’s infrastructure is the substrate or physical network.

Another important design aspect is to use XML
technologies as the underlying structure to compose CloudML
documents. By adopting this well-established technology, in
contrast to new ones such as JSON [8], it is possible to use
solutions and APIs present in the XML ecosystem to guarantee
syntax correction, query of documents, and other facilities.

Next sections show the XML Schemas that describe
CloudML. They were divided into three groups: schemas for
resource description, schemas for service description, and
schemas for requirements description. For didactical purposes,
it was opted for presenting the CloudML through intuitive
diagrams generated by the Web Tools Platform (an Eclipse
plug-in [9]) instead of the tangled XML Schema code. In next
sub-sections, these three XML Schemas will be described in
more details.

A. Resource Description

This first group has two subgroups: one for short reports on
the status of resources, and the other group provides a complete
description of resources.

XML Schemas for reporting status

The basic XML element for reporting resources’ status is
the NodeStatusType (Figure 3) which represents the status
of both physical servers and virtual machines (called just nodes
in our language). This type is composed by two required
attributes (CPU and RAM) and a sequence of Storage
elements. These attributes are presented in percentage values
while the Storage has a type defining the absolute value of the
used space (Size), the Unit relative to this space (KB, MB, GB,
etc.), and an ID.
The next type is the VirNodeStatusType (Figure 4) that is used to report the status of a specific virtual machine. Such type has three attributes: the ID is a unique value used for identification purposes and defined when the VM is created; the Owner is the identification of the developer that owns such VM; and the VMState indicates the current state of the VM.

CloudML defines three self-descriptive VM states: stopped, running, suspended. The VirNodeStatusType still has a Status element whose type is the NodeStatusType described previously. The PhyNodeStatusType is similar to the one for virtual nodes, except for the omission of the VMState and Owner attributes.

XML Schemas for reporting complete descriptions

One basic element for complete descriptions is the PhysicalNodeType (Figure 5). This type has the ID attribute and four elements: NodeParams, PhyFace, VirNodeID, and VirEnvironment. The NodeParametersType (Figure 6) describes relevant characteristics including: node parameters (memory, processor, and storage), its geographical location (Location element), its functionality in the network (switch, server, etc), its current status (which is an element of the type NodeStatusType) and the OtherParams for general use and extension.

Here, there are two aspects that should be highlighted. First, the Location is the element that enables a provider to know where resources are geo-located in the infrastructure. Second, the OtherParams element can be used by providers or equipment vendors to extend CloudML including other parameters not covered by this current version. In this way, CloudML presents itself as an extensible language.

The PhysicalInterfaceType (Figure 7) is an extension of InterfaceType and is used to describe physical links associated to the interface (PhysicalLinksID element) and virtual interfaces (VirtualInterfacesID element) related to the physical node. The general InterfaceType has an ID, MAC, IPv4, and IPv6 attributes that are inherited by the PhysicalInterfaceType.

As part of the PhysicalNodeType, the VirNodeID is a simple list of the IDs of the virtual machines hosted on the node, and the VirEnvironment is a list containing information about the virtualization environment. Each item in the list informs the architecture (32 or 64 bits), the virtualization method (full or paravirtualized), and the hypervisor. Thus, an item indicates a type of virtual machine supported.

The VirtualNodeType (Figure 8) gives a complete description of a virtual machine and is similar to the physical
node. The VirtualInterfaceType also inherits from the InterfaceType, and the VirEnvironment contains only two attributes: one indicating the hypervisor and the other indicating the virtualization mode of the VM.

The PhysicalLinkType (Figure 11) describes physical links. It has an ID, a LinkParams element, and zero or more VirLinkID elements. The LinkParametersType describes all characteristics of the link such as: link technology (Ethernet, Wi-Fi, etc), capacity, the current status (current delay, current allocated rate and bit error rate), and also an extensible element (OtherParams) serving for extensions. The VirLinkID identifies the virtual links currently allocated on this physical link.

Similarly to the physical infrastructure there is a type dedicated towards the collection of virtual nodes and links called VirtualInfraStructureType (Figure 12). It has an ID, an Owner attribute (identifying the client who owns these virtual resources) and can be composed of one or more VirNode elements (of the described VirtualNodeType) and several VirLink elements of the VirtualLinkType, which is very similar to the type for physical links.

B. Service Description

CloudML provides a profile-based method for describing provider’s services, which are described by a XML Schema whose root element is described by the ServiceType (Figure 13). This type has a Version attribute and a sequence of Node and Link profiles elements. The Node element is described by the NodeProfileType and the Link uses the LinkProfileType. A Coverage element, from the CoverageType, is also described.

The NodeProfileType uses the MemoryType for RAM and Storage elements and the CPUProfileType for CPU. The first has two attributes indicating amounts of memory and the second has three attributes indicating the following aspects: CPU frequency, number of cores, and CPU architecture. The LinkProfileType has the ID for identification of the profile, the Rate for reserved rate, and the Delay for the maximum delay.

The CoverageType is intended to inform the geographical areas that the provider covers. Thus, this type is just a sequence of Location identified by three attributes:
Country, State, and City, that allows a provider to offer location-awareness services.

It is important to notice that there is a Location element in NodeParametersType (already explained) used to geo-locate nodes in the infrastructure. With these two elements (Location and Coverage) a provider is able to geo-locate its resources and offers location-aware services.

![Image 13. Type describing the service profile offered by the D-CRAS.](image)

**C. Requirements Description**

Developers describe their application requirements through a request document, whose root element is the RequestType (Figure 14). Such type is composed by three attributes: an ID, an Owner, and a Tolerance value (delay value in milliseconds), which expresses how far the virtual nodes can be placed from their required location specified on the Node element.

The NodeSpecType and LinkSpecType have an attribute to indicate their ID and an attribute to indicate the ID of the corresponding service profile chosen by the developer. The NodeSpecType has also a Location element indicating where the node must be positioned, which is defined using the LocationType. The LinkSpecType has several Node elements indicating the ID of the requested nodes that the link will be connecting.

![Image 14. Type describing the requirements that can be requested by a developer.](image)

**III. A CLOUDML USAGE EXAMPLE**

Let’s consider now an example of the CloudML usage. Through CloudML, the provider should list the different profiles using the Service description model, informing nodes and links configurations, and the developer should inform their requirements using the Requirements description document. Moreover, the Cloud Management System uses the description and status documents to describe physical and virtual resources with regard to the D-Cloud. These can be used for internal communication between equipments and components of the system and for external communication with the provider.

Next, these XML documents are described in more details. Please notice that some irrelevant parts of the XML documents were omitted for a better visualization.

**A. Services XML**

The XML document shown in Figure 15 represents services profiles defined by the Cloud provider. There are two node profiles (nodeprofile01 and nodeprofile02), two link profiles (linkprofile01 and linkprofile02), and three Coverage items with Country, State, and City specifications.

The nodeprofile01 is a node with the Linux operating system, 2 GB of RAM, 80 GB of storage and acts as a server. The nodeprofile02 profile description is similar to the nodeprofile01, with some minor differences to the parameters. The linkprofile01 profile represents a link with maximum delay and capacity equal to 0.150 ms and 1.5 Mbps, respectively. The linkprofile02 profile description is similar to that of linkprofile01.

The Coverage profile is defined as a set of Country, State, and City. In this case, the Cloud provider informs that developers can request resources located at three different localities: “Brazil, Pernambuco, Recife”, “USA, Texas, Houston”, or “Japan, Shizuoka, Hamamatsu”.

![Image 15. Services profiles XML](image)

**B. Request XML**

This example considers a developer making a simple request of two nodes and one link between them. To compose this document, the developer should read the XML documents offered by the provider, select a correspondent profile, and make its request.

The XML document shown in Figure 16 represents such simple request. The nodes are of the same profile, nodeprofile01; and the link is linkprofile01. Both previously described in the provider’s services XML. Also, node01 is located at (Brazil, Pernambuco, Recife); and node02 is at (Japan, Shizuoka, Hamamatsu).
C. Description XML

The description document represents the infrastructure of the D-Cloud, including all physical and virtual nodes. Depending on the size of the D-Cloud, this document can be very long. Thus, for a better visualization, the next examples will show only some parts of CloudML for illustration purposes: the physical infrastructure, the virtual infrastructure, and the virtual links.

The XML document at Figure 17 presents a complete description and status of all physical nodes and physical links of the D-Cloud, with the first <PhyNode> tag informing resource characteristics (like CPU, RAM...) for node 100. The node 101 description was omitted, since it is similar.

```xml
<Request ID="request01" Owner="owner01" Tolerance="0.200M"/>
<Node ID="node001" ProfileID="nodeprof012"/>
<Location Country="Brazil" State="Pernambuco" City="Recife"/>
</Node>
<Node ID="node002" ProfileID="nodeprof012"/>
<Location Country="Japan" State="Shirakawa" City="Hamamatsu"/>
</Node>
<Link ID="link001" ProfileID="linkprof012"/>
<Node ID="node001"/>
</Link>
</Request>
```

Figure 16. Request XML

The description document represents the infrastructure of the D-Cloud, including all physical and virtual nodes. Depending on the size of the D-Cloud, this document can be very long. Thus, for a better visualization, the next examples will show only some parts of CloudML for illustration purposes: the physical infrastructure, the virtual infrastructure, and the virtual links.

The XML document at Figure 17 presents a complete description and status of all physical nodes and physical links of the D-Cloud, with the first <PhyNode> tag informing resource characteristics (like CPU, RAM...) for node 100. The node 101 description was omitted, since it is similar.

```xml
<PhysInfra ID="PhysInfra01"/>
<PhysNode ID="node001" Owner="owner01"/>
<PhysNode Parameters>
  <RAI Unit="GB"/>
  <CPU Core="4" Freq="3.00GHz" Unit="GHz"/>
  <Architecture x86_64/>
</PhysNode>
<Storages>
  <Storage ID="virtstorage" Size="5.00GB" Unit="GB"/>
</Storages>
<Locations>
  <Location Country="Brazil" State="Pernambuco" City="Recife"/>
  <Status CPU="4.64GHz" RAM="6.165GB"/>
</Locations>
<Links>
  <Link ID="link001" ProfileID="linkprof012"/>
</Links>
</PhysInfra>
</PhysNode>
<PhysNode ID="node002" Owner="owner02"/>
<PhysNode Parameters>
  <RAI Unit="GB"/>
  <CPU Core="8" Freq="3.00GHz" Unit="GHz"/>
  <Architecture x86_64/>
</PhysNode>
<Storages>
  <Storage ID="node001.mount" Size="80.60GB" Unit="GB"/>
</Storages>
<Locations>
  <Location Country="Brazil" State="Pernambuco" City="Recife"/>
  <Status CPU="4.64GHz" RAM="6.165GB"/>
</Locations>
<Links>
  <Link ID="link001" ProfileID="linkprof012"/>
</Links>
</PhysInfra>
</PhysNode>
<PhysNode ID="node001" Owner="owner01"/>
<PhysNode Parameters>
  <RAI Unit="GB"/>
</PhysNode>
</PhysNode>
<PhysNode ID="node002" Owner="owner02"/>
<PhysNode Parameters>
  <RAI Unit="GB"/>
</PhysNode>
</PhysNode>
</PhysInfra>
```

Figure 17. Physical infrastructure description

The <VirNodeID> tag informs the IDs of the virtual nodes that are running at the specific physical node. In this case, according to our example, just the virtual node node01 is running at physical node 100.

There are also two physical links (<PhyLink> tags). The physical link phylink01 has virtual link virlink01 associated to it. Further information about this link was omitted here and will be described in the next Figure.

Figure 18 shows the description and the status of the all virtual nodes and virtual links of a specific owner in the D-Cloud. Particularly, this example shows how the virtual network allocated resources after receiving the request in Figure 16. Please note that this description is very similar to the physical infrastructure.

The virtual node node01 has many characteristics, such as RAM, CPU, storage, network interface, and virtual environment. In this case, as the two virtual nodes that resulted from the same type, the virtual node node02 was omitted in the document, since it is very similar to the other virtual node.

```xml
<VirInfra ID="VirInfra01" Owner="owner01"/>
<VirNode ID="node001" Owner="owner01" VirtualState="Running"/>
<VirNode Parameters>
  <RAI Unit="GB"/>
  <CPU Core="2" Freq="3.00GHz" Unit="GHz"/>
  <Architecture x86_64/>
</VirNode>
<VirStorages>
  <VirStorage ID="node001.mount" Size="80.60GB" Unit="GB"/>
</VirStorages>
<VirFunctionality>
  <Functionality Server/>
</VirFunctionality>
<Locations>
  <Location Country="Brazil" State="Pernambuco" City="Recife"/>
  <Status CPU="4.64GHz" RAM="6.165GB"/>
</Locations>
<VirInterfaces>
  <VirInterface ID="eth0-01" IPv4="192.168.0.239/24" MAC="00900B36E98"/>
  <VirEnvironment/>
</VirInterface>
<VirLinks>
  <VirLink ID="eth0-01" ProfileID="link001"/>
</VirLink>
```

Figure 18. Virtual infrastructure description

Furthermore, this example is also about the description and the status of all virtual links established in the D-Cloud. The virtual link virlink01 has information, such as technology and rate described in <LinkParams> tag. Note that virlink01 was referenced previously in physical infrastructure description as a link associated to a physical one.
As previously stated, resource and requirements descriptions are important when dealing with D-Clouds involving numerous loosely coupled components. The main objective of such descriptions is to define how the D-Cloud works and how it copes with the dynamic management of infrastructural resources according to developers’ requirements.

Authors in [2] assert that software systems need to be described differently when deployed into a Cloud (distributed or not). The reason is that scalability, availability, reliability, and other software features depend on a critical integration of the overall system composed by developer’s software and provider’s underlying resources. Therefore, developers must express their requests in terms of the parts that composed them, their relationship, and their resource requirements.

In this Section, some alternatives for resource and requirements description in D-Clouds are summarized. Additionally, CloudML characteristics will be contrasted, and a comparison will be done in order to discuss these languages, highlighting their advantages and weaknesses.

In the search for a description language, we found the OpenStack software project [11], which is focused in producing an open standard Cloud operating system. It defines a Restful HTTP service that supports JSON and XML data formats and is used to request or to exchange information about Cloud resources, beyond to send actions commands. The flavor concept is used to create types of servers, with specific hardware configurations. Furthermore, OpenStack also gives ways to describe how to scale server down or up (using pre-configured thresholds); it is extensible, allowing the addition of new features in an easy way; and it defines returns additional messages in faults case.

The main goal of the Virtual Resources and Interconnection Networks Description Language (VXDL) is to describe resources that compose a virtual infrastructure while focusing on virtual grid applications. The VXDL is able to describe the components of an infrastructure, their topology, and an execution chronogram [3]. These three aspects compose the main parts of a VXDL document. The computational resource specification part describes resource parameters. Furthermore, some peculiarities of virtual Grids are also present, such as the allocation of virtual machines in the same hardware and location dependence. The specification of the virtual infrastructure can consider specific developers’ requirements such as network topology and delay, bandwidth, and the direction of links. The execution chronogram specifies the period of resource utilization, allowing efficient scheduling, which is a clear concern for Grids rather than Cloud computing. Another interesting point of VXDL is the possibility of describing resources individually or in groups, according to application needs. Despite the fact that VXDL offers many attributes that match our goals for D-Clouds, such as the use of a virtual infrastructure description, this language lacks support for distinct services descriptions, since it is focused on grid applications only. In addition, the use of an execution chronogram is not important for Clouds.

The proposal presented in [4], called VRD hereafter, describes resources in a network virtualization scenario where infrastructure providers describe their virtual resources and services prior to offering them. It takes into consideration the integration between the properties of virtual resources and their relationships. An interesting point in the proposal is the use of functional and non-functional attributes. Functional attributes are related to characteristics, properties, and functions of components. Non-functional attributes specify criteria and constrains, such as performance, capacity, and QoS. One of the functional properties that must be highlighted is the set of component types: PhysicalNode, VirtualNode, Link, and Interface. Such properties suggest a flexibility that can be used to represent routers or servers, in the case of nodes, and wired or wireless links, in the case of communication links and interfaces. VRD proves to be a good language due its flexibility and its functional and non-functional attributes. Actually, VRD was the start-point to CloudML. One can note that the physical and virtual infrastructures descriptions are somewhat similar. Nevertheless, VRD does not cover the request description.

Another proposal known as the Manifest language was developed in [2]. Its authors proposed new meta-models to represent service requirements and constrains, and elasticity rules of a software deployment project in a Cloud environment. The building block of such framework is the OVF (Open Virtualisation Format) standard, which was extended by these authors to realize the vision of D-Clouds considering aspects as elasticity and locality constrains. These two points are very interesting to our D-Cloud scenario. With regard to elasticity, it assumes a rule-based specification formed by three fields: a monitored condition related to the state of the service (such as workload), an operator (relational and logical ones are accepted), and an associated action to follow when the condition is met. The location constraints identify sites that should be favored or avoided when selecting a location for a service. Nevertheless, the Manifest language is focused on software architecture and constrains descriptions. Hence, the language is not concerned with other infrastructural aspects, such as its status or network management.

Cloud# is a language for modeling Clouds proposed by [5] to be used as a basis for Cloud providers and clients to establish trust. The model is used by clients to understand the behavior of Cloud services. The main goal of Cloud# is to describe how Cloud services are delivered, taking into consideration the interaction among physical and virtual resources. The main syntactic construct on Cloud# is the computation unit CUnit, which can model Cloud systems, virtual machine, or operating systems. A CUnit is represents as a tuple of six components modeling characteristics and behaviors. This language can give to the end-users a better understanding of Cloud organization and how their applications are dealt with. However, Cloud# goals are different of those for CloudML, and as such do not cover all the aspects tackled by the CloudML.

The CloudML presented in this work was developed to be a vendor-neutral language for resource/request description on D-Clouds. Such neutrality is obtained both internally and externally to the D-Cloud.
First, in the internal viewpoint, the CloudML brings a common language that can be implemented by different Cloud equipment vendors in order to integrate their different solutions. Certainly, such integration cannot be made without some common protocol implemented by the vendors, but CloudML offers a common terrain for data representation that is a crucial step towards interoperability.

Also, CloudML supports the vendors’ innovation offering flexibility through the use of the OtherParams element in the description of virtual and physical nodes and links. Such optional field can be used by different vendors to transmit private information in order to tune equipments of the same vendor in the infrastructure. This characteristic is similar to OpenStack, however the last do not present network descriptions.

In the second and external viewpoint, the supported neutrality allows developers to request services from different D-Cloud providers in order to compare characteristics from each one and choose the appropriated services for their tasks. Here, it is important to notice that these providers should use some standardized interface, such as OCCI (Open Cloud Computing Interface) [12], to handle this information model.

All the languages covered here describe, in some way, computational and network resources in the Cloud. Service description is also a common place for description languages. However, these services are described in different manners. For example, the CloudML uses profiles to represent distinct types of nodes and links that compose services; the VXDL is itself a representational way to describe grid applications; the OpenStack uses flavors idea, but it is restrict to computational resources. Request description is not treated by the VRD. One interesting aspect of CloudML has to do with geo-location. With this information, the Cloud may offer interesting services with location-awareness. This point is also covered by the VXDL, VRD, and Manifest languages, but it is described in a very shy way in the respective works.

In addition to these points, the main CloudML characteristic is the description integration. With CloudML, different Cloud providers may easily describe their resources and services and make them available for developers. Thus, developers may search for the more suitable Cloud to submit their requests to.

V. CONCLUSIONS AND FUTURE WORKS

This paper presented the CloudML, a description language that expresses resources, services, and requests in an integrated way. Despite the existence of several languages, CloudML presents some characteristics that make it interesting to both Cloud providers and developers. Different providers may use CloudML to describe their resources and easily offer different types of services. With this, developers may make use of these service descriptions to verify which D-Clouds attend best their requests. As a result, developers may opt for the most convenient to use at a given time.

Currently, CloudML has been used in an experimental D-Cloud management system for handling virtual machines and virtual networks, services, and requests. This implementation was essential for adaptation of the initial theoretical model, and to validate the execution of CloudML in a real scenario.

One can note that the current CloudML specification lacks of a way for describing scaling rules. It is expected that the RequestType in CloudML would be extended to support rules describing how and when new virtual nodes and virtual links might be added to the virtual network. The amendment of scaling rules to the CloudML is currently under investigation. Other languages treated such aspect, the EML language provides high-level polices to provide scalability, where components may be replicated depending on a scaling factor. Also, the Manifest language implements an Event-Condition-Action approach to elasticity rule specifications, where actions are taken based on monitoring events from the infrastructure.

ACKNOWLEDGMENT

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil.

REFERENCES