Computing and Informatics, Vol. 31, 2012, 45–60

SERVICE ORCHESTRATION ON A HETEROGENEOUS CLOUD FEDERATION

Jorge Ejarque, Javier Álvarez, Raül Sirvent

Grid Computing and Clusters Group
Barcelona Supercomputing Center
Nexus II Building, Jordi Girona, 29, 08034 Barcelona, Spain
e-mail: {jorge.ejarque, javier.alvarez, raul.sirvent}@bsc.es

Rosa M. Badia

Grid Computing and Clusters Group
Barcelona Supercomputing Center
Nexus II Building, Jordi Girona, 29, 08034 Barcelona, Spain
&
Artificial Intelligence Research Institute
Spanish National Research Council
e-mail: rosa.m.badia@bsc.es

Henar Muñoz

Telefónica Research and Development
Distrito C, Edificio Oeste 1, Ronda de la Comunicación s/n, 28050 Madrid, Spain
e-mail: henar@tid.es

Abstract. During the last years, the cloud computing technology has emerged as a new way to obtain computing resources on demand in a very dynamic fashion and only paying for what you consume. Nowadays, there are several hosting providers which follow this approach, offering resources with different capabilities, prices and SLAs. Therefore, depending on the users' preferences and the application requirements, a resource provider can fit better with them than another one. In this paper, we present an architecture for federating clouds, aggregating resources from differ-

ent providers, deciding which resources and providers are the best for the users' interests, and coordinating the application deployment in the selected resources giving to the user the impression that a single cloud is used.

Keywords: Cloud computing, cloud federation, service orchestration, interoperability

1 INTRODUCTION

Cloud computing [1] has caused a big impact in the way a traditional data center was managed: whenever users wanted a machine to compute some of their processes, they had to negotiate in person for a good price with a data center in order to use the machines. When this contract was established, users could then deploy their services in the data center. Nowadays, the dynamism of the cloud enables the users to make this negotiation on-line, checking the price of the different infrastructure providers available, and selecting the most appropriate one for their needs. Once selected, virtualization technologies enable an easy deployment of the software that users want to run in the cloud. Many IaaS providers exist nowadays since the appearance of Amazon EC2 [2], which was one of the first providers to adopt the utility computing paradigm. There are some commercial products such as Flexiscale [3] or ElasticHosts [4], and some other cloud middleware coming from the academia side, such as Eucalyptus [5], OpenNebula [6] or EmotiveCloud [7].

One of the problem that arises from such a big set of infrastructure offers in order to deploy a service is the lack of interoperability between the different platforms: users must decide in advance to which provider they are willing to deploy their services, and thus use its API in order to do so. This has the clear limitation that only a single infrastructure provider can be used in a deployment. Although some initiatives are gaining importance in order to solve this problem (i.e. OCCI [8]), many infrastructure providers are reluctant to change their defined APIs. Moreover, it is not only a problem about having different interfaces in the different providers, since they also use different data models to represent their resources, making the interoperability even more difficult.

From what has been explained, it is clearly seen that an intermediate layer is needed in order to shield users from such a variety of cloud offers, and not only for this, but also for aggregating the resources of the different providers in a single entity. This is what is known as *cloud federation*, the ability of interacting with all these different technologies in order to deploy a service. The federation of clouds is a hot topic nowadays, and is one of the objectives of the NUBA project.

The NUBA project [9] is a Spanish funded R & D project whose main objective is to develop a framework which makes the deployment of business services in an automatic and easy way, allowing them to dynamically scale taking into account performance and BLOs. More specifically, in the NUBA architecture there is a layer

in charge of the federation of the underlying cloud infrastructures, which is able to handle the full service deployment lifecycle among different private and public cloud providers. This layer is the main work described in this paper. NUBA also provides a low-level infrastructure to build a private cloud, and an upper layer which facilitates the service creation.

The rest of the paper is structured as follows: Section 2 presents the state of the art regarding federation in cloud infrastructures. Section 3 describes the designed architecture in order to achieve federation of private and public clouds, analyzing each of the components of the architecture, whose implementation is described in Section 4. Section 5 shows a usage case which tests the implemented architecture functionalities. Finally, Section 6 draws conclusions of this paper, and shows future directions of this research.

2 RELATED WORK

Cloud computing focuses on delivering infrastructure (IaaS), platform (PaaS), and software (SaaS) as services. In the particular case of IaaS, we also distinguish between private clouds (a cloud made with the private machines of a company) and public clouds (those offering computing resources to the general public i.e. Amazon EC2). In this paper we focus on the federation of both public and private clouds.

There exist some projects already tackling the federation of clouds, RESERVOIR [10] being one of the first European projects focusing on this topic. In RESERVOIR, the federation deals with problems such as having different providers, each of them with its own API. Besides, the federation model follows an outsourcing pattern between Infrastructure Providers (IPs): when an IP runs out of computing capacity, it outsources this demand to another IP which whom it already has an agreement to do so. RESERVOIR does not have an aggregator figure for integrating different infrastructures, such as the Federated Cloud Orchestrator in our paper, and the outsourcing cannot be done to public clouds, as we do.

Another important project dealing with cloud federation is OPTIMIS [13]. The OPTIMIS goal is to create a toolkit for scalable and dependable service platforms and architectures that enable flexible and dynamic provisioning of cloud services. The innovations behind the toolkit are aimed at optimizing the whole service life cycle, including service construction, deployment, and operation, on a basis of aspects such as trust, risk, eco-efficiency and cost. Despite the similarities with our work, OPTIMIS is at its early stages and no toolkit is available by now.

In InterCloud [14], cloud federation is achieved by means of three main components: a Cloud Broker (acting on behalf of users), a Cloud Coordinator (handling the private cloud), and a Cloud Exchange (aggregating the information to match user requests with service providers). In their simulation experiments, a Cloud Coordinator decides if a private cloud has capacity to attend a request. If this is not the case, the service is outsourced to Amazon EC2. Our policy is to aggregate the information of the private and public clouds available, and to decide what is the

best distribution of the service considering this aggregated information. Thus, public clouds are included into the equation from the beginning, and not only in case of lack of resources (which is also known as cloud bursting). The fact that InterCloud is in a very preliminary stage of design is proved because nothing is mentioned in their paper about how to deal with the integration of data coming from different data models.

In addition, in terms of API interoperability between clouds, there have been several proposals for standardization, such as vCloud [11], OCCI [8] and TCloud [12]. In the NUBA architecture, both OCCI and TCloud are considered.

Our work also contributes in data conversion between different data models from different providers by using semantics. Previous work such as [15, 16] and [17] has focused on automatic semantic annotation of XML and XML schemes to RDF and OWL. In our translation, we use these results for creating ontologies from the providers' schema. Moreover, we complement this work providing a set of translation rules, which translate the concepts from one generated ontology to another.

3 ARCHITECTURE

The main goal of the federation layer is to coordinate and maintain a cloud infrastructure composed by several public and private clouds and offered to users in a uniformed way as if they were using a single cloud. Our proposal consists on a set of components which interact to each other in order to deploy the users' services in the different managed cloud providers.

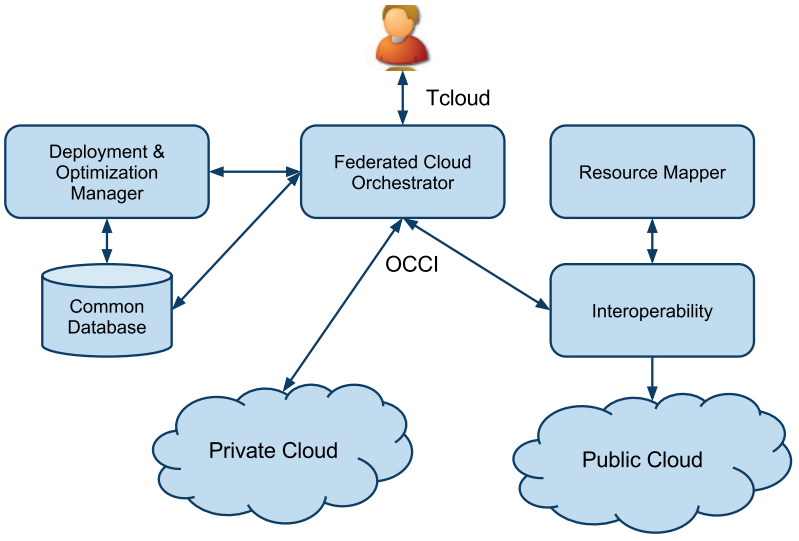

Fig. 1. Federated cloud architecture

Figure 1 depicts the overall architecture showing the components of the system. The Federated Cloud Orchestrator (FCO) is the main component of the architecture. It is the entry point of the users providing an abstraction layer between them and cloud providers. It also coordinates the actions required to deploy a service. The FCO is supported by the Deployment and Optimization (D & O) Manager, which decides what is the best provider for each service VM. All the required information to manage the cloud federation is stored in the Common Database (CDB). The D & O Manager queries this component to know which providers are available, where services have been deployed, etc.

Regarding the interaction with cloud providers, it is done by means of the Open Cloud Computing Interface (OCCI) because most of cloud middleware for setting private clouds (OpenNebula, EmotiveCloud, etc) already offer an OCCI implementation. However, public clouds offer their own interfaces for interacting with them and their own schemes for describing data. For these reasons, two additional components have been added to the architecture: the Resource Mapper (RM), which is used to translate data between the different cloud providers' models and the Interoperability component, which serves as a bridge between the FCO and the different infrastructure providers' interfaces. The following sections provide a detailed description of each one of the architecture components.

3.1 Federated Data Model

The different components of the federation layer store and exchange messages containing information about the services, resources, providers, etc. A common data model has been defined for facilitating the communication between the components and having a common understanding of each of the parts of the federation.

The Federated Data Model (FDM) is divided into three layers (Figure 2). The lowest one uses the Common Information Model (CIM) [19] for describing different types of computational resources and the management services offered by the cloud providers. The middle layer is in charge of describing the service deployment by means of the Open Virtualization Format (OVF) [20]. OVF models a service as a set of sections which describe the computational resources (*Virtual Systems*), shared disks and interconnection networks required by a service. It uses CIM concepts for describing these computational requirements facilitating the matching between the service requirements and the resource capabilities. The upper layer provides the concepts for the abstraction layer offered by the FCO which are defined by the TCloud [12] interface. With this interface, the user can define a *Virtual Data Center (VDC)* to describe a virtual infrastructure, *Virtual Appliances (VApp)* which are software system used to provide services, and *Virtual Networks* for interconnecting the different VApps. The TCloud data model is also related with the lower parts parts of the figure. It uses OVF and CIM terms for describing the VApps and Networks.

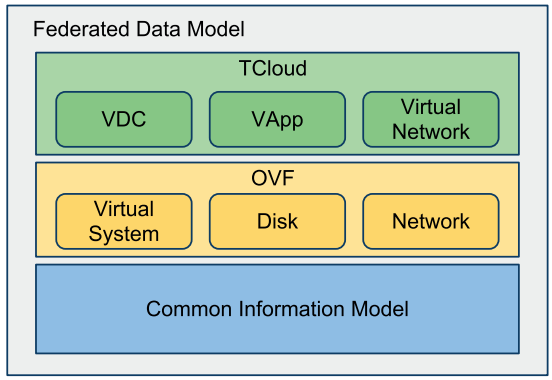

Fig. 2. Federated data model

3.2 Federated Cloud Orchestrator

The function of the Federated Cloud Orchestrator (FCO) is to provide an abstraction layer to allow service deployment management in the different cloud infrastructures. The FCO is the core of the architecture, as it is connected with the other components and coordinates them in order to achieve the orchestration of service deployments. Figure 3 depicts these relationships as well as the internal design of the component.

The FCO has TCloud on its top, which defines a REST interface to build a virtual structure and to create a service environment without the need of keeping in mind the real infrastructure underneath. Furthermore, within that virtual structure, service providers are able to specify the connectivity between the VMs they deploy, which is maintained by the FCO regardless of the cloud providers where the VMs are finally located.

Regarding service management, the FCO provides the functionalities to deploy, undeploy and redeploy a service orchestrating all the required actions to guarantee the correct deployment and undeployment of VMs in the different providers (deployment order, connection conditions, fault tolerance, etc). When a new service descriptor is received, the FCO obtains from the D & O the best Infrastructure Providers which can host each service VM. Afterwards, the FCO selects the most appropriate providers to guarantee the connexion conditions, deciding which VMs can share the same private network and thus be deployed in the same provider, or if a Virtual Private Network (VPN) between different providers is needed. Once the providers have been selected, the FCO creates each of the mentioned VMs through the Interoperability component following the order specified in the whole service description. Moreover, it has to guarantee that all the VMs are deployed correctly or none of them is. If some of the VMs of a service have already been deployed when the FCO realizes that one of the conditions above cannot be satisfied, it undeploys all of them returning to the initial state of the process and throwing an error. Be-

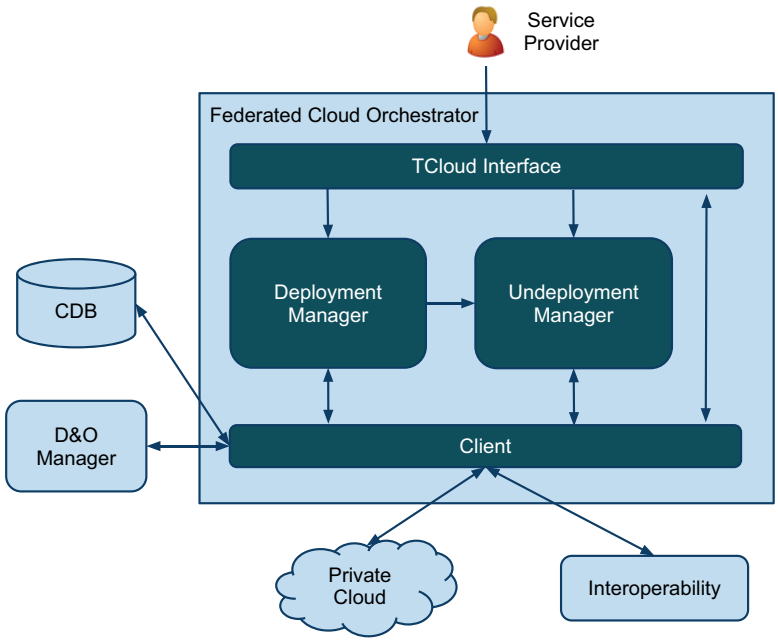

Fig. 3. Federated cloud orchestrator design

sides, the FCO uses the CDB to store all the important information related to the managed services.

In the case that an undeployment request is submitted to the FCO, it checks in which provider the VMs of the service have been deployed and then ensures their correct removal through the Interoperability component. If one or more VM undeployments cannot be completed after several tries, the FCO informs the user, which can try the undeployment again in the future. After the process has been completed, the FCO updates the information stored in the CDB.

Finally, a redeployment occurs when the FCO receives an optimization request over an already deployed service for deploying a new VM or undeploying existing VMs. When this happens, the FCO must search for the providers which can host this new VM taking into account the connectivity issues as it is performed in the deployment case. Furthermore, the FCO guarantees that these actions are accomplished or none of them is.

3.3 Common Database

The Common Database (CDB) serves as a storage system for all the components of the architecture. It provides tools to manage the storage of service descriptions as well as other information related to the infrastructures available in the federation.

The FCO uses the CDB to store TCloud entities (e.g. VDCs) and other data required to manage services. On the other hand, the RM uses the CDB to store its translation rules and data models. Finally, the D & O uses the CDB information about the available infrastructure providers and their resources. Thus, it is important that the CDB provides an efficient XML management system, because all the information that the other components need to store is represented either in XML or plain text.

3.4 Deployment and Optimization Manager

All the business logic in the federation layer is carried out by the Deployment and Optimization (D & O) Manager. The D & O Manager is in charge of taking placement decisions, for selecting the most suitable Cloud providers according to Service Providers (SP) and Infrastructure Providers (IP) policies and technical constraints. These decisions are taken when a new VM has to be deployed in the federated cloud (a deployment requested by the FCO) or proactively when it detects a situation where the current deployment of a VM can be improved.

Figure 4 shows the high level design of the component. The central module is a rule engine, which is in charge of evaluating a set of rules (from the knowledge base) over a facts base. The rule evaluation infers what is the most suitable deployment and invokes optimization actions when optimization facts are found. The Fact's base is periodically synchronized with the CDB by means of a sensor which queries it to obtain new available providers and the resources provided by them. The knowledge base (composed by a set of rules) is provided by the entities involved in the federation to model the enterprise, user o service policies. These rules have been classified in three types according to their role in the federation. On the one hand, the SP rules model the SP preferences on the resource selection such as the location, energy efficiency or preferred and forbidden providers (black list). On the other hand, there are IP rules, which model the IP preferences for selecting their customers. Finally, the federation rules model the common federation policies, such as rules for discarding resources and providers which do not fulfill the VM requirements.

3.5 Resource Mapper

The Resource Mapper (RM) is in charge of performing a semantic translation between the models used by the different cloud providers and the FDM. The semantic translation methodology used by the RM consists on applying a set of mapping rules over semantic annotated descriptions.

Figure 5 shows an overview of this methodology. From the different cloud providers' schemes, the RM creates an ontology where each concept used by the provider is mapped to an ontology class. In Addition, some semantic mappings are included in the RM in order to model the equivalences between the different cloud providers concepts and the FDM. Once the ontology and the semantic mappings have been defined, the RM translates descriptions from these providers to the FDM and vice versa. When a description translation is requested, the RM automatically

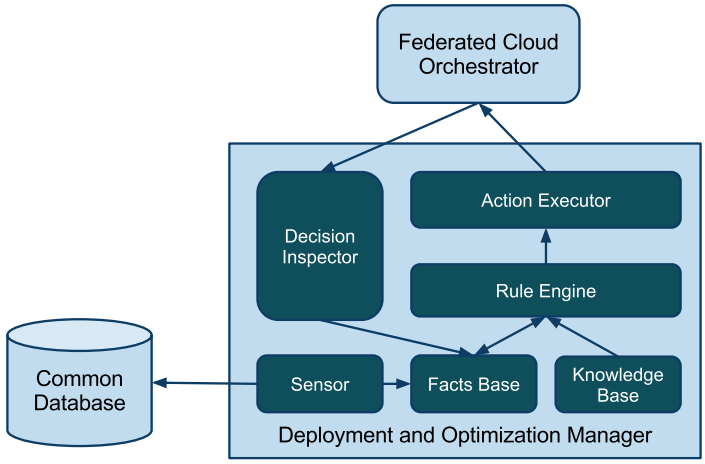

Fig. 4. Deployment and optimization manager design

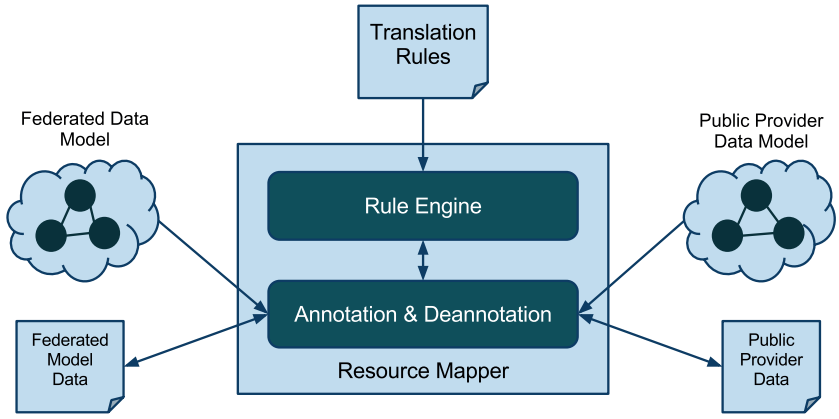

Fig. 5. Resource mapper design

annotates it according to the generated provider's ontology. The annotated description together with the mapping rules are loaded into a rule engine, which evaluates the mapping rules over the annotated descriptions. As a result of this evaluation, the rule engine infers an equivalent description following the federated model. The same process is used for the reverse case (from FDM to provider's schema). The description in the federated format is automatically annotated according to the FDM concepts and the rule engine applies the reverse mapping rules. In both cases, once the translated description has been obtained, it is serialized in the corresponding format (XML in the case of the FDM and XML or JSON object for the providers schemes).

3.6 Interoperability

Public providers offer resources to their users by means of Web Service interfaces. Unfortunately, there is not a clear standard still for these interfaces and currently each provider implements their own interface. For this reason, there is a need to unify them in a common interface which is used by the FCO to deploy the service VMs. The Interoperability component tries to cover this need. It transforms the OCCI methods for managing VM instances, storage and network resources into the equivalent calls to the providers' methods. To achieve this functionality, the Interoperability component is supported by the RM for translating the input and output data between the FDM and the providers' schemes and by a set of plug-ins which implement the equivalent execution of the OCCI methods using the providers' interfaces.

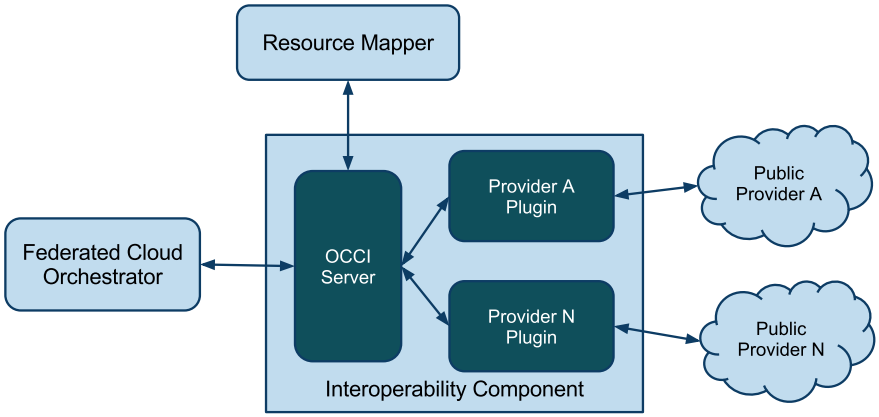

Fig. 6. Interoperability component design

Figure 6 gives a detailed view of the Interoperability component. When the FCO invokes an OCCI method of the Interoperability component, the OCCI server extracts the input data, the invoked method and the requested provider. Then, the input data is sent to the RM, which translates it into the provider's schema. Once the input data has been translated, it is used to execute the corresponding provider's plug-in, which executes the required method using the provider's interface. Finally the output data is translated into the FDM using the RM again.

4 IMPLEMENTATION

This section describes the implementation of the architecture, which has been done in Java. Every component has been developed as a RESTful Web Service to keep uniformity in all the interactions. Their interfaces have been implemented using

Jersey [21], which has been also used to develop clients for the different components to facilitate the interaction between them.

The FCO has been implemented following the TCloud interface specification. TCloud was chosen over OCCI or a custom made interface because the former was considered to be too low-level oriented and the latter would have moved the component further away from standards. The FCO has to deal with two kinds of XML data: TCloud entities and service descriptors (OVF). To make easier the management of the former, the TCloud XML schema has been bound to a set of Java classes using JAXB [22].On the other hand, to deal with OVF files, the FCO makes use of an OVF manager API [27], which is based in JAXB and provides methods to split service descriptors into groups of VMs within the same private network or to easily extract service information.

An open source XML repository called eXist [23] acts as CDB, as it covers all the requirements needed. Thus, eXist can be installed as a standalone server or deployed in any servlet container as an efficient XML management method which provides a comfortable REST interface. eXist also supports the definition of queries in XQuery language [24], which can be invoked through a REST interface to easily obtain the stored information. Some of these queries have been written to allow the rest of the components to retrieve datasets or specific information within the CDB. For instance, the D & O can obtain the providers' descriptors satisfying certain conditions, and ignoring the rest of them.

The D & O Manager is mainly composed by the rule engine. For the federated cloud, the JBOSS expert rule engine [28] has been used. This rule engine uses DROOLs as the rule language. In this sense, all the rules which arrive from the service manifest description or directly from the providers are translated into the DROOL language by the D & O Manager. The facts base is composed by a set of Java objects which are instances from the FDM, whose values are taken from the Common Database thanks to the Sensor module. Finally, there is the Descriptor Inspector, which is the D & O manager REST API (extending the TCloud specification), and the Action Executor, as set of plug-in actions to execute the rule actions when the condition threshold have been satisfied.

The Resource Mapper has been developed with a custom made REST interface that provides methods to manage translation rules and data models, storing them in the CDB, and to translate data applying these rules. The translation part has been done using Jena [25], which provides a engine to evaluate rules over RDF graphs. The RM first converts the XML to RDF using an XSLT document based on [18]. Then, it applies the set of rules for the requested provider, and finally turns back the resulting RDF into XML using another XSLT file, which has been written from scratch. Those XSLT transformations are managed with JAXP [26].

The Interoperability front-end has been implemented following the OCCI v1.0 specification with OVF as VM descriptor, because it maintains the uniformity within the system. In its back end, the Interoperability is composed by a set of plug-ins that are able to interact with the different public cloud infrastructure providers. This

plug-ins use the RM to translate the OVF descriptor to the input data expected by the cloud provider.

5 EXPERIMENTATION

We have performed a set of tests to validate architecture implementation and to check the correct behavior of components. To make these tests, one private cloud and two external cloud providers (Amazon and Flexiscale) have been registered in the CDB with the capabilities described in Table 1. For interoperating with the external providers, the Resource Mapper and Interoperability component have been configured, defining the translation rules and implementing the corresponding plug-ins for each external provider. The translation rules have been written to map data from the FDM to the Amazon and Flexiscale schemas and vice versa. In addition to these rules, two plug-ins have been implemented to provide basic OCCI functionalities such as the *POST* method in *Compute* using the *RunInstances* and *CreateServer* methods offered by the Amazon and Flexiscale APIs. An example of a rule to convert an Amazon's small instance to the FDM is shown below.

```
[Rule1:
      (?ec2 rdf:type ec2:AWS_Small)
      ->
      (?cs rdf:type cim:ComputerSystem)
      (?cs fdm:hasMemory ?mem)
      (?cs fdm:hasDisk ?disk)
      (?cs fdm:hasProcessor ?cpu)
      (?mem rdf:type cim:CIM_Memory)
      (?mem cim:BlockSize '1024'^^xsd:float)
      (?mem cim:ConsumableBlocks '1782579'^^xsd:float)
      (?d_mem rdf:type cim:CIM_Memory)
      (?d_mem cim:BlockSize '1024'^^xsd:float)
      (?d_mem cim:ConsumableBlocks '167772160'^^xsd:float)
      (?disk rdf:type cim:CIM_DiskDrive)
      (?disk cim:hasMediaPresent ?d_mem)
      (?cpu rdf:type cim:CIM_Processor)
      (?cpu cim:MaxClockSpeed '1000'^^xsd:float)]
```

Once the infrastructure has been set up, a service deployment (Table 2) is requested using the FCO. It divides the overall Service's OVF in two OVFs which include a single VM description each one. Then, the FCO contacts the D & O Manager to get the best possible allocation for each VMs. For validating this part, the D & O has been configured with a rule for discarding the providers whose resources do not fulfill the technical requirements and a query which selects the resources ranked by price (lowest price is the best). The following lines show how the rule and query have been implemented.

```
EnvelopeType(env) ^ VirtualSystemType (env,vs)^ VirtualHardwareSectionType(vs,vh)^
CIMComputerSystemType(comp) ^ HasProcessorPool(comp,cpu) ^HasMemoryPool(comp,ram) ^
HasDiskPool(comp,disk) ^ !item(vh,10)>disk ^ item(vh,3)>!cpu ^ !item(vh,4)>ram
:-
com.setHasScore (0)

SELECT com WHERE CIMComputerSystemType(com)^hasScore(com,score)^score > 0
ORDER BY Price(com)
```

Table 3 shows the selected resources for the requested service deployment according to the implemented rules. These lists are returned to the FCO which gets the first resource of the VM lists and performs the VM creation. For VM1, it invokes directly the OCCI interface provided by the private cloud middleware and, for VM2, the FCO invokes the Interoperability component which will use the Resource Mapper to translate the OVF of the VM2 to the Flexiscale's schema. In case that a deployment fails, the FCO will try to do the same with the next resource of the list.

Provider	Instances	Capabilities (Mem./CPU/Disk)	Price
Amazon	small	0.5 GB/1 CPU/160 GB	0.085 $/h
	large	7.5 GB/2 CPU/850 GB	0.34 $/h
	xlarge	7.5 GB/4 CPU/1.6 TB	0.68 $/h
Flexiscale	server 17	0.5 GB/1 CPU/no limit	0.035 $/h + disk
	server 18	1 GB/1 CPU/no limit	0.053 $/h + disk
	server 21	4 GB/2 CPU/no limit	0.178 $/h + disk
Private Cloud	max	2 GB/2 CPU/2 TB	0 $/h

Table 1. Providers' description

VM	Capabilities
VM1	1 GB/1 CPU/100 GB
VM2	3 GB/2 CPU/1 TB

Table 2. Service description

VM	Selected Resources
VM1	1. Private Cloud (0 $) – 2. Server 18 (0.061 $/h) 3. Server 21 (0.20 $/h) – 4. Large (0.34 $/h) 5. XLarge (0.68 $/h)
VM2	1. Server 21 (0.20 $/h) – 2. XLarge (0.68 $/h)

Table 3. Resource selection for service VMs

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the architecture of a cloud federation layer designed in the context of the NUBA project. This layer is composed of different components in charge of the set of functionalities to achieve the federation of private and public clouds. The FCO shields the SP about having to deploy its services in different cloud providers using different interfaces for each case. The CDB stores all the information needed to aggregate the information of the cloud providers. The D & O Manager is able to decide which is the best provider to deploy a service, taking into consideration

rules both from the SP and the IP. Finally, the RM and Interoperability components provide the functionalities to interact with public clouds translating the different data models used by IPs to the FDM, which is the data model used by all our components in order to represent the information in an homogeneous way. All these components have been implemented as RESTful Web Services, using Java, and taking into consideration current standard proposals, such as OCCI, TCloud or OVF.

We have shown we go a step beyond current work in cloud federation, aggregating available resources in order to consider both public and private clouds in the first step of the deployment (not only to achieve cloud bursting), and also considering the conversion of data between the different data models of the cloud providers. Besides, we have an available implementation of the described architecture.

Our future work includes enhancing the functionalities supported by the federation layer (more specifically by the FCO) in order to cover the full lifecycle of a service, and the development of new plug-ins for other public cloud providers in the Interoperability module.

Acknowledgements

This work is supported by the Ministry of Science and Technology of Spain and the European Union under contract TIN2007-60625 (FEDER funds), the Ministry of Industry of Spain under contract TSI-020301.1009.3 (Avanza NUBA project) and Generalitat de Catalunya under contract 2009-SGR-980.

REFERENCES

[1] Vaquero, L.—Rodero-Merino, L.—Caceres, J—Linder, M.: A Break in the Clouds: Towards a Cloud Definition. In ACM SIGCOMM Computer Communications Review, Vol. 39, 2009, No. 1, pp. 50–55.

[2] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[3] Flexiscale Public Cloud. http://www.exiant.com/products/exiscale/.

[4] Elastic Hosts. http://www.elastichosts.com/.

[5] Eucalyptus Community. http://open.eucalyptus.com/.

[6] Open Nebula. http://opennebula.org/.

[7] EMOTIVE Cloud. http://emotivecloud.net/.

[8] Open Cloud Computing Interface Working Group. http://www.occiwg.org/.

[9] NUBA project. http://nuba.morfeo-project.org/.

[10] Rochwerger, B. et al.: The Reservoir Model and Architecture for Open Federated Cloud Computing. IBM Journal Res. Dev., Vol. 53, 2009, No. 4, pp. 535–545.

[11] vCloud API Specification v1.0, http://communities.vmware.com/docs/DOC-12464.

[12] TCloud API Specification v0.9., http://www.tid.es/files/doc/apis/TCloud_API_Spec_v0.9.pdf.

[13] Ferrer, A. J. et al.: OPTIMIS: A Holistic Approach to Cloud Service Provisioning. In 1st International Conference on Utility and Cloud Computing 2010.

[14] Buyya, R.—Ranjan, R.—Calheiros, R. N.: InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. Algorithms and Architectures for Parallel Processing, LNCS Vol. 6081, 2010, pp. 13–31.

[15] Battle, S.: Round-Tripping Between XML and RDF. In 3rd International Semantic Web Conference 2004.

[16] Ferdinand, M.—Zirpins, C.—Trastour, D.: Lifting XML Schema to OWL. In 4th International Conference on Web Engineering 2004.

[17] Bohring, H.—Auer, S.: Mapping XML to OWL Ontologies. In 13th Leipziger Informatik-Tage 2005.

[18] AstroGrid-D XML2RDF. http://www.gac-grid.org/project-products/Software/XML2RDF.html.

[19] Distributed Management Task Force Inc., Common Information Model Infrastructure Specification v2.6, DMTF Standard DSP0004, 2010.

[20] Distributed Management Task Force Inc. Open Virtualization Format Specification, v1.1, DMTF Standard DSP0243, 2010.

[21] Jersey: JAX-RS implementation. http://jersey.java.net/.

[22] Java API for XML Binding. http://jaxb.java.net/.

[23] eXist Open Source Native XML Database. http://exist.sourceforge.net/.

[24] W3C Consortium. XQuery 1.0: An XML Query Language 2010.

[25] Jena 2 Semantic Web Framework. http://jena.sourceforge.net/.

[26] Java API for XML Processing. http://jaxp.java.net/.

[27] Claudia's OVF Manager. http://claudia.morfeo-project.org/wiki/index.php/OVF_Manager.

[28] Drools. http://www.jboss.org/drools/.

Jorge Ejarque received his engineering degree in telecommunications (2005) and his M. Sc. degree in computer architecture network and system (2009) from the Technical University of Catalunya. In 2004, he was with Technical University of Delft where he did his Final Thesis. In 2005, he worked as IT consultant in Better Consulting, and at the end of 2005, he joined the Grid Computing Group in the Barcelona Supercomputing Center (BSC). During his career at the BSC, he has contributed in the design and development of different tools for grids and clouds and has been involved in several national and international projects (Red Temática para la Coordinación de Actividades Middleware en Grid, e-Ciencia, Latin American GRID, CoreGRID and BREIN). Moreover, he was a member of the experts' board of the National Grid Initiative and member of the scientific committees of the Ibergrid (2009/10), ADVCOMP (2009/11) and Cloud Computing (2010/11)

conferences. He is currently working towards his Ph. D. on semantic interoperability of distributed computing platforms and is also involved in the Spanish NUBA project and in the FP7 project OPTIMIS, which are focused on the construction of platforms for efficient service deployment in different cloud providers.