A Tool for Mapping and Editing of Cloud Patterns: the Semantic Cloud Patterns Editor

Beniamino DI MARTINO*, Antonio ESPOSITO

Department of Engineering, Universitá degli Studi della Campania Luigi Vanvitelli, Aversa, 81031 Italy

e-mail: beniamino.dimartino@unina.it (**Corresponding author*), antonio.esposito@unicampania.it

Abstract: Owing to the huge number of Cloud services which are currently available on the market and to the lack of a commonly accepted standard for a machine-readable description of their interfaces, automatic discovery and composition tools and techniques for interoperable Cloud services are still in an early stage of development. Moreover, the current tools do not provide a very user-friendly interface to interact with. In this paper a service for the automatic discovery and composition of Cloud services, guided by Cloud Patterns, is presented. By means of a user-friendly interface, the user can both define a new services' composition, with the creation of a new pattern, or modify an existing one. The proposed service exploits a semantic based representation of Cloud services and patterns, complemented by a description of the input and output parameters for the several described services and of the patterns' workflow.

Keywords: Cloud Computing, Cloud Patterns, Services Orchestration, Services Discovery, Services Composition.

1. Introduction

The use of Patterns in Software Development is well documented, as many Patterns repositories currently exist and are being developed. Great interest has been shown in the definition of new, updated Patterns for modern programming and developing paradigms. Cloud Computing, for instance, has strongly benefited from the introduction of Pattern based solutions, as they can provide useful means to reduce many challenges connected to Cloud development, such as portability and interoperability issues [3,10,11]. The solutions provided by such Patterns are, in many cases, linked to each other: it is possible, for instance, to design one or more components from a Pattern by exploiting the information provided in another one. **Pattern Languages** are based on the idea to provide useful connections among Patterns, in order to understand how they are related to each other and how they can be used together to provide refined solutions and more powerful functionalities. However, as formalisms to describe Patterns are still under development and struggle to provide a comprehensive representation of all possible Pattern categories, also the existing relations among them are still not fully described by standards.

In the past years the IT market has been revolutionized by the advent of Cloud Computing, which has appealed to both small and big enterprises, and to public Governments [8] thanks to the benefits it brings: the "pay as you go" paradigm, according to which customers can rent hardware and software resources instead of buying them, thus reducing upfront-investments; the world-wide distribution of the hardware resources, which makes Cloud infrastructures more robust to natural disasters and thus more reliable; the possibility to make Cloud applications automatically scale, which implies a better use of existing resources and less management costs. Besides these and many more advantages we haven't mentioned here, Cloud Computing also offers the opportunity to compose services from different providers, in order to obtain complex applications, which exploit the best characteristics of different platforms. However, there are often practical issues which limit the interoperability of the existing Cloud platforms: different data formats, parameters' semantics, unclear descriptions of the exposed APIs and so on. Furthermore, many vendors try to bind their customers to their own platform, making it difficult or expensive for them to port their applications to another environment when needed ("vendor lock-in"). The extreme variety of services and resources currently available represents a good opportunity for customers, who can leverage offers from several providers and choose the best ones; conversely, this can also represent a source of confusion for users, whose knowledge of the Cloud Computing panorama is limited to a restricted set of services and platforms, thus making Cloud Services integration an open challenge [15]. In order to support interoperability and to promote a better interaction between services exposed from different providers, we have exploited semantic-web technologies and Cloud patterns to help customers in building their Cloud applications. By means of a semantic-based formalism, it is possible to discover Cloud Services and compose them, being guided by Patterns in the whole process. In particular,

https://doi.org/10.24846/v27i1y201812

this paper presents **Semantic Cloud Patterns Editor (SCoPE)**, a service supporting the user in selecting and managing services from several providers to build Cloud patterns. Such a service (implemented as a tool) is based on semantic descriptions of cloud providers' resources and services, modelled with ontologies. The tool uses a graph model to present the available services and the palette of already-defined patterns, and provides a GUI that allows to build, modify and manage such a model.

The remainder of this paper is organized as follows: Section 2 reports the current state of the art regarding the standards for Pattern definition and their capability to represent their relations and connections among Patterns; Section 3 describes our graph-based approach; Section 4 describes the application of the approach to the discovery and composition of Cloud Services, through examples; Section 5 provides a description of the Graphical Interface of the **SCoPE** tool, by examples; Section 6 concludes the paper with some considerations on current results and future work.

2. State of the Art

2.1 Description of Workflows

The orchestration and composition of cloud services has been the topic of several initiatives and research efforts. Some of them receive the support from industry and are adopted by important companies in the field. The mOSAic Fp7 project [9] explicitly addressed the issues related to Cloud Services discovery and composition, by exploiting ontologies and semantic-web technologies to describe and annotate Cloud resources and then compose them through adapters and connectors, whilst mOSAic featured patterns, didn't focus on Cloud patterns and missed a tool for their creation, composition and automatic deployment. A similarly semantic-based approach to services composition is proposed in [16], where Cloud services' interfaces are described in terms of their inputs and outputs and a similarity function is applied to identify corresponding parameters and determine possible concatenations of services. The work presented in [17] proposes an artificial intelligence based technique in which a search tree is created for each Cloud provider and then scanned to obtain a composition of services according to the customers' requirements. Both [16] and [17] assume that a complete knowledge of the target Cloud computing environment is available, while neglecting information regarding fees and SLAs management. The work presented in [13] applies an agent-based approach to retrieve also partial information on the services to be composed and actually perform their composition in response to a customer's request. This approach is similar to the one proposed in [12], where intelligent agents are used to implement tools which enable users to discover, compose and monitor cloud services and resources.

All the previous approaches, while being scientifically relevant and providing useful results, can be quite difficult to practically exploit, due to the lack of user friendly GUIs to support customers in the specification of their requirements. With **SCoPE**, we want to exploit the capabilities of semantic technologies and Cloud Patterns to provide a user-friendly interface for the discovery and composition of Cloud services, which can suggest the most suitable application architectures even when the target is not completely known.

Non-academicals research efforts have also produced user oriented solutions, with immediate applications to real-world situations. For an instance, orchestration and composition (but not discovery) of Cloud Services are also the focus of the OpenStack **Heat** project [6], which has developed an interesting template-based formalism going under the name of **HOT**. Such a formalism allows users to define services' properties and how such services should interact in order to provide useful functionalities. The access to the Heat service and management of HOT templates are available via the **Horizon** dashboard, which allows users to graphically interact with the entire OpenStack platform. While being compliant with the AWS **CloudFormation** [5] template format, HOT is still limited as regards the supported services and general expressivity.

Topology and Orchestration Specification for Cloud Applications (TOSCA) [4], is an OASIS standard language used to describe both a topology of Cloud based web services, consisting in their components, relationships, and the processes that manage them, and the orchestration of such services, that is their complex behaviour in relation to other described services. The combination of topology and orchestration, in what the standard defines as **Service Template**, accurately describes all the essential elements needed by each service

to provide its functionalities, in order to ease deployments in different environments and to enable interoperability. Also, management of services throughout their complete life-cycle (deploying, scaling, updating, monitoring...), when applications using them are ported to different Cloud platforms, is also supported.

In the current Cloud market, several vendors are offering different and competitive services which try to meet the requirements of their potential customers. However, because of such a variety, it can become difficult for users to decide how to build and manage their applications in different and specific contexts. Developers could benefit from a guidance in identifying the best suited architectural solutions for their applications: in this situation, Cloud Patterns can represent the perfect deal, since they have been created to provide solutions to design and based on previous experiences encountered by other programmers and developers.

Recently a number of initiatives related to the description of Cloud patterns emerged, both from the academic field such as [14,2] and from vendors of cloud services such as Amazon [1] and Microsoft [7].

Patterns not related to a specific Cloud vendor (referred to as **Agnostic Patterns**) are extremely generic and can be easily applied to different contexts. However, they require more effort in the implementation phase. Conversely, vendor specific Patterns offer better details and can be immediately implemented on the reference platform, but are less flexible and adaptable to other situations. Agnostic versions of vendor specific patterns (or **Proprietary Patterns**) can be derived and used to define new architectural solutions for different Cloud platforms.

Proprietary patterns provide directions on the particular services or components which can be used to implement the functionalities portrayed, obviously referring to the platform they have been designed for: this does not mean that they can't be used in a more general context, but they result to be less flexible than Agnostic patterns. On the other hand, Agnostic patterns never refer to a particular implementation but describe general concepts: so, they are flexible and can be applied to different platforms, but their implementation is not as immediate as for Proprietary patterns.

Since there is no machine-readable definition of the Patterns provided in such catalogues, users have to rely on the textual description provided by publishers to understand how they work and are interrelated. As for Design Patterns, such descriptions lack the necessary level of detail to fully understand the correspondences existing among Patterns' participants.

3. The graph-based Approach

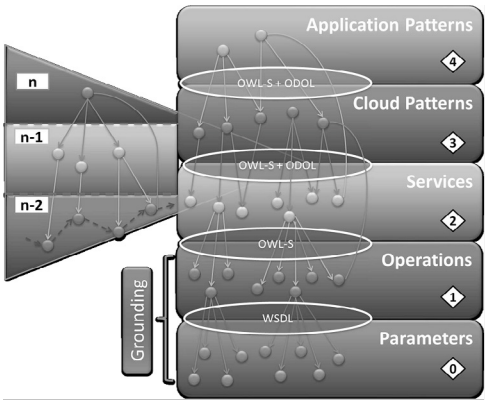

Figure 1. The Conceptual Layers

The graph-based mapping among Patterns which we are going to introduce is strongly related to a semantic representation of such Patterns. Here we briefly introduce such a representation, while a more precise description of the notation can be found in [10].

The overall model is a graph-based representation, structured into five conceptual layers. The graph represents concepts (graph nodes) and relationship (graph edges) at different levels. In each level are represented relationships among concepts of the same level in addition to inter-level relationships. The five conceptual levels are reported in Figure 1:

- The **Parameters Level** represents the description of the data exchanged among services as input and output of the operations they expose.

- The **Operations Level** represents the syntactic description of the operation and functionalities exposed by the Cloud Services; it provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.

Table 1. Results from query in Listing 1

Equivalent Parameter	Vendor	Service	Operation
InstanceID	Google	ComputeEngine	DeleteInstance
InstanceID	Google	ComputeEngine	DeleteInstance
ImageID	Amazon	EC2	StartVM
InstanceID	OpenStack	Nova	NovaBoot
VMID	Azure	Virtual Machine	StartRole

- The **Services Level** represents the semantic annotation of the vendor-dependent Cloud Services (exposed through OWL-S) and the supporting ontologies needed to identify the cloud provider supported operation, input and output parameters. This level presents details of the cloud provider platform architecture, the functionalities exposed and the underlining details. This level contains also the semantic description of the agnostic Cloud Services exposed through an ontology that reports, in vendor neutral terms, cloud resources, operations and exchanged parameters.

- The **Cloud Patterns Level** represents the semantic description of agnostic and vendor-dependent Cloud Patterns realized through an OWL representation based on ODOL. It contains patterns at infrastructural level and at platform level.

- The **Application Patterns Level** represents the description of patterns describing the application to be ported. An **Application Pattern** is a composition of application components embodying application domain functionalities, services at PaaS and SaaS level, platform Cloud Patterns and resource configuration patterns.

4. Discovery and composition of cloud services

```
PREFIX paramOntology: <http://127.0.0.1/ontologies/
    Parameters.owl#>
PREFIX serviceOntology: <http://127.0.0.1/ontologies
    /CSontology.owl#>
SELECT ?equivalentParameter ?vendor ?service ?
    operation
WHERE {paramOntology:InstanceID paramOntology:
    isEquivalentTo ?equivalentParameter.
?operation serviceOntology:hasParameters ?
    equivalentParameter.
?service serviceOntology:hasOperation ?operation.
?service serviceOntology:hasVendor ?vendor}
```

Listing 1. Comparison of default Resources Configurations

```
PREFIX agn: <http://127.0.0.1/ontologies/
    AgnosticServices.owl#>
PREFIX map: <http://127.0.0.1/ontologies/
    MicrosoftAzure.owl#>
PREFIX cso: <http://127.0.0.1/ontologies/CSOntology.
    owl#>
PREFIX amz: <http://127.0.0.1/ontologies/Amazon.owl
    #>
SELECT    ?EC2Operation ?agnosticOperation ?
    AzureVMOperation
WHERE {    cso:Amazon_ElasticComputeCloud amz:
    hasMethod ?EC2Operation.
?EC2Operation agn:isEquivalentTo ?agnosticOperation.
cso:Azure_VirtualMachines map:hasMethod ?
    AzureVMOperation.
?AzureVMOperation agn:isEquivalentTo ?
    agnosticOperation.}
```

Listing 2. SPARQL query to retrieve compatible operations exposed by EC2 and Azure VirtualMachines services

In order to correctly assess the equivalence among parameters, a hierarchical agnostic ontology is used to define generic parameters, while service-specific ones are grouped according to the target platform in self-contained ontologies. Such vendor specific ontologies are connected via object properties to the agnostic one, which acts as an intermediary. A simple query which can be run against the semantic representation has been reported in Listing 1: the query retrieves all the vendor specific parameters which are equivalent to the agnostic **InstanceID**, and also shows the services, operations and vendors they belong to. Results of the query have been reported in Table 1. Knowing how parameters are interrelated is not sufficient to enable the discovery of equivalent Cloud Services. This is clearly visible in the query shown in Listing 1, since another ontology is used. Such an ontology represents an additional layer of our representation, referred to as the **The service layer**, with definitions of Cloud Services, their operations with a list of input and output parameters (which realize a direct connection to the bottom layer) and their relationships. As for the parameters layer, here we use an agnostic ontology which describes generic cloud services and arranges them hierarchically. Such agnostic services are used as a common ground for comparison among vendor specific services, in

order to assess their equivalence. For an instance, let's suppose we want to retrieve all services which are equivalent to **Amazon EC2**: the simple query reported in Listing 2 is able to answer our request, thanks to the **aKindOf** property which allows to recognize the category a specific service falls in.

Results of the query are reported in Table 2.

Table 2. Results of Query in Listing 2

Service	Provider
OpenStack_Nova	OpenStack
Oracle_Compute	Oracle
Azure_VirtualMachines	Azure
RedHat_CloudForms	RedHat

Once we have retrieved one or more equivalent (by functionality) services, we can run an additional query to assess equivalences between their operations. Query in Listing 3 reports an example of such an interrogation, with the comparison between the methods exposed by the services EC2 and Virtual Machine provided by Amazon and Azure respectively. Agnostic concepts are used, as in previous examples, as a common ground for comparison.

```
PREFIX on:<http://127.0.0.1/ontologies/CSOntology.
    owl#>
SELECT ?service ?provider
WHERE { on:Amazon_EC2 on:aKindOf ?category.
?service on:aKindOf ?category.
?service on:offeredBy ?vendor}
```

Listing 3. SPARQL query to retrieve services and/or appliances compatible with EC2

While vendor-specific patterns' participants are directly connected to the implementing services described in the corresponding platform-specific ontology, agnostic ones refer to the agnostic ontology. In this way, it is possible to retrieve, for a vendor-specific pattern, a list of equivalent solutions provided by another platform. Furthermore, from an agnostic definition, it is possible to build the vendor-specific one, just by selecting the target platform.

Figure 2 reports a schematic representation of the ontologies used by **SCoPE** and of their relations. The three layers mentioned in this section are arranged horizontally from left to right. The lower part of the figure reports vendor specific ontologies, while the upper side contains agnostic ones. The ontologies for the description of both agnostic and vendor specific services and patterns are written using the OWL language. However, the pattern ontologies are also complemented with a set of OWL-S descriptions, for the orchestration of the involved Cloud services. In this way, such patterns describe both the structure of a Cloud application and the different interactions that occur among the involved services. Thanks to the connections which have been defined between the different ontologies through ad-hoc data-type and object properties, it is possible to retrieve information on the services via the simple SPARQL queries we have shown.

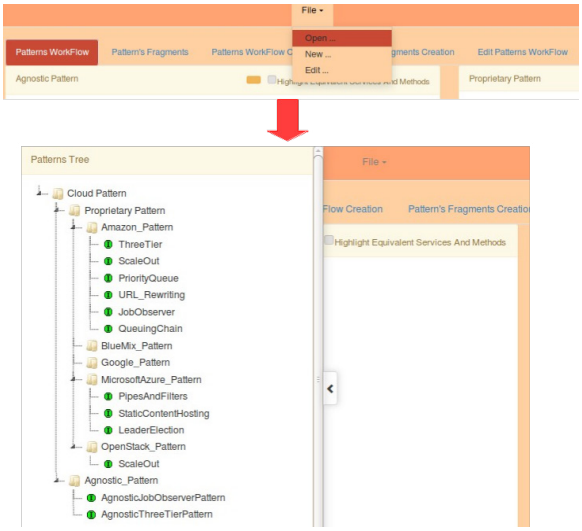

Figure 2. The Pattern Catalog as shown by **SCoPE**

5. The SCoPE Interface

In this section we will dive through the different functionalities offered by **SCoPE**, by showing its graphical interface and the actions a user can perform while interacting with it. The graphical interface is composed of two menus, located on the upper side of the interface's window, and a central panel which is used to draw Patterns and compare or compose them. The uppermost menu is used to load or save the Patterns configurations, while the lower menu is composed of tabs which display different panels, with specific commands according to the task they serve to.

Comparing Patterns

The first functionality exposed to the user is represented by the possibility to browse the Cloud Pattern catalog, organized according to the vendor for which the specific pattern has been designed. In order to visualize the catalog, the user has to load the ontology using the upper menu: Figure 3 shows exactly what the user is presented after the Patterns' definitions have been loaded.

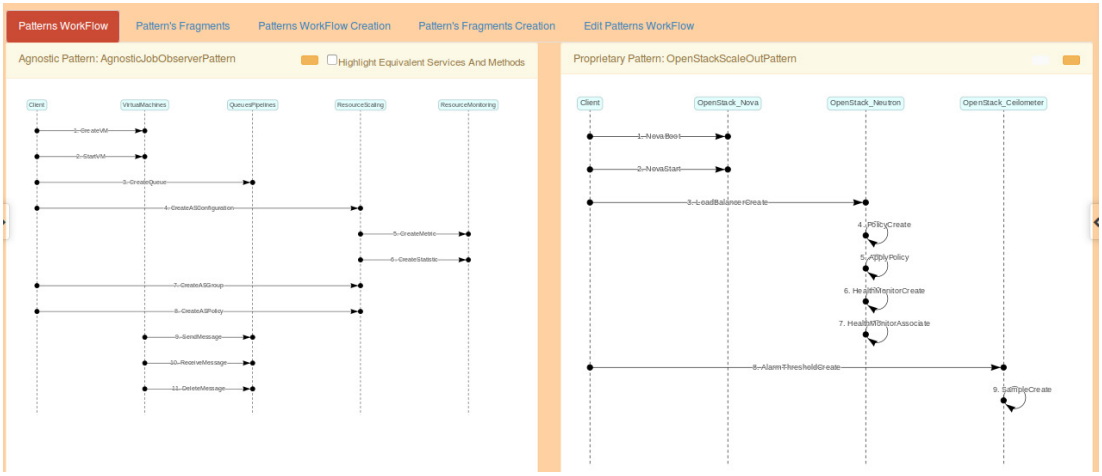

Figure 3. Comparison between Agnostic and Proprietary Pattern

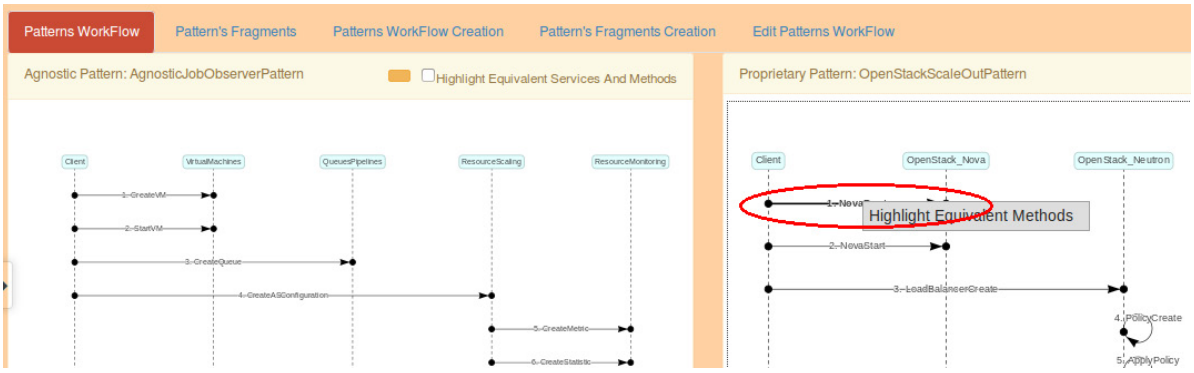

Figure 4. Requesting corresponding methods

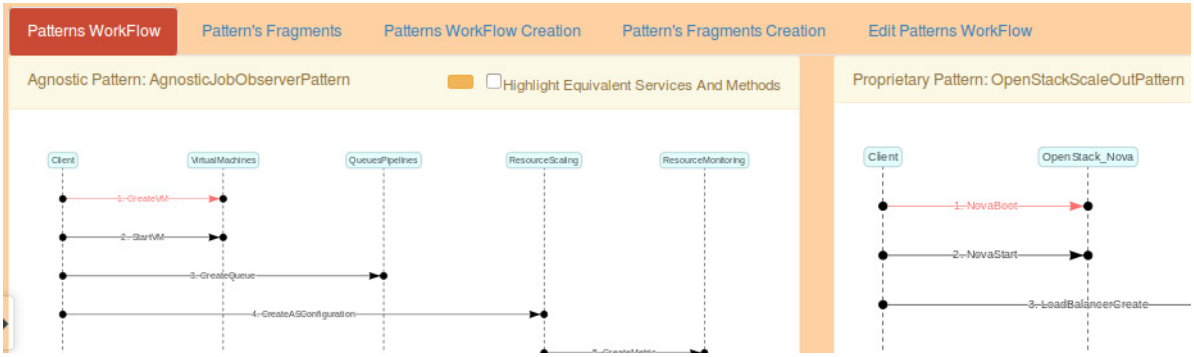

Figure 5. Highlighting corresponding methods in Patterns

The offered view is a tree-like organized catalog, in which Patterns are divided into Proprietary and Agnostic patterns, and further categorized according to their specific vendor. In Figure 3 it is possible to see the list of Patterns defined for Amazon, Azure and OpenStack.

The user can browse the catalog, and then he or she can select one of the Patterns and drag the corresponding name or green dot to the center of the **Patterns Workflow** Tab. Proprietary Patterns are automatically drawn on the right side, while Agnostic Patterns are always shown on the left. Figure 4 shows on the left side a representation of the Agnostic Job Observer Pattern, while on the right an Openstack specific version of the same Pattern has been drawn.

The Patterns are visualized similarly to a UML sequence diagrams, with Cloud Services depicted as actors and functionalities' calls as ordered, numbered actions, which describe the Patterns' workflow. If there exists a correspondence between elements of the compared Patterns, it is possible

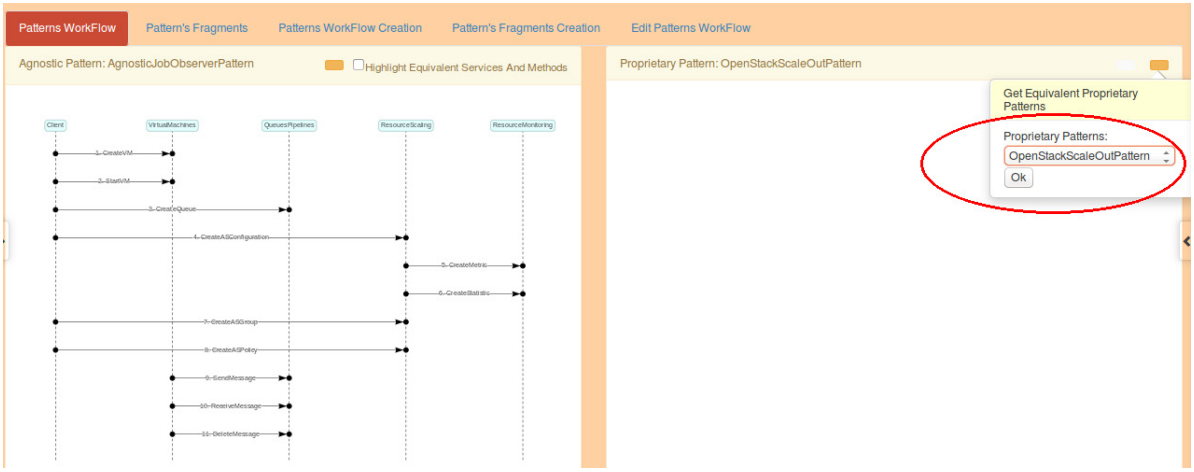

Figure 6. Equivalent Proprietary Patterns shown to the user

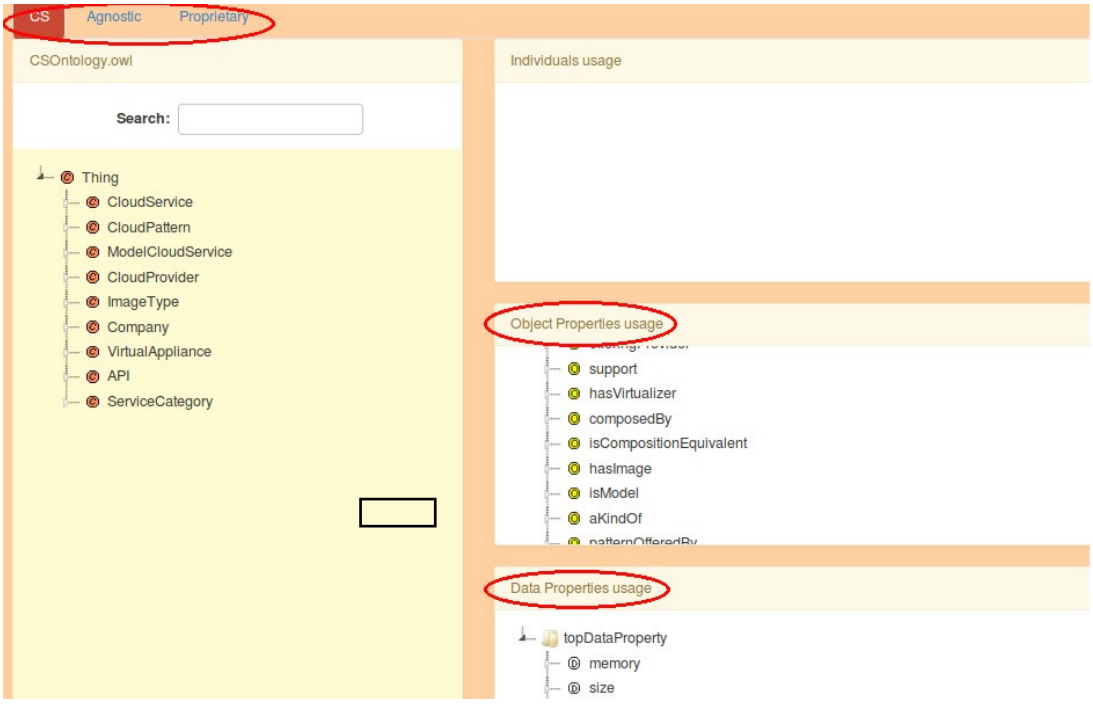

Figure 7. Browsing the Services ontology

to highlight it. Figure 5 shows how, by right-clicking on a specific method call, it is possible to highlight all the corresponding methods in the compared Pattern on the other side. In this case, the **NovaBoot** method in the OpenStack Job Observer Pattern has been defined as equivalent to the **StartVM** method in the Agnostic Job Observer Pattern, as shown in Figure 5.

The same highlighting can be done for servicesUntil now, we have supposed that the user has selected two equivalent Patterns on his/her own, but since we are considering the possibility that she is not a Pattern expert, we have implemented the possibility to automatically display equivalent Patterns, either Agnostic or Proprietary, has been selected.

Let's suppose that the user has selected the Agnostic Job Observer Pattern and has dragged it into the central Panel. Once the Pattern has been drawn on the left side, it is possible to select any proprietary equivalent Pattern by using a selection menu which appears on the left side, as shown in Figure 6.

Once the user has selected the desired Pattern, it is automatically drawn on the left side, bringing to the same state displayed in Figure 7, with the possibility to highlight corresponding elements.

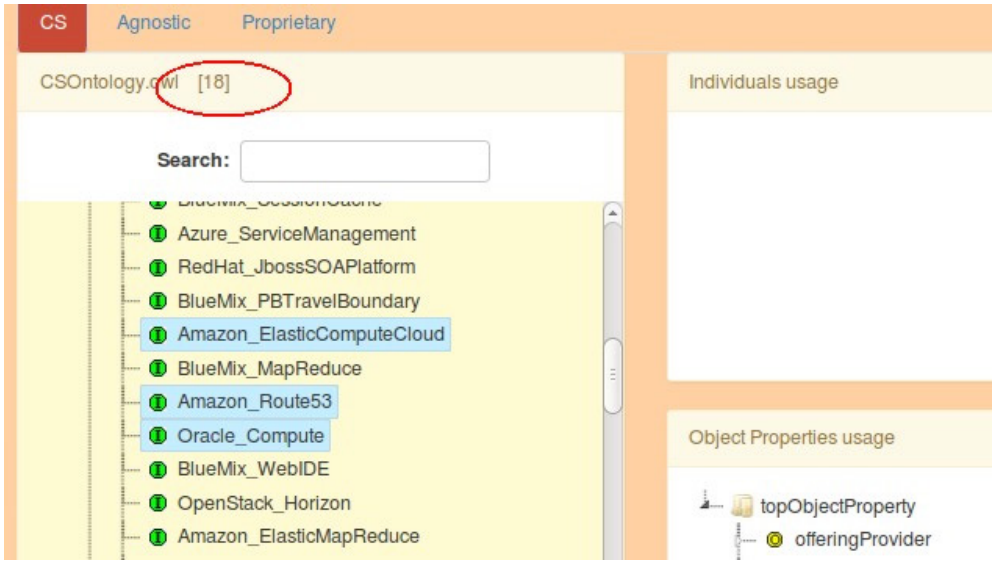

Figure 8. Browsing the Services ontology - Equivalent Services

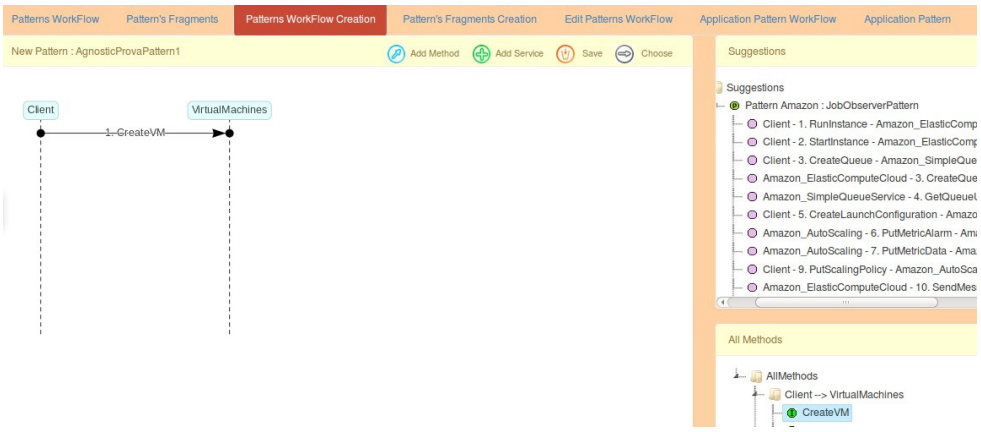

Figure 9. Adding Suggested methods and services to the Pattern

Browsing the Ontology to compare Services

The user can compare whole Patterns and highlight their equivalent services and method calls, but she can also browse the Services' catalog and obtain more information on the single components of each Pattern. The user can access the Services' catalog in two ways: by double-clicking on a service already present in a displayed Pattern, or by selecting a sliding window on the right side of the Pattern Workflow tab. The difference is simply that, in the first case, the double-clicked Service is directly highlighted in the catalog, while in the second case no Service is selected at all: in both cases, the user can browse the catalog freely.

If a Service has been previously selected, the browsing Panel focuses on it, as shown in Figure 8, where the **OpenStack Nova Service** has been highlighted. By right-clicking on a Service, it is possible to automatically discover all the ones that are equivalent to it according to its functionalities, via the **GetEquivalentServices** button. After pressing such a button, all the Service equivalent to the focused one are automatically highlighted, as shown in Figure 8. In this case, the user can read the number of equivalent services above the search box, while on the right side of the Panel specific information on the selected service (provider, exposed methods, input and output parameters) is displayed.

Create a Pattern's Workflow

Apart from browsing the existing catalog, the user can modify existing Patterns' workflows or create new ones from scratch. In both cases, she is supported by the **SCoPE** tool in choosing the Services and the Methods to use in the composition of the Pattern. By selecting the **Patterns Workflow Creation** tab, the user is

presented with a blank panel, with an initial **Client** Service acting as an initiator actor for the new Pattern's workflow. In particular, above the blank panel there is a button menu, composed of:

- An **Add Method** button, which is used to add a method call between two services
- An **Add Service** button to add a new Service
- A **Save** button to save the newly created Pattern Workflow into the catalog
- A **Choose** button which is used to add to the canvas one of the suggested relationships, methods or services from the rightmost panel.

Figure 9 reports an overview of the central panel on which new Patterns are created, and of the right Panels which provide suggestions regarding the services or methods which could be added to the canvas, according to what has already been placed on it.

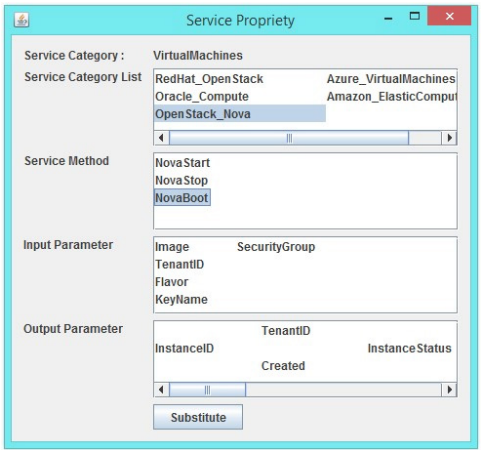

Figure 10. Adding Services to the Pattern

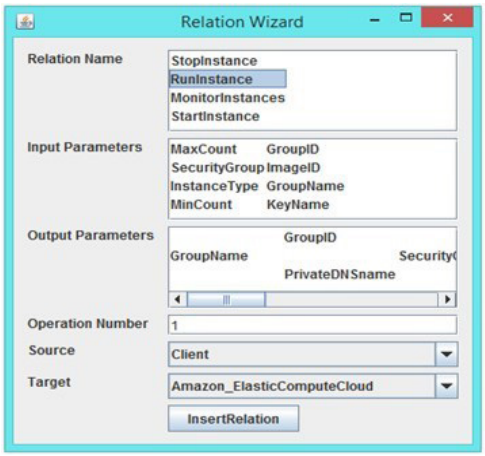

Figure 11. Adding Methods to the Pattern

If the user clicks on the Add Service button, a new wizard window is displayed, as shown in Figure 10. From this window, the user can select the Service category (Virtual Machines in the picture) and a list of candidates is displayed in the Service Category List box. When the user selects one of these services, all its exposed methods, together with its related input and output parameter, are listed in specific boxes. In this way, the user can indirectly browse the provided services by category and choose the one she deems suitable. If an existing service is double clicked, the Service Category is bound and cannot be changed, and a Substitute button is used instead of an Add one, to remove the old service and replace it with the new chosen one. If the user clicks on the Add Method button, a different wizard window is displayed, as shown in Figure 11. According to the Services already present in the panel, different Methods (or relations) can be added. The user can select the Source and Target of the method at the bottom of the new window, and available methods and parameters change accordingly in the respective boxes. The user can also select the exact number of the operation, so that she can specify the execution order of multiple method calls existing between the same services. When one or more Services are added onto the central panel, the Suggestion box on the right is populated. The box contains a list of triplets Service-Method-Service which are present in other Patterns, and which could be used with the services currently added to the creation panel. This is shown in Figure 9, where a list of possible triplets is shown according to the currently added Services. If the user does not want to just add new Services, she can use the **All Methods** box just below to select only methods which are compatible with the already chosen Services. Once a triplet or a method has been selected, the user can add it using the Choose button.

6. Conclusion and Future Works

In this paper a graph-based approach is presented and applied to a pre-existing semantic-based representation of Patterns, in order to express their correspondences and compositions. In particular, a Graphical tool named Semantic Cloud Patterns Editor (SCope) has been developed, to support users in:

- Discovering Cloud Services by browsing a semantic-based catalog

- Composing Cloud Services by following preexisting or newly defined Patterns
- Defining new Patterns either by following existing ones or from scratch
- Identifying correspondences among Services and Patterns' components.

In the future, we plan to extend the existing knowledge base in order to include a complete set of Cloud Services and Patterns, with correspondences and equivalences.

Also, future developments will feature graph matching capabilities to automatically map Patterns' components and integration with deployment and orchestration tools to automatize the deployment and execution of composed Patterns.

REFERENCES

1. AWS Cloud Design Patterns (2015). <http://en.clouddesignpattern.org>.
2. Cloud Patterns (2015). <http://cloudpatterns.org>.
3. Cloud Standards Customer Council. *Interoperability and portability for cloud computing: a guide*. Accessed in October 2017. <http://www.cloudstandardscustomercouncil.org/CSCC-Cloud-Interoperability-and-Portability.pdf>.
4. TOSCA Technical Committee et al. *Topology and orchestration specification for cloud applications (tosca)–committee specification 01*. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>.
5. Cloudformation templates. <http://aws.amazon.com/cloudformation/aaw-cloudformation-templates/>.
6. Openstack services (2015). Accessed in October 2017. <http://www.openstack.org/software>.
7. Windows Azure Application Patterns (2015). <http://blogs.msdn.com/b/jmeier/archive/2010/09/11/windows-azure-application-patterns.aspx>.
8. Cojoacă, E. S. D., Popescu, M. A. & Ambăruş, G. C. (2017). Cloud Computing Technology to Assist Government in Decision Making Process, *Studies in Informatics and Control*, *26*(2), 249-258. ISSN 1220-176.
9. Di Martino, B. & Cretella, G. Semantic technology for supporting Software Portability and Interoperability in the Cloud-Contributions from the mOSAIC Project, *Cloud Computing and Big Data*, *23*, 66. DOI 10.3233/978-1-61499-322-3-66.
10. Di Martino, B., Esposito, A. & Cretella, G. Semantic Representation of Cloud Patterns and Services with Automated Reasoning to support Cloud Application Portability, *IEEE Transactions on Cloud Computing*, *PP*(99), 1-1. doi: 10.1109/TCC.2015.2433259.
11. Di Martino, B., Cretella, G. & Esposito, A. (2015). Methodologies for cloud portability and interoperability, *Cloud Portability and Interoperability: Issues and current trends*, 15-44. Springer. DOI: 10.1007/978-3-319-13701-8.
12. Di Martino, B., Tasquier, L., Venticinque, S. & Aversa, R. (2013). Agent Based Application Tools for Cloud Provisioning and Management. In Yousif, M. & Schubert, L. (eds.) *Cloud Computing. CloudComp 2012, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, *112*. Springer, Cham. DOI https://doi.org/10.1007/978-3-319-03874-2_4.
13. Gutierrez-Garcia, J. O. & Sim, K. M. (2013). Agent-based cloud service composition, *Applied intelligence*, *38*(3), 436-464.
14. Leymann, F., Fehling, C., Retter, R., Schupeck, W. & Arbitter, P. (2014). *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer Science & Business Media.
15. Popa, S.* & Vaida, M. F. (2016). A Practical Strategy for ERP to Cloud Integration, *Studies in Informatics and Control*, *25*(3), 375-384. ISSN 1220-1766.
16. Zeng, C., Guo, X., Ou, W. & Han, D. (2009). Cloud computing service composition and search based on semantic, *Cloud Computing*, 290-300. Springer.
17. Zou, G., Chen, Y., Yang, Y., Huang, R. & Xu, Y. (2010). Ai planning and combinatorial optimization for web service composition in cloud computing. In *Proc international conference on cloud computing and virtualization* (pp.1-8).