Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 97 (2016) 104 – 113

Procedia Computer Science

CLOUD FORWARD: From Distributed to Complete Computing, CF2016, 18-20 October 2016, Madrid, Spain

Semantic Techniques for Multi-Cloud Applications Portability and Interoperability

Beniamino Di Martino[a], Antonio Esposito[a,*]

[a]*Second University of Naples, Department of Industrial and Information Engineering, Via Roma 29, Aversa (CE), 81031, Italy*

Abstract

The composition of Cloud Services to satisfy customer requirements is a complex task, owing to the huge number of services that are currently available. The advent of Big Data and Internet of Things (IoT), which rely on Cloud resources for better performances and scalability, is pushing researchers to find new solutions to the Cloud Services composition problem. In this paper a semantic-based representation of Application Patterns and Cloud Services is presented, with an example of its use in a typical distributed application, which shows how the proposed approach can be successfully employed for the discovery and composition of Cloud Services..

Keywords: Cloud Computing; Semantics; Cloud Patterns; Application Patterns; Big Data; Distributed Algorithms

1. Introduction

Cloud Computing is a fast evolving and spreading resource sharing paradigm, which is being adopted by a growing number of companies to provide competitive services and functionalities, and at the same time to reduce upfront investments and management costs. The rapid adoption of Cloud solutions has pushed Cloud providers to continuously enhance and update their offers, releasing into the market a huge set of services. While this variety positively affects the market, in terms of services' costs and quality, it can however represent a hindrance for customers who have to select the right services suiting their exact needs and then compose them to obtain the desired functionality. In many cases, owing to the lack of a shared standard for services' interfaces description and incompatibilities between the adopted data formats, it can be difficult to effectively compose Cloud Services and exploit their full functionality. To provide a guide to service composition and application deployment on Cloud platforms, Patterns have been described by private companies, such as Amazon[1] and Microsoft[2], or have been defined as a result of research efforts[3 4]. In this paper we show how, exploiting a Cloud Pattern or a composition thereof, it is possible to guide a customer in building

* Corresponding author. Tel.: +39-377-186-2406
E-mail address: antonio.esposito@unina2.it

her own Cloud application, regardless of her actual knowledge of the Cloud domain. In particular, we will show that, through a semantic representation of Application and Cloud Patterns, it is possible to support the development and deployment of Big Data, IoT and generally distributed applications.
The remainder of this paper is organized as follows: section 2 reports some existing approaches to Pattern description and a comparison with the **Topology and Orchestration Specification for Cloud Applications** (TOSCA) approach; section 3 introduces the Case Study used to describe our semantic approach; section 4 describes the semantic approach and how it is applied to the case study; section 5 draws conclusions and provides directions for future works.

2. State of the Art

A Design Pattern is defined as a general and reusable solution to a common and recurrent problem, within a given context in software design. The objective of a Design Pattern is to support developers in building their applications, suggesting fully functional solutions that are less error and bug prone than completely new implementations. Design Patterns provide both a static view of the architecture of the software, including its components (or participants) and their relationships, and a dynamic view of the interactions among such participants. The most famous Design Pattern catalogue is represented by[5], in which a set of 23 patterns has been introduced and deeply discussed. As of today, a number of Design Patterns catalogues exists for several purposes, like ontology creation[6 7] and definition of SOA-oriented applications[8 9]. Cloud Patterns can be considered as a particular Pattern category, focusing on the description of problems and solutions related to Cloud Computing. Cloud Patterns describe common aspects of cloud computing environments and of application design for cloud computing. They can be useful in understanding the changes to apply to an application, in terms of source code and architecture, in order to successfully migrate it from an in-house environment to the Cloud[10]. The use of Cloud Patterns for the design, implementation and management of Cloud Applications has been widely discussed[11 12 13].
Several online catalogues of Cloud Patterns have been published, and they are continuously updated. Vendor specific patterns, such as those published by Microsoft for their platform **Azure**[14] and by Amazon for **Amazon Web Services** (AWS)[1], are tailored for a target environment and provide optimized solutions for it. They provide many useful details regarding the actual Cloud components and services to use in order to deploy an application on the target platform, thus actively supporting developers in their work. Patterns have also been defined as a result of independent research efforts, such as those reported in the online catalogues accessible at[3] and[4]. Such patterns provide generic solutions, which are not bound to a specific platform and are therefore more flexible and seamlessly applicable to different targets. In the remainder we will refer to such patterns as **Agnostic Patterns**, because they are not related to a specific Cloud Platform and can virtually be applied to any target environment.

2.1. Orchestration and Composition of Cloud Services

In the past few years the orchestration and composition of cloud services has been the topic of several initiatives and research efforts. The mOSAic Fp7 project[15] 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. A similar 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. These approaches do not refer to a Pattern collection, and require users to develop and compose their applications manually. By using a Pattern based approach, as in ours, users can leverage pre-defined solutions which already offer info on the services' composition.
Non-academical research efforts have produced user oriented solutions, with immediate applications to real-world situations. For an instance, orchestration and composition of Cloud Services is also the focus of the OpensStack **Heat** project[17], which has developed an interesting template-based formalism going under the name of **HOT**. While being compliant with the AWS **CloudFormation**[18] template format, HOT is still limited as regards the supported services and general expressivity. TOSCA[19], 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. However, the support to existing Cloud Patterns is not natively provided, and the lack of semantic description does not allow the automatic service discovery, which is instead one of the main features we intend to provide with our representation.

3. Case Study

Clustering consists in an unsupervised classification that categorizes a set of objects into groups, which then contain similar entities. The use of non-hierarchical clustering techniques is well documented, in particular to build a simplified data representation of large data-sets. In this Section, we have chosen to analyse, through the proposed semantic model, the Batch K-Means algorithm and, in particular, its distributed version. The Batch K-Means algorithm belongs to the class of alternating optimization algorithms, as it alternates two optimization phases iteratively. The first phase, called the **assignment phase**, takes as input a dataset and a given set of prototype clusters, and assigns each element in the dataset to the nearest one. The algorithm is very generic, as it does not imply the use of a specific data structure for the representation of elements (or points) in the dataset. The developer can choose to represent the data in the way she prefers, as long as she chooses the distance criteria accordingly. The second phase, referred to as the **recalculation phase**, is run after the assignment phase has been completed. During this second phase, each prototype is recomputed as the average of all the points in the data set that has been assigned to it. Once the second phase has been completed, phase 1 is run again, and phase 1 and 2 are run iteratively until a stopping criterion is met. The distributed version known as **Distributed Batch K-Means** (DBK) follows exactly the same procedure, but it divides the initial data-set in blocks (again, there is no restriction on the data structure to use) and distributes it to several processing units, while information on the initial clusters are stored in a shared memory. The processing units execute the assignment phase and update the clusters. The recalculation phase is then executed by either a subset of the processing units employed for the assignment phase, or by dedicated processors. The results of the recalculation phase are stored in the same shared memory. While better algorithms exist for clustering, the Batch K-Means has very interesting and useful characteristics:

- It is a powerful pre-processing technique, as it allows for the classification of very large sets of data into smaller groups (prototypes).
- It has a low processing cost, proportional to the data size and the number of clusters.
- The Distributed Batch K-Means has the same results of the sequential version and is an embarrassingly parallel algorithm.
- Since no prescriptions are made on the employed data structures, the developer can select the one best suiting her needs. Obviously the processing code should be implemented accordingly.

Figure 1(a) reports the pseudocode of the algorithm, as it is described in[20].

In the following sections the semantic approach for the description of Patterns will be presented and applied to the DBK algorithm.

4. The semantic-based approach

In this Section we present the integrated representation of Cloud Services, appliances and Cloud Patterns we have devised. We use a number of terms (like agnostic Cloud Pattern, vendor-dependent Cloud Pattern, agnostic Cloud Service and vendor-dependent Cloud Service) that we are defining in the following. An **agnostic Cloud Pattern** is an architectural cloud solution to certain problems that does not rely on specific vendor services or platform characteristics. A **Vendor-Dependent Cloud Pattern** is a vendor defined solution to a cloud problem provided by the provider itself, that relies on specific services offered by the specific cloud platform. A **vendor-dependent Cloud Service** is basically a service offered by a cloud vendor that can be effectively used to utilize a cloud platform functionality. A **Vendor-Agnostic Cloud Service** is an abstraction of a vendor-dependent Cloud Service and represents a generic

```
Code run by the computing unit m
Get same initial prototypes $(\omega_k)_{k=1}^{K}$
repeat
    for t = 1 to n do
        for k = 1 to K do
            compute $||z_t^m - \omega_k||_2^2$
        end for
        find the closest prototype $\omega_{k*(t,m)}$ to $z_t^m$
    end for
    for k = 1 to K do
        Set $p_k^m = \#\{t, z_t^m \in S^m \& k^*(t,m) = k\}$
    end for
    for k = 1 to K do
        $\omega_k^m = \frac{1}{p_k^m} \sum_{\{t, z_t^m \in S^m \& k^*(t,m) = k\}} z_t^m$
    end for
    Wait for other processors to finish the for loops
    if i==0 then
        for k = 1 to K do
            Set $\omega_k = \frac{1}{\sum_{m=1}^{M} p_k^m} \sum_{m=1}^{M} p_k^m \omega_k^m$
        end for
        Write $\omega$ into the shared memory so it is available for other processing units
    end if
until the stopping criterion is met
```

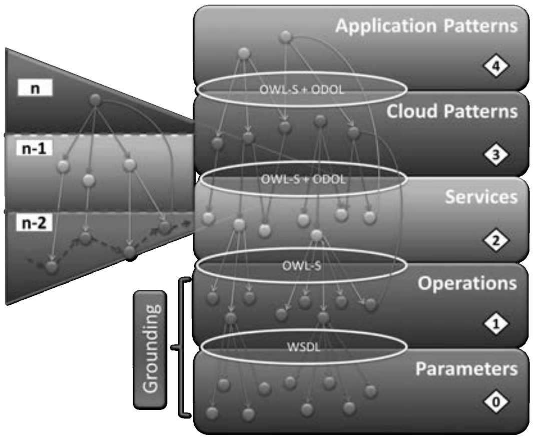


Fig. 1: (a) Pseudocode of the DBK algorithm; (b) The Conceptual Layers

functionality offered by a Cloud Service. 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 relationships among concepts of the same level are represented in addition to inter-level relationships. The five conceptual levels are the following (Figure 1(b)):

- The **Parameters Level** represents the description of the data type exchanged among services as input and output of the operations.
- The **Operations Level** represents the syntactic description of the operation and functionalities exposed by the Cloud Services (described through WSDL); it provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.
- 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 functionality exposed and the underlining details. This level contains also the semantic description of the agnostic Cloud Services exposed through an ontology that expose 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. 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.

Each level **n** of our conceptual model is realized by elements of one level below (**n-1**) through the composition of elements of two levels below (**n-2**). Direct edges among nodes belonging to levels distant more than two levels do not exist. In particular: the concept of application pattern is realized through the composition of elements of the Cloud Patterns level that are composed together using concepts of the Cloud Service level; the concept of Cloud Pattern is a composition of Cloud Services that are orchestrated using concepts of the operation levels; a Cloud Service is a set of operations exposed that are built using concepts of the parameters level.

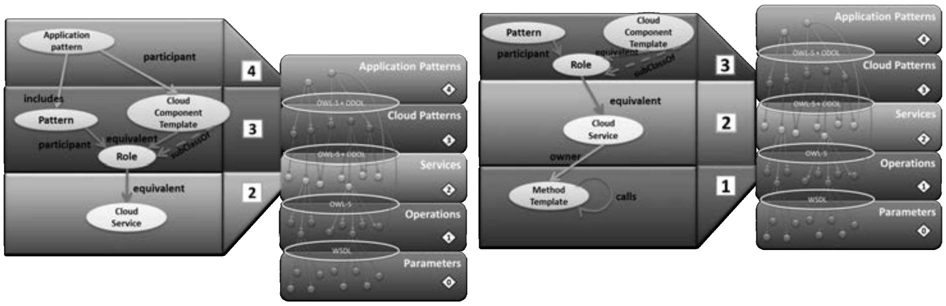


Fig. 2: (a) Semantic description of an Application Pattern; (b) Semantic description of a Cloud Pattern

4.1. Application Pattern Representation

The root element of this representation is the Application Pattern. The main classes and relations describing this concept are reported in (Figure 2a): the class **Cloud Component template** and the relationship **participant** that represent the application components, that means the conceptual building blocks of the application's workflow; the class **Pattern** and the relationship **includes** that represent the platform Cloud Pattern and the resource configuration pattern that are included in the overall architecture of the application; each pattern is composed of a number of **Roles** that can be embodied by the **Cloud Component Templates**. Each Role and Cloud Component template can be equivalent to a Cloud Service and a Cloud Pattern can be built through the composition of the Cloud Service components.

An Application Pattern has been created to represent the DBK, with the following participants:

- **Assigner** is the processing component that takes care of assigning data to the first prototype groups. Several instances of such a component can exist.
- **Re-calculator** is the processing component that runs the optimization phase, by recalculating the prototypes. Several instances of such a component can run simultaneously.
- **Shared Memory** represents the shared memory used by the different processing units to communicate information on the clusters and on data.

All such participants are represented as individuals of the **ApplicationComponent** Owl Class defined in our semantic model, and are connected to a participant of the agnostic **Map-Reduce** pattern.

4.2. Cloud Pattern Representation

A Cloud Pattern is a subgraph including concepts of levels 3, 2 and 1 (Figure 2b). The concept of **Pattern** is realized through a number of Participants that are represented in the concept of **Role**. A particular subconcept of the concept **Role** is the class **Cloud Component Template** that embodies the role of a participant represented by a Cloud Service. Each *Cloud Service* holds its own methods and by means of the composition of them the realization of a Cloud Pattern takes place. In particular, as the DBK algorithm can be easily parallelized via an **Iterated Map-Reduce**, the selected Cloud Pattern is represented by **Map-Reduce**. Please refer to Figure 3 for a graphical representation of such relations. Indeed, the DBK Application Pattern **includes** the agnostic Map-Reduce Pattern, and the correspondences among their participants are highlighted by the **equivalent** property. In the same way, equivalences between the Map-Reduce components and agnostic Cloud Services, as shown in Figure 3, are modelled. Please note that the user who wants to add a new Application Pattern, such as the DBK algorithmic Pattern used in this example, to the knowledge base only has to provide connection between the Application Pattern itself and one or more agnostic Cloud Patterns. If no connection is possible (that is, no known pattern can be used to cloudify the application), or if one or more component of the original application cannot be mapped to some pattern participants, then a direct link to an agnostic Cloud Service is needed.

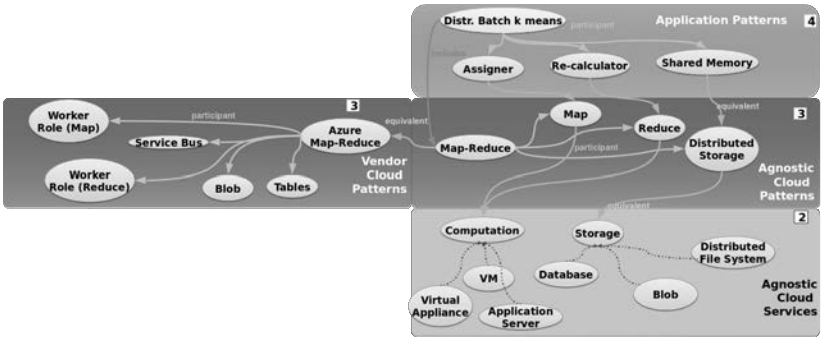


Fig. 3: Semantic representation of the Distributed Batch K-Means algorithm

4.3. The Entity level

To support the semantic description of services and patterns the representation is flanked by an **Entity Level** containing also a classification ontology that represents the upper level to classify Cloud Services based on the service category they belong to, concepts representing the operation and parameters exchanged. In particular this Entity Level contains the **Cloud Provider Ontology** and the **Cloud Service Ontology**. The Cloud Provider Ontology describes the concepts related to the cloud provider offering with classes and relationships, that means all the resources offered by the cloud provider and all the operations available on them, including the parameters used as input and output of the operations. This cloud vendor-dependent description is annotated semantically with cloud provider-independent concepts, thus making the representation agnostic. The Cloud Service Ontology (Figure 4) defines a classification of services offered by different Cloud providers and identifies the most common service categories currently provided.

The structure of the Cloud Services ontology consists of different OWL classes, among which the following ones are particularly interesting:

- The **Service Category** class, used to specify the typology of the described services and better categorize them.
- The **Cloud Service** class, whose individuals represent specific Cloud Services.
- The **Virtual Appliance** class, representing ready-to-use software solutions, offered by a specific company, that can be also classified alongside the described Cloud Services in this same ontology.

The main object property present in the Cloud Services ontology is the **aKindOf** property, that connects Cloud Service individuals with Service Category instances. Other object properties allow to specify, for each Cloud Service individual, the API language supported, the provider offering that specific service and the reference service model, in terms of Infrastructure, Platform or Software as a Service layers. Particularly relevant are data properties that expose Virtual Appliances architectural aspects, specifying the minimum resources necessary to instantiate a particular appliance. These properties can be utilized to match a Virtual Appliance and its minimum requirements with the default resource configurations offered by a single or multiple Cloud providers. Further information about the Cloud Service ontology can be found in[21], where all of its classes, individuals and properties have been discussed in detail. This cross layer is used to enrich the five levels with semantic information. In particular it is used to annotate semantically the Cloud Services. There are two ways to represent functionalities of a Cloud Service: the first way provides an extensive ontology of functions where each class in the ontology corresponds to a class of homogeneous functionalities; the second way is to provide a generic description of a function in terms of the state transformation that it produces (outputs from inputs). OWL-S supports both as the **Service Profile** module provides a high level description of services as a transformation to one state to another but also enables to describe a specific class of capabilities for the service. More precisely, a Service Profile provides two types of information: the first one is a functional description of

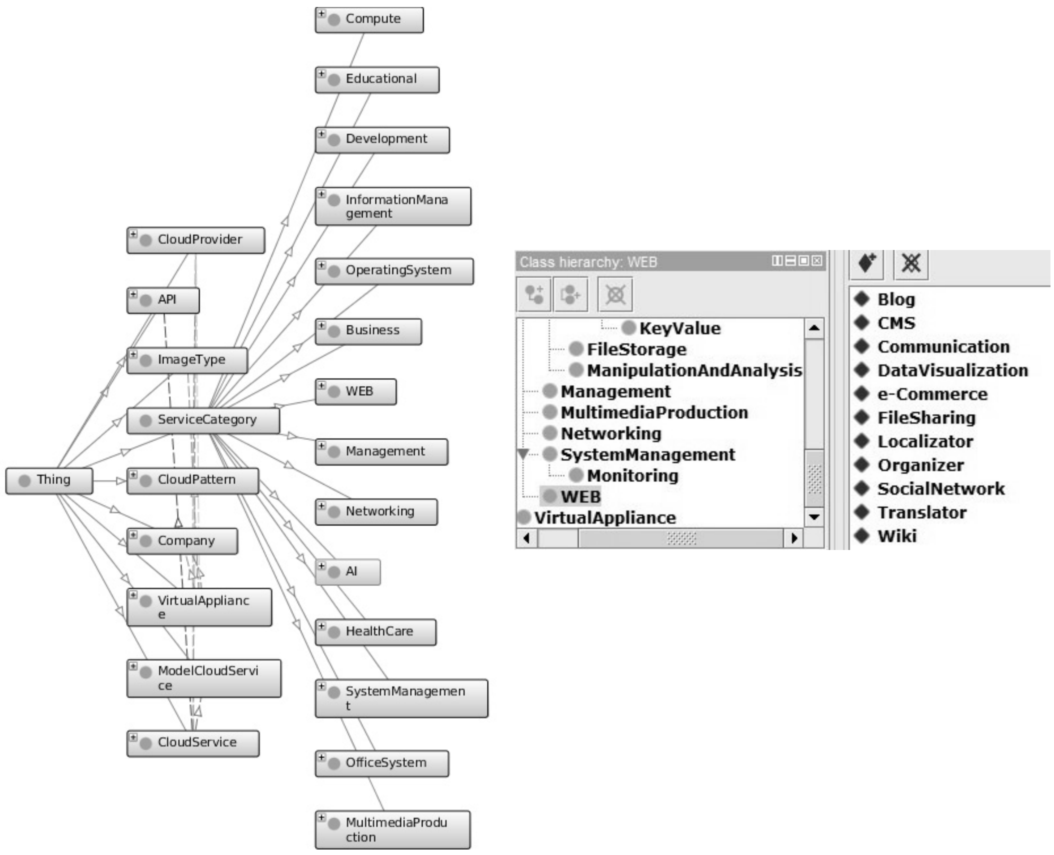


Fig. 4: The Cloud Services Ontology

the service in terms of the transformation that the service produces, the second one is a set of non-functional properties that specify constraints on the service provided. For our representation we exploit both functional and non-functional properties. The functional properties link the services elements to concepts of the Cloud Provider Ontology while the non-functional properties link the service to concepts described in our Cloud Service Ontology or other ontologies.

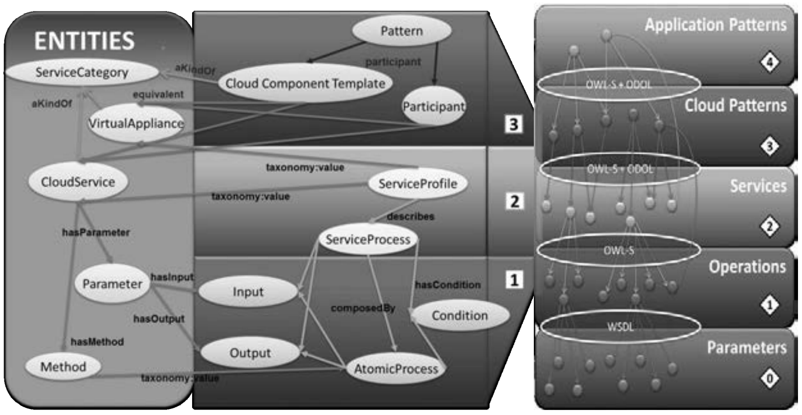


Fig. 5: Entities to Enrich the Semantic Description

```
SELECT ?agnosticComponent
WHERE { patternOntology:DBK patternOntology:participant ?DBKComponent.
        ?DBKComponent patternOntology:equivalent ?agnosticComponent }
```

Fig. 6: SPARQL query to retrieve DBK equivalent agnostic cloud components

```
SELECT ?azureComponent
WHERE { patternOntology:ReCalculator patternOntology:equivalent ?agnosticComponent.
        ?agnosticComponent patternOntology:implementedBy ?azureComponent.
         ?azureComponent patternOntology:hasVendor patternOntology:Azure
        }
```

Fig. 7: SPARQL query to retrieve DBK equivalent cloud component having Azure as a target

Table 1: List of actual services that can be used to deploy the ReCalculator component of the DBK pattern

Agnostic Service	Vendor Specific Service	Vendor Name
Virtual Machine	EC2	Amazon
	Nova	OpenStack
	VirtualMachines	Azure
	Oracle Compute	Oracle

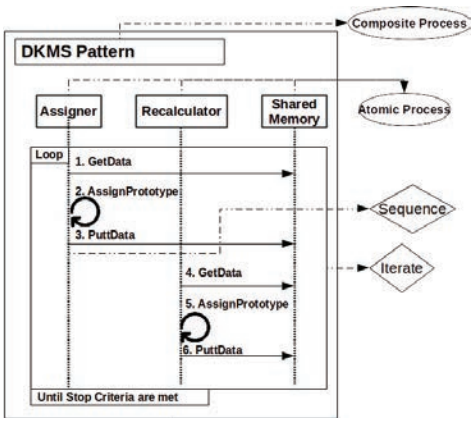
Using such ontologies it is possible to annotate the participants of the Cloud Patterns in order to retrieve the services which can be used to implement them. As an instance, in figure 3 the **Map** agnostic participant of the Map-Reduce Pattern is connected to the **Computation** category, which is specialized by agnostic Cloud Services, among which we find **VM** (Virtual Machine).

In order to retrieve the agnostic components corresponding to the DBK algorithm, it is possible to query the semantic representation via a simple SPARQL query, as reported in Figure 6.

Whilst the retrieval of agnostic components can be useful to have a generic representation of the algorithm that can then be adapted to the actual target platform, it is necessary to provide more links among the agnostic and vendor specific services. Since the knowledge base already contains information on the equivalences occurring between the agnostic services and the vendor specific ones, the only task left to the user is to select the desired target platform. When a user has selected the target platform, it is possible to retrieve vendor specific components to implement the original algorithm. As an instance, the SPARQL query reported in Figure 7 determines the Azure components needed to implement the **Re-Calculator** participant.

Whilst the query reported in 7 is specific for Azure services, there are a number of other available vendors that offer similar/compatible components. Table 1 reports all the services compatible with the ReCalculator component, limited to the **Virtual Machine** agnostic service, with the providing vendor.

Since equivalences among the operations exposed by the agnostic components and the vendor specific one are encoded in the knowledge base, together with information on the parameters they exchange, it is possible to suggest to the user how the selected services can interoperate. In particular, via the equivalences among the parameters needed to call the operations exposed by the Cloud Services, the system can suggest the user how the services can be composed. Some providers actually offer integration services to connect their own services with components running on external platforms or private systems (Hybrid Cloud). As an instance, the IBM Bluemix platform exposes the **Secure Gateway** service[22]. When such services are available and their use is needed to guarantee the compliance of the proposed composition to some non-functional requirement expressed by the user, the knowledge base provides directions on how to configure them. Again, simple SPARQL queries can be used to retrieve the necessary information. The considerations made so far imply that the developer decides to implement the Application Pattern by composing services from different providers or published by a vendor for that a specific Cloud Pattern has not been defined. Conversely, if a vendor specific Cloud Pattern corresponding to the original Application Pattern exists, and it employs the target platform selected by the user, then the service composition is already encoded in the knowledge base. In the example described in Figure 3, an Azure implementation of the Map-Reduce pattern exists, so the user could use it to deploy the application on the Microsoft platform. Since equivalences among the vendor specific components described in the pattern and agnostic concepts are always encoded in the knowledge base (such equivalences are not



Provider	Services	Patterns
Amazon	35	15
BlueMix	20	8
Azure	23	6
OpenStack	20	15
Google	10	4

Fig. 8: (a) OWL-S description of the DKMS Pattern; (b) Number of services and patterns supported by Cloud Composer

reported in figure to avoid confusion due to links overlapping), the user can decide to substitute one or more of the original services by selecting equivalent ones, by using a variation of the query in Listing 7.

4.4. Pattern Workflow

For the behavioural representation of patterns we use an OWL-S document describing the overall workflow of the pattern, thus defining how the different participants communicate and relate to each other dynamically. Every OWL-S document defines a **Composite Process**, that refers to a particular individual in the ontology if we are describing its internal behaviour, or to the behaviour of the whole pattern. The Composite Process formalizes the element's behaviour by introducing a series of simple atomic services and a composition of these services using language native flow control structures. Every service can declare a connection to the reference ontology, using tags contained in its profile description. In particular, we can exploit the property **serviceCategory** and its attributes. Participants' methods are necessary to define the pattern behaviour, since they describe how its different elements communicate and interact to change the system state. Every method represents a possible interaction between participants and their composition follows a specific order, defined by the particular behaviour of the owner, and is enforced by conditions imposed on their execution. A specific service is defined for every method of every participant in the pattern. All atomic services correspond to a profile and an atomic process description. The profile is then used to connect the specific service to the individual whose method is being described. Preconditions can be stated for every single atomic process, thus indicating the assumptions under that the corresponding method can be called and executed in the pattern.

Figure 8a shows the behaviour of the DBK pattern described through OWL-S constructs. The whole Pattern is an instance of a Composite Process, while its participants are instantiated as Atomic Processes. A Sequence control structure is used to represent the consecutive execution of the methods exposed by the Patterns' participants, while a Iterate component represents the iterative execution of the Assignment and Re-calculation phases.

5. Conclusions

The Cloud Computing scenario is steady evolving, with new Cloud services being launched onto the market continuously. Furthermore, each provider tends to use its own terminology in order to differentiate from others and try to gain new pieces of the market. Due to this complex scenery, a categorization of services to support the choice of the right solution, as well as a computable description of services and patterns that enables the comparison and the mapping between different providers' services, proves to be extremely appealing. The application of semantic web technology to cloud computing can bring advantages at different levels, within and across cloud platform. The approach we have presented consists in the creation of a machine readable and queryable Knowledge base, consisting of a set of interrelated OWL ontologies, that describe Cloud Services and APIs/methods used to invoke them (together with their relative parameters), Cloud Patterns, and finally application patterns. This semantic representation of Cloud Patterns, Cloud Services and Virtual Appliances supports the development of portable and potentially interoperable Cloud solutions and the porting of existing applications to multiple Clouds. This formalization enables the comparison of solutions provided by different vendors, despite possible differences in their semantics, as long as they refer to a common shared representation. While the number of services currently described through our representation is quite satisfactory, the number of patterns actually represented is still limited and needs to be improved. Also, new definitions of agnostic patterns are needed. Table 8b reports the number of services and patterns currently represented, organized by provider. In the near future we will provide further pattern definitions and will also support the automatic deployment of the supported pattern, which is still not possible as of now. In order to ease the developing of applications through the use of our representation, a user-friendly GUI is being implemented and a prototype version of it has already been described in papers submitted to other conferences. By using the prototype, the user is supported in the design and composition of patterns and in retrieving the information needed to actually deploying them, without the need to actually write and execute SPARQL queries manually.

The prototype will also be used to properly evaluate the approach, and to test if the information provided by the formal description of Patterns, Services and their composition are is sufficient for their actual deployment in a real world scenario, or if further improvement is needed.

Acknowledgment

This paper was written within the European Union TOREADOR project (TrustwOrthy model-awaRE Analytics Data platfORm). Granting authority: European Union. Call: H2020-ICT-2015. Topic: ICT-16-2015 (Big data - research). Type of action: RIA. Grant agreement no.: 688797, Starting date: 1st January 2016, Ending date: 31th December 2018.

References

1. AWS Cloud Design Patterns, `http://en.clouddesignpattern.org` (2015).
2. Microsoft Azure patterns, `http://msdn.microsoft.com/en-us/library/dn568099.aspx` (2015).
3. Cloud Computing Patterns, `http://www.cloudcomputingpatterns.org/` (2015).
4. Cloud Patterns, http://cloudpatterns.org (2015).
5. G. Erich, H. Richard, J. Ralph, V. John, Design patterns: elements of reusable object-oriented software.
6. Ontology Design Patterns [online], `http://ontologydesignpatterns.org/` (2015).
7. A. Gangemi, Ontology design patterns for semantic web content, in: The Semantic Web–ISWC 2005, Springer, 2005, pp. 262–276.
8. M. Endrei, et al., Patterns: service-oriented architecture and web services, IBM Corporation, International Technical Support Organization, 2004.
9. SOA Patterns, http://www.soapatterns.org/ (2015).
10. C. S. C. Council, Migrating Applications to Public Cloud Services: Roadmap for Success, `http://www.cloudstandardscustomercouncil.org/Migrating-Apps-to-the-Cloud-Final.pdf` (2013).
11. C. F. F. Leymann, et al., Cloud computing patterns.
12. B. Di Martino, et al., Semantic and Agnostic Representation of Cloud Patterns for Cloud Interoperability and Portability, in: Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on, Vol. 2, IEEE, 2013, pp. 182–187.
13. A. Homer, et al., Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications, Microsoft patterns & practices, 2014.
14. Windows Azure Application Patterns, `http://blogs.msdn.com/b/jmeier/archive/2010/09/11/windows-azure-application-patterns.aspx` (2015).
15. D. M. B., C. G, Cloud Computing and Big Data, Vol. 23, 2013, Ch. Semantic technology for supporting Software Portability and Interoperability in the Cloud - Contributions from the mOSAIC Project, pp. 66–78. doi:10.3233/978-1-61499-322-3-66.
URL `http://dx.medra.org/10.3233/978-1-61499-322-3-66`
16. C. Zeng, X. Guo, W. Ou, D. Han, Cloud computing service composition and search based on semantic, in: Cloud Computing, Springer, 2009, pp. 290–300.
17. Openstack services, `http://www.openstack.org/software` (2015).
18. Cloudformation templates, `http://aws.amazon.com/cloudformation/aws-cloudformation-templates/`.
19. T. T. Committee, et al., Topology and orchestration specification for cloud applications (tosca)–committee specification 01.
20. M. Durut, Distributed clustering algorithms over a cloud computing platform, Ph.D. thesis, Télécom ParisTech (2012).
21. B. Di Martino, G. Cretella, A. Esposito, Towards a unified owl ontology of cloud vendors' appliances and services at paas and saas level, in: Complex, Intelligent and Software Intensive Systems (CISIS), 2014 Eighth International Conference on, IEEE, 2014, pp. 570–575.
22. Bluemix secure gateway service, `https://console.ng.bluemix.net/catalog/services/secure-gateway` (2016).