Cloud Services Composition Through Semantically Described Patterns: A Case Study

Beniamino di Martino, Giuseppina Cretella(✉), and Antonio Esposito

Second University of Naples, Via Roma 29, Aversa, CE, Italy
beniamino.dimartino@unina.it,
{giuseppina.cretella,antonio.esposito}@unina2.it

Abstract. With the proliferation of Cloud services and the huge number of Cloud offers currently available in the IT market, it can be difficult for customers to understand which one fits their need. Patterns, if correctly applied to the design and development of Cloud applications, can ease programmers' burden and reduce errors and bugs in application implementation. In this paper we use a methodology, based on the semantic representation of Cloud patterns, Cloud services and applications, to support users in developing Cloud oriented software meeting their explicit requirements.

Keywords: Cloud computing · Services composition · Cloud patterns · Semantics · Ontology · OWL

1 Introduction

The Cloud Computing scenario is a plethora of always new and changing Cloud proposals, platforms and capabilities. Furthermore, each provider tends to use its own terminology in order to differentiate itself from others and try to gain new market shares. Thus, it can be difficult for users to clearly understand which services are more suitable for their requirements and needing. In such a situation, also portability and interoperability of Cloud applications and services is badly influenced, making it difficult to make services and resources from different providers to cooperate in order to provide specific functionalities. In this paper we show how, by using a semantic-based representation of Patterns, Cloud services and Virtual appliances, based on the work presented in [16,17,20], it is possible to describe a classical application and support users in deploying it to the Cloud. Such a uniform, integrated and machine-readable representation aims at supporting the migration of applications to the Cloud and at easing the procedures needed to port them across different platforms.

The paper is organized as follows: Sect. 2 reports related works and offers some insight on the technologies used in our representation; Sect. 3 briefly describes the methodology we have applied to the description of patterns and services; Sect. 4 describes the use case and the application of our methodology; finally, in Sect. 5 we report some consideration on the present work and address future directions of research.

A. Celesti and P. Leitner (Eds.): ESOCC 2015 Workshops, CCIS 567, pp. 404–418, 2016.
DOI: 10.1007/978-3-319-33313-7_31

2 State of the Art

The classification and categorization of Cloud Services has been the topic of many research efforts [12,13], which have tried to systematize their exposed functionalities, operations, parameters and service models. A freely navigable online taxonomy has been provided by the OpenCrowd [6] consortium, which categorizes Cloud Services according to both their service model (IaaS, PaaS or SaaS) and application context. Nevertheless the criteria followed to categorize each service are not clear, as well as the limitations and controls under which the taxonomy creation is performed. Machine readable standards for services' representation and orchestration have been proposed and approved: among these, remarkable results have been accomplished by the **Topology and Orchestration Specification for Cloud Applications** (TOSCA - an OASIS standard) and by the orchestration template language **HOT**, developed by **Openstack** [7] within the **HEAT** project. TOSCA describes 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. HOT is a new template format, compliant with the **CloudFormation** Template defined by Amazon, which details everything that is required for orchestration and it is written in YAML. A Comparison of such formats is available in [19]. Semantic based approaches have been considered and applied in order to overcome limits related to automated processing and reasoning, caused by differences in semantics and syntactic. A semantic ontology is a formal, machine readable knowledge representation of a set of domain-related concepts and the relationships between them. It is used to reason about the properties of that domain and may be used to efficiently describe it by providing a shared vocabulary. The Web Ontology Language (OWL) [30] is a semantic mark-up language for publishing and sharing ontologies on the World Wide Web. A number of ontologies related to cloud computing emerged in the past few years. The authors of [10] provide an overview of Cloud Computing ontologies, their applications and focuses. Some ontologies are used to describe Cloud resources and services, classify the current services and pricing models or define new types of Cloud services [15,34]. Many research efforts have been carried-out to develop ontologies to achieve interoperability among different Cloud providers and their services: different solutions have been discussed in [32]. A remarkable result has been reached by the mOSAIC cloud ontology [31] developed for the mOSAIC platform [21]. Such an ontology has also been adopted by the IEEE P2302 Working Group (Intercloud) [4] for the development of the Intercloud Interoperability and Federation (SIIF). In [11] Bernstein and Vij present the InterCloud Directories and Exchanges mediator to allow collaboration among Cloud vendors which work on an ontology of Cloud Computing resources to deal with providers heterogeneity. In [33], the authors propose a resource selection mechanism based on the users' requirements regardless of where the services are hosted. Han and Sim [25] propose a Cloud service discovery system that uses Cloud ontologies, matchmaking and agents to determine the similarities between and among services. In [14] the author present an ontology-based discovery system to help users in deploying their virtual

appliances on the most appropriate IaaS providers, based on their definition of QoS requirements.

2.1 Cloud Patterns

Design Patterns, defined as a general and reusable solution to a common and recurrent problem within a given context [23], have been used for a long time in software design and development. Their objective is to support developers in the design of their application, reducing design and developing time, known errors and bugs. As of today, a number of Design Patterns catalogues exist for several purposes, like ontology creation [5,24] and definition of SOA-oriented applications [8,22]. Recently both Cloud vendors and independent researchers have developed catalogues of Cloud Patterns, which define architectural solutions for designing and developing efficient applications on the Cloud. Remarkable examples are represented by the vendor specific catalogues developed by **Microsoft** [9] for **Azure** and by **Amazon** for **Amazon Web Services** [1]. Independent catalogues are instead retrievable at [2,3]. In the remainder of this paper, we will refer to the formers as to **Vendor Specific** patterns, since they are bound to the specific platform they have been designed for. The latter will be referred to as **Agnostic** patterns, since they provide generic solutions, which are not bound to a specific platform and are therefore more flexible and applicable to different targets. The use of Cloud patterns for the design, implementation and management of Cloud Applications has been widely discussed in the literature [26,27,29].

Our semantic representation focuses on Patterns, and Cloud Patterns in particular, because they can provide the necessary information to build an application's architecture on a platform and, in the case of vendor-specific ones, also to deploy such an application and configure the services which compose it. As an instance, suppose that a user needs to monitor a certain applications, which is running on a server owned by a Cloud Provider. Without knowing which provider hosts the server, it is possible to leverage an agnostic Cloud pattern to know in advance the components needed. The pattern **Usage Monitoring** provided in [3] defines the main components needed to monitor the usage of a simple Cloud Service, by providing access to a set of collected metrics via a portal, which collects information through a ad-hoc monitoring service. The different possible interactions between the user and the system are also described in the pattern, as shown in Fig. 1.

Since the agnostic pattern is extremely general, it is possible to determine a whole set of possible implementations for a certain Cloud platform, each one addressing a specific issue. For example, if we consider the AWS platform and we want to deploy a generic monitoring application on it, the **Monitoring Integration Pattern** describes the architecture and the components needed. As it is shown in Fig. 2, the pattern points out the Amazon services needed to deploy a monitoring application and also shows how to actually connect it to the services to monitor.

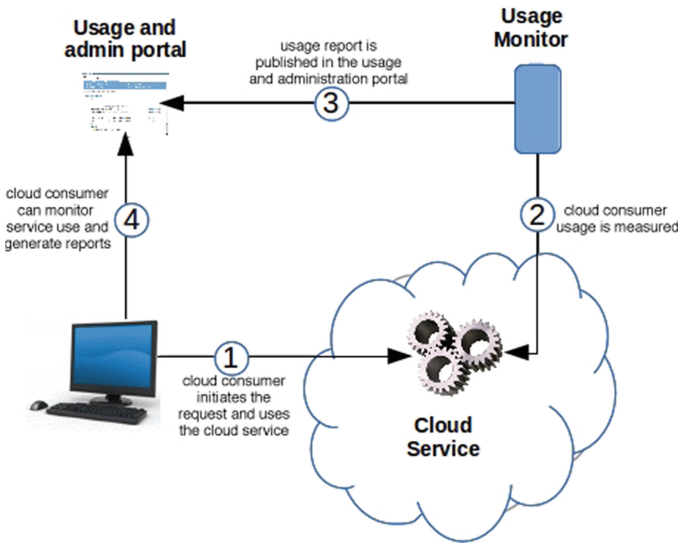

Fig. 1. Usage Monitoring pattern

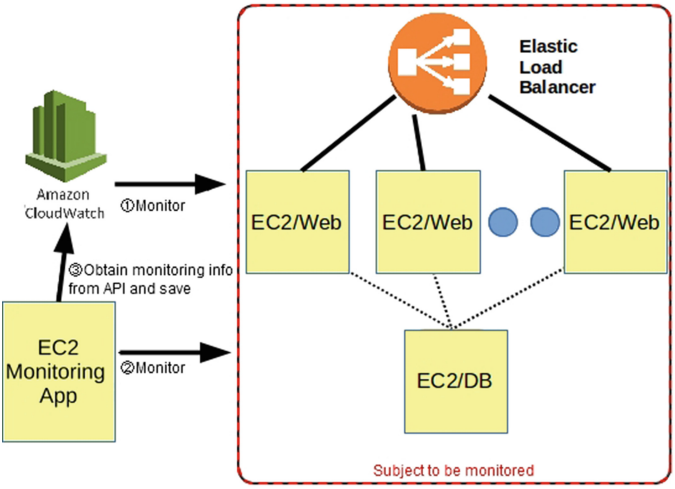

Fig. 2. Monitoring Integration Pattern

3 Methodology Description

In this section we briefly introduce the integrated representation of cloud services, appliances and cloud patterns we have devised. For a more detailed description of such a semantic representation, please refer to [16].

The model we use to describe Cloud related concept is based on a graph representation, which can be divided into five conceptual levels. Each level is

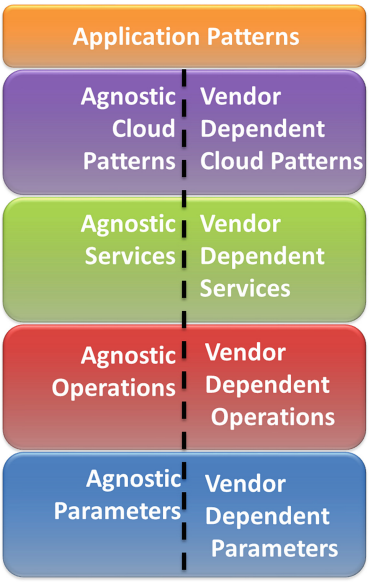

Fig. 3. The Conceptual Layers

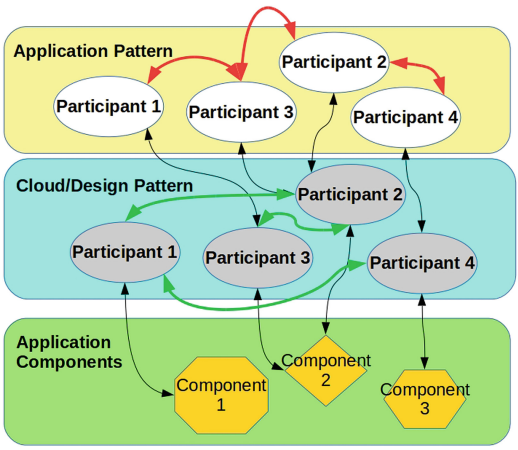

Fig. 4. Application Pattern Composition

connected to the others through relationships, which enable the Cloud services discovery and composition. Figure 3 reports the layered organization of our representation:

- The **Parameters Level** contains the semantic description of the data types exchanged among Cloud services as input and output of their exposed operations.
- The **Operations Level** provides a syntactic description of the operations and functionalities exposed by a cloud service, in a machine readable format. In this way, it is possible to automatically retrieve information on how to call the service and interact with it.
- The **Services Level** provides a semantic annotation of Cloud services, which are organized according to a hierarchical classification described in [18]. Both vendor specific services and agnostic ones are represented at this level.
- The **Cloud Patterns Level** represents the semantic description of agnostic and vendor dependent Cloud patterns realized through an OWL representation. The patterns described here are composed of services delivered at infrastructure and platform level.
- The **Application Patterns Level** contains information on high level patterns, which describe entire applications with their components. Such patterns are general enough to be applied to different contexts, not necessarily Cloud related: in this way, it is possible to describe a generic application through one or more of Application patterns and then retrieve the components to be used for its implementation using the lower levels of our representation. The

organization of such components and how they should interact to achieve the required functionality in Cloud are described in the Cloud Pattern level.

3.1 Pattern Representation

The core element of our representation is the Application Pattern, which is used to describe general applications with their architectural details and information on the interactions taking place among their components. Application patterns can consist of multiple Design/Cloud patterns, and their components are connected through relationships expressing the equivalence between their participants and potential implementing services. As shown in Fig. 4, each of the application patterns' participants is connected to a Cloud/Design patterns' component: the matching is not necessarily one to one, since two elements of an Application pattern could be embodied by the same participant in a composing Cloud pattern and vice-versa. The same applies to the mapping between Cloud/Design patterns' participants and application components. In the semantic-based representation we use to describe Patterns, the participants are represented by individuals of the OWL class **ComponentTemplate**, while the connections between elements of different layers is obtained via instances of an object property **equivalent**. The connections existing within the same layer, representing workflow and interactions among patterns' participants, are represented via OWL-S [28] native constructs. In order to keep trace of the Patterns involved, of their interconnections and participants, each pattern is represented by an instance of the **Pattern** OWL class, while the object properties **hasParticipant** and **includes** are used to connect a pattern to its owned elements and to other contained patterns, respectively. The representation we use for pattern description is applied to both vendor specific and agnostic patterns, in order to have a homogeneous definition of them. The only difference between agnostic and vendor specific pattern representations resides in the nature of the application components used to realize them: vendor specific patterns will be connected to real components, while agnostic patterns will be composed of agnostic services.

3.2 Services Representation

As we have stated in Sect. 3.1, patterns are connected to application components which can be potentially used to implement them. In particular, since we are addressing a Cloud-oriented implementation, such components will be represented by Cloud Services or Virtual Appliances. Such components are defined in the Services layer, which contains both representation of vendor specific and agnostic services. Agnostic services act as place-holders for services' functionalities, and constitute a hierarchical architecture against which vendor services are annotated. In this way, equivalences between several services and their functionalities can be automatically inferred, through the explicitly declared equivalence with agnostic concepts and logical rules.

Physically, the agnostic services are all defined within a single **Cloud Service ontology**; vendor specific services are organized in self-contained ontologies, independent of each other, which import the agnostic one to annotate their services. The annotation is possible via a set of three object properties:

- **exactEquivalence** defines and exact correspondence between the vendor specific and agnostic service.
- **plugin** is used if a service has not a single correspondence, but it exposes functionalities offered by more than one service.
- **subsumes** represents the inverse situation of *plugin*, that is when the functionality exposed by a service can be obtained only by composing two or more different agnostic or vendor specific services.

All these are sub-properties of a more generic *equivalent* object property. The description of the input and output parameters of vendor specific services relies on OWL-S descriptions, which also leverage an underlying parameter ontology for the disambiguation of similar variable types and the support to logical inferences.

Figure 5 reports a schematic representation of the different ontologies involved in the services representation: the top **Agnostic Service Description Ontology** contains abstract descriptions of services, parameters, operations and resources, which are used as a common ground for comparisons among concepts described in the bottom **Cloud Provider Ontologies** and **Cloud Services OWL-S Descriptions** which, instead, contain platform-specific information. Our knowledge base contains a specific Cloud Provider ontology and OWL-S description for each Cloud platform (AWS, Azure, OpenStack, Google AppEngine, BlueMix) we have considered. Categorization of services is provided by the **Cloud Services Categorization Ontology**, which is used as a bridge between the agnostic descriptions and the OWL-S representations. The connections shown in the figure are obtained through OWL object properties which assess the equivalence among services, parameters and operations. Such properties enable the free navigation of the ontology framework, making it possible to rapidly determine how to replace one or more services and operation calls when necessary.

4 Case Study

The application of Design and Cloud pattern to software development can ease and speed-up programmers' work: common problems that can be encountered in designing and developing a new application can find immediate solutions in the appropriate pattern. The case study we propose in this section aims at showing how, with our semantic base representation, it is possible to effectively support programmers in choosing and applying the needed patterns to the development of a new Cloud-oriented software and/or the migration of an existing application to a Cloud platform.

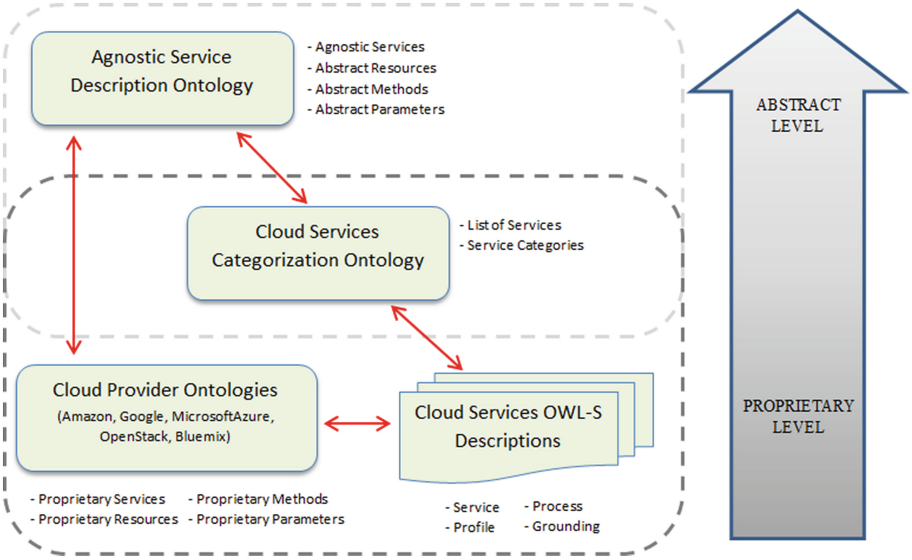

Fig. 5. Ontology Organization

Each of the steps we are showing is executed through a prototype graphical tool, which hides all the SPARQL queries which are automatically run against our knowledge base. Such a tool is still at its early stages of development and will not be shown here.

The example we are taking in consideration regards the complex information system needed to manage a railway reservation web-site. Figure 6 reports a schematic representation of the main components of such a system:

- the **Reservation front-end** that provides a user friendly web interface to customers, allowing them to interact with the system.
- the **Back-end system**, a complex component which in turn consists of an **Availability Checker system** (responsible to check tickets availability), a **Reservation system** (in charge of making the actual reservations) and a **Payment system** (that validates online transactions).
- a **Database** that holds information on trains, stations, timetables, purchased tickets and reservations.

4.1 Step 1: Selection of the Application Pattern

Using a graphical interface, a user can select the type of application she wants to build and deploy on the Cloud: for each application category there will be one or more specific application patterns which will be presented to the user, who can then refine the selection. The software we want to develop in our case study can be easily represented through a very generic Application Pattern, namely the

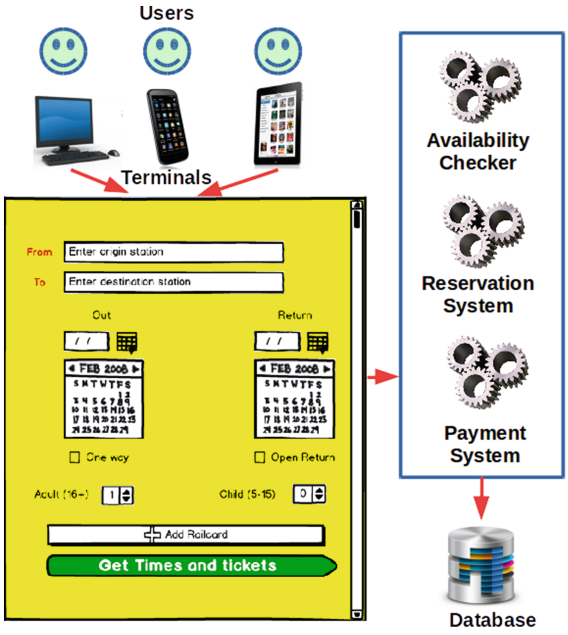

Fig. 6. The railway reservation system example

Reservation System pattern, whose components and correspondence with the agnostic **Three-Tier Cloud Application Pattern** [2] are shown in Fig. 7. The *Reservation System* application pattern can be easily applied to other similar applications, since it is sufficiently general and does not impose specific requirements. Our semantic-based representation can be also extended, so that new patterns can be built from the existing ones to add functionalities.

As soon as the user chooses the application type and a corresponding Application pattern is selected, the system automatically maps its components to an agnostic high-level Cloud pattern, which will represent the starting point for further refinements. The mapping shown in Fig. 7 simply matches the components of the application pattern, corresponding to the Reservation System used as an example, to the three major layers composing a three-tier application on Cloud: the correspondence is not one-to-one, as the Back-end system's components are all automatically matched with the Business Logic layer, where all the processing components belong to. At this point, the user can select one of the components of the high-level Cloud pattern in order to refine it further.

4.2 Step 2: Refinement of the Pattern's Components

Each of the three layers of the selected *Three-Tier Cloud Application Pattern* can be further refined via a mapping to other agnostic Cloud Patterns, which provide better instructions regarding the possible implementation of the needed

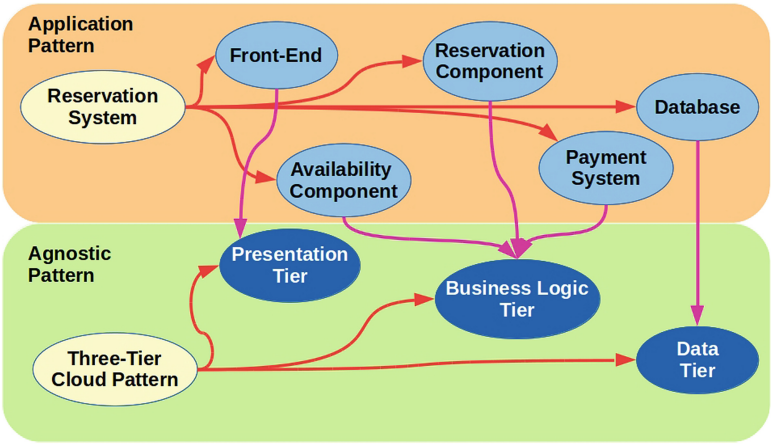

Fig. 7. Mapping between application pattern and agnostic cloud pattern

infrastructure. For each tier a composition of cloud patterns is suggested to implement and improve the single tier performance.

- the presentation tier (Fig. 8) can be implemented using a combination of an Elastic Load Balancer pattern (providing application scalability), a Stateless Component pattern (managing the status of the application's components) and a User Interface pattern (serving as a bridge between the synchronous access of the human user and the asynchronous communications used with other application components).
- the Business Logic tier (Fig. 9) is built via the combination of a **Processing Component Pattern** (providing elaboration capabilities), a **Stateless Component Pattern** and a **Data Access Component Pattern** (which guarantees access to the needed data).
- the Data Tier (Fig. 10) is the simplest of the three different layers as it is composed by a single agnostic patterns, namely the **Data Access Component Pattern**

Composing patterns can be shared among the different layers (the Data Access Component Pattern is used in both the Business Logic and Data tiers) and can also share participants: in the considered Presentation layer the **Elastic Load Balancer** participant is shared among two of its composing patterns.

4.3 Step 3: Selection of a Target Platform

The patterns used to compose the application layers are all agnostic ones: their components are not immediately implemented, as they need to be connected to an existing target platform. By following our pattern-based approach, it is possible to further refine the application composition by identifying vendor specific patterns with more detailed information regarding the implementation of the

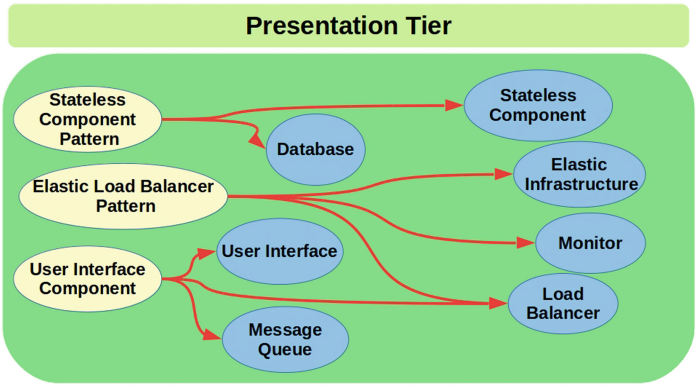

Fig. 8. Presentation tier composing patterns

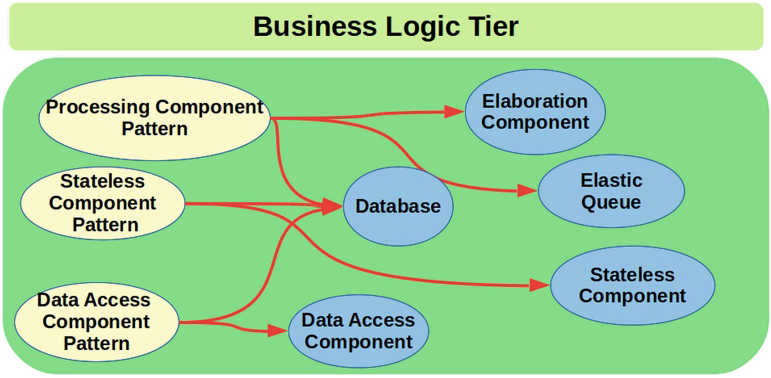

Fig. 9. Business tier composing patterns

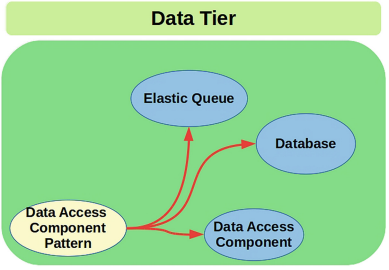

Fig. 10. Data tier composing patterns

different components. In this example, we will refer to Amazon Web Services as a possible target for the application deployment. By automatically analysing the semantic-based representation of the Amazon Cloud patterns catalogue [1]

we have devised, it is possible to retrieve equivalent patterns and to determine correspondences between their components. Figure 11 shows how the agnostic patterns and components identified for the Presentation tier are mapped to corresponding Amazon Cloud Patterns and services. In particular:

- Corresponding patterns are connected with black arrows: Stateless Component Pattern with State Sharing Pattern; Elastic Load Balancer Pattern with Scale-out Pattern.
- Blue arrows connect corresponding services/components: Stateless Component, Elastic Infrastructure and User Interface with EC2; Elastic Infrastructure with AutoScaling; Monitor with CloudWatch; Load Balancer with Elastic Load Balancer; Message Queue with Simple Queue Service.

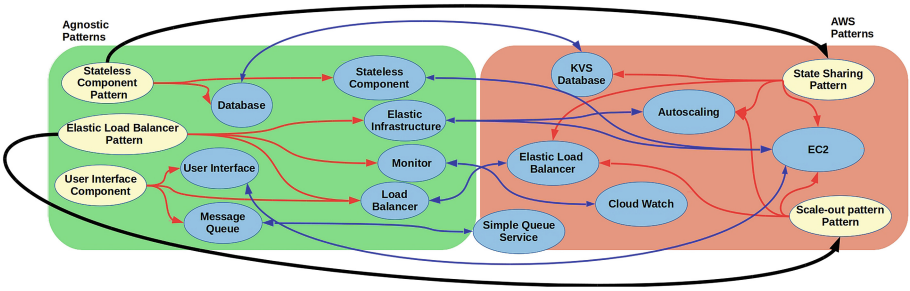

Fig. 11. Implementation of the Presentation tier with AWS patterns

The User Interface pattern does not correspond to any Amazon Cloud pattern, so the system simply identifies suitable services from the target vendor to implement them, without applying a specific pre-defined configuration. In this way, we can avoid the restrictions that the use of vendor specific Cloud patterns could impose to the implementation of an application. Once all the pattern elements have been mapped to a potential implementing Cloud service, the system supports users in making them interact, thanks to the information on the parameters and the operations they expose, which are contained in our semantic-enabled knowledge base. All the correspondences and mappings shown in figures are retrieved by means of SPARQL queries, which are run against our OWL-based knowledge base. Such queries leverage the properties described in Sects. 3.1 and 3.2, and are described in more details in [16]. Here we report, in Listing 1.1, an example of the SPARQL query used to retrieve all the vendor specific services which are equivalent to the agnostic *Elastic Load Balancer* used in the use case. The query first retrieves the category to which the *Elastic Load Balancer* service belongs, using the *equivalent* property defined in the pattern ontology (hence the prefix *patternOntology* we have used). Than, using the service category (*Type* in the query) we retrieve all the corresponding services via the *equivalent* property defined in the Cloud service ontology (hence the

cloudOntology prefix). The vendor name is retrieved via the **hasVendor** property. Table 1 reports the results of the query. Please note that all the retrieved components are Cloud services, apart from **ZeusExtensibleTrafficManager** which is a Virtual Appliance. The knowledge base contains information on the resource requirements needed to run the virtual appliance on a virtual machine and supports users in selecting the best suited offer on the target platform. The first three Cloud services belong to two different service categories, regarding balancing of application and network loads.

```
SELECT ?Component  ?Vendor ?Type
WHERE {patternOntology:ElasticLoadBalancer
   patternOntology:equivalent ?Type.
       ?Component cloudOntology:equivalent ?Type
       ?Component cloudOntology:hasVendor ?Vendor.
       }
```

Listing 1.1. SPARQL query to retrieve equivalent patterns' components from multiple vendors

Table 1. Partial results from query in Listing 1.1

Component	Vendor	Type
Openstack_Neutron	Redhat	NetworkLoadBalancing, ApplicationBalancing
Azure_Trafficmanager	WindowsAzure	NetworkLoadBalancing, ApplicationBalancing
Amazon_ElasticLoadBalancing	Amazon	NetworkLoadBalancing, ApplicationBalancing
ZeusExtensibleTrafficManager	Riverbed	NetworkLoadBalancing

5 Conclusion and Future Work

In this paper we have applied a semantic-based approach for the description of Cloud Patterns and services to a simple use case, in order to demonstrate the capability of such an approach to support users in developing Cloud oriented applications, without a deep and extensive knowledge of the entire Cloud Computing panorama. The different steps needed to deploy the example application to the Cloud have been described, and an example of the queries run against the proposed semantic knowledge-base has been provided. In the future, we are planning to develop the user friendly graphical interface to ease users' interactions with the system, and to include further services and patterns in our description.

Acknowledgements. This research has been supported by the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement n 256910 (mOSAIC Project), by PRIST 2009, "Fruizione assistita e context aware di siti archeologici complessi mediante dispositivi mobili"and CoSSMic (Collaborating Smart Solar-powered Micro-grids - FP7-SMARTCITIES-2013).

References

1. Aws cloud design patterns. http://en.clouddesignpattern.org
2. Cloud computing patterns. http://cloudcomputingpatterns.org
3. Cloud patterns. http://cloudpatterns.org
4. Ieee p2302 working group (intercloud). http://grouper.ieee.org/groups/2302/
5. Ontology design patterns. http://ontologydesignpatterns.org/
6. Opencrowd: Cloud computing vendors taxonomy. http://cloudtaxonomy.opencrowd.com/
7. Openstack services. http://www.openstack.org/software
8. Soa patterns. http://www.soapatterns.org/
9. Windows azure application patterns. http://blogs.msdn.com/b/jmeier/archive/2010/09/11/windows-azure-application-patterns.aspx
10. Androcec, D., Vrcek, N., Seva, J.: Cloud computing ontologies: a systematic review. In: The Third International Conference on Models and Ontology-Based Design of Protocols, Architectures and Services, MOPAS 2012, pp. 9–14 (2012)
11. Bernstein, D., Vij, D.: Intercloud directory and exchange protocol detail using XMPP and RDF. In: 2010 6th World Congress on Services (SERVICES-1), pp. 431–438. IEEE (2010)
12. Buyya, R., Vecchiola, C., Thamarai Selvi, S.: Mastering cloud computing: foundations and applications programming, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2013)
13. Catteddu, D.: Cloud Computing: benefits, risks and recommendations for information security. In: Serrão, C., Díaz, V.A., Cerullo, F. (eds.) IBWAS 2009. CCIS, vol. 72, p. 17. Springer, Heidelberg (2010)
14. Dastjerdi, A.V., Tabatabaei, S.G.H., Buyya, R.: An effective architecture for automated appliance management system applying ontology-based cloud discovery. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), pp. 104–112. IEEE (2010)
15. Deng, Y., Head, M., Kochut, A., Munson, J., Sailer, A., Shaikh, H.: Introducing semantics to cloud services catalogs. In: 2011 IEEE International Conference on Services Computing (SCC), pp. 24–31, July 2011
16. Di Martino, B., Esposito, A., Cretella, G.: Semantic representation of cloud patterns and services with automated reasoning to support cloud application portability. IEEE Trans. Cloud Comput. **PP**(99), 1 (2015). doi:10.1109/TCC.2015.2433259
17. Di Martino, B., Cretella, G., Esposito, A.: Semantic and agnostic representation of cloud patterns for cloud interoperability and portability. In: Proceedings of the IEEE Fifth International Conference on Cloud Computing Technology and Science (CloudCom 2013) (2013)
18. Di Martino, B., Cretella, G., Esposito, A.: Towards an unified owl ontology of cloud vendors appliances and services at PaaS and SaaS level. In: Proceedings of the 8th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2014), pp. 570–575 (2014)
19. Di Martino, B., Cretella, G., Esposito, A.: Defining cloud services workflow: a comparison between TOSCA and OpenStack hot. In: Proceedings of the 9th International Conference on Complex, Intelligent, and Software Intensive Systems, July 8th–July 10th 2015. IEEE (2015)
20. Di Martino, B., Esposito, A.: Towards a common semantic representation of design and cloud patterns. In: Proceedings of International Conference on Information Integration and Web-Based Applications & Services, p. 385. ACM (2013)

21. Di Martino, B., Petcu, D., Cossu, R., Goncalves, P., Máhr, T., Loichate, M.: Building a mosaic of clouds. In: Guarracino, M.R., et al. (eds.) Euro-Par-Workshop 2010. LNCS, vol. 6586, pp. 571–578. Springer, Heidelberg (2011)
22. Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., Newling, T.: Patterns: service-oriented architecture and web services. IBM Corporation, International Technical Support Organization (2004)
23. Catteddu, D.: Cloud Computing: Benefits, Risks and Recommendations for Information Security. In: Serrão, C., Aguilera Díaz, V., Cerullo, F. (eds.) IBWAS 2009. CCIS, vol. 72, p. 17. Springer, Heidelberg (2010)
24. Gangemi, A.: Ontology design patterns for semantic web content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 262–276. Springer, Heidelberg (2005)
25. Han, T., Sim, K.M.: An ontology-enhanced cloud service discovery system. In: Proceedings of the International Multiconference of Engineers and Computer Scientists, vol. 1, pp. 17–19 (2010)
26. Homer, A., Sharp, J., Brader, L., Narumoto, M., Swanson, T.: Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications. Microsoft Patterns & Practices (2014). ISBN:1621140369 9781621140368
27. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2014). doi:10.1007/978-3-7091-1568-8, ISBN: 9783709115671, 9783709115688
28. Mark, B., Jerry, H., Ora, L., Drew, M., Sheila, M., Srini, N., Massimo, P., Bijan, P., Terry, P., Evren, S., Naveen, S., Katia, S.: OWL-s: Semantic markup for web services. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
29. Martino, B.D., Cretella, G., Esposito, A.: Semantic and agnostic representation of cloud patterns for cloud interoperability and portability. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), vol. 2, pp. 182–187. IEEE (2013)
30. McGuinness, D.L., Van Harmelen, F., et al.: Owl web ontology language overview. In: W3C Recommendation, vol. 10, no. 10 (2004)
31. Moscato, F., Aversa, R., Di Martino, B., Fortis, T., Munteanu, V.: An analysis of mosaic ontology for cloud resources annotation. In: 2011 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 973–980. IEEE (2011)
32. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. (CSUR) **47**(1), 7 (2014)
33. Xu, B., Wang, N., Li, C.: A cloud computing infrastructure on heterogeneous computing resources. J. Comput. **6**(8), 1789–1796 (2011)
34. Youseff, L., Butrico, M., Da Silva, D.: Toward a unified ontology of cloud computing. In: Grid Computing Environments Workshop, GCE 2008, pp. 1–10. IEEE (2008)