

A Semantic Rule Based Expert System for the Discovery and Composition of Cloud Services

Beniamino Di Martino^{1,2,3,4,*}, Antonio Esposito^{1,2,†} and Michele Di Giovanni^{1,†}

¹Università della Campania Luigi Vanvitelli, Dipartimento di Ingegneria, Caserta, Italy

²Consorzio Interuniversitario Nazionale per l'Informatica, Rome, Italy

³Department of Computer Science and Information Engineering, Asia University, Taichung, Taiwan

⁴Department of Computer Science, University of Vienna, Vienna, Austria

Abstract

Cloud Services discovery and Composition represents a hot topic, due to the high number of Cloud Offers currently available, and the extreme variability in the exposed services' interfaces. Such a variability often results in interoperability and portability issues of applications and data among different Cloud Platforms, especially considering that Providers themselves often tend to lock-in their own users. Semantic technologies have indeed shown the capability to reduce such issues for users, by providing a common and shareable set of concepts, properties and relations that can be used as a ground to efficiently compare Services and adapt their exposed interfaces. In this work, a semantic description of Cloud Services and their agnostic composition is used to support users in implementing their own solution, by explicitly declaring their requirements. The proposed Expert System uses an existing semantic representation of Cloud Services, which has been further extended, and OWL-S descriptions of Agnostic Services compositions, which are then mapped to a set of Vendor Specific Cloud Services, to discover the required Services and finally compose a solution for the user. Users' requirements are analysed and processed by logical rules written in Prolog, which take care of building the actual composition. In this paper such and Expert System is described in terms of its main inputs and outputs, together with a description of the extended knowledge base used to represent the Cloud Services, and of the OWL-S definitions of Agnostic Services compositions. Also, the Prolog rules used to create the Provider specific Cloud Services compositions are reported.

Keywords

Cloud Services, Semantics, OWL-S, Prolog.

1. Introduction

Cloud Computing has been widely adopted, during past years, by several companies and enterprises, especially of small and medium dimension. The high availability and reliability of Cloud Services have strongly influenced and incentivised the adoption of Cloud solutions, also considering the virtually infinite amount of resources that Cloud Platforms can provide. This is strengthened by the relatively reduced necessity of initial investments that using Cloud

ITADATA2022: The 1st Italian Conference on Big Data and Data Science, September 20–21, 2022, Milan, Italy

*Corresponding author.

†These authors contributed equally.

✉ beniamino.dimartino@unicampania.it (B. D. Martino); antonio.esposito@unicampania.it (A. Esposito); michele.digiovanni@studenti.unicampania.it (M. D. Giovanni)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

solutions implies. One of the main drawbacks that several companies need to address, when approaching the Cloud Market, is represented by the extreme variety of available solutions, which is paired by a general lack of formal, homogeneous descriptions of offered services. This fact hinders the adoption of Cloud solutions, as the potentially of existing services is often hidden by their lack of description or overshadowed by the technical difficulties which arise when trying to build a complex solution, possibly merging several existing services. Indeed, interoperability and portability issues are quite common when users try to create a complex service starting from existing Cloud offers, especially when the composing services are exposed by different companies and Cloud platforms.

This is where the adoption of Semantic Technologies for the description of Cloud Services, their composition and orchestration, has demonstrated to be a possible solution to the lack of formal descriptions of Services interfaces, dramatically mitigating interoperability and portability issues.

Indeed, a Semantic representation of Services can be seamlessly exploited to achieve two different goals.

The first one consists in defining a simplified means for users to access descriptions of Services interfaces. This can be done by querying a Semantic knowledge base that, on the base of specific requirements expressed by the users themselves, proposes a set of available Cloud Services suiting such requirements.

The second one is instead focused on identifying possible compositions of identified Services which, once adequately orchestrated and eventually adapted, provide complex functionalities as requested by the user.

The two aforementioned goals can be treated separately, considering the Services Discovery as a preliminary, necessary step to Services Composition and Orchestration. Indeed, in this paper the two different aspects are treated separately, with a focus on the discovery goal. By leveraging a Semantic Description of Agnostic Cloud Services, obtained through an OWL Ontology which will be briefly introduced in Section 2, representing generic exposed functionalities that can be then composed together, the prototype tool proposed in this paper selects the actual Vendor Specific Services and propose a collection thereof, meeting the requirements expressed by the user. Logical rules expressed in Prolog allow for the identification of the required Services.

Furthermore, to enable the actual composition and orchestration, OWL-S is exploited: in this way the expert system prepares the ground for actual orchestration, by leveraging a composition of Services using the OWL-S formalism.

The remainder of this paper is organised as follows: Section 2 provides an introduction to the CSOntology and to OWL-S, which are necessary to understand the presented work; Section 3 describes the main components of the proposed Expert System; Section 3.1 provides an insight on the main extensions provided to the existing CSOntology; Section 3.2 focuses on the semantic representation of Agnostic Services, which is needed by the Expert System to define higher level Service Compositions; Section 3.3 describes the NFR class used to represent Non-Functional Requirements; Section 4 provides a description of the Prolog rules used by the Expert System; Section 5 closes the paper with some final considerations and remarks.

2. Background

Classification, categorisation, discovery and Composition of Cloud Services has to be considered as a hot topic, considering the consistent number of research efforts on its regards [1, 2]. Systematisation of exposed functionalities, operations, parameters and service models is not a trivial matter, but it is a necessary step to achieve discovery and composition. Often the criteria followed to categorise services are not clear, and provided semantic descriptions fail to cover all the necessary aspects, in particular when it comes to exposed methods and parameters. Machine readable standards for services' representation and orchestration exist, and have shown to be particularly reliable. The **Topology and Orchestration Specification for Cloud Applications** (TOSCA) standard, proposed by OASIS, provides the means to describe topologies of Cloud based web services, consisting in their components, relationships, and the processes that manage them, and allow for their orchestration. Services described through TOSCA cannot rely on Semantic descriptions, which renders them less discoverable and difficult to make interoperant. Recent research results, published in [3], have tried to fill this gap by providing a Semantic representation base for TOSCA services, by exploiting Semantic Web technologies to provide discovery functionalities. The ontology used to semantically enrich TOSCA, namely the **Cloud Services Ontology** (CSOntology), has been presented in [4], and it has been also used in the current work as a base for Cloud Services identification and discovery. The CSOntology, based on the work carried out within the mOSAIC project [5], and also successfully applied to the definition and application of Cloud Patterns to Service Composition [6], provides a functional categorisation of Cloud Services and Virtual Appliances. The ontology mainly divides Cloud Services into two groups, namely **Agnostic Services** which represent generic functionalities and are not specifically connected to a real Cloud Vendor, and **Vendor Specific Services** which instead collect all Services exposed by a provider. Agnostic Services can be considered as placeholders for services' functionalities, and constitute a hierarchical architecture against which Vendor specific services are annotated. Equivalences among several Services and their exposed functionalities can be automatically inferred, starting from the explicitly declared equivalences and exploiting SWRL rules contained within the CSOntology itself. Part of the work behind the creation of the Expert System described in this manuscript consists in the extension of the original CSOntology, as explained in Section 3.1. OWL-S [7] is a standard Ontology that has been explicitly created to semantically enrich Web Services and compositions thereof. It has been successfully used to represent the composition of Cloud Services, with limitations depending on the different technologies Web and Cloud Services use for deployment.

3. An Expert System for the Composition of Cloud Services

In this Section an Expert system supporting users in the Composition of Cloud Services is presented. The main objective of the System is to identify a possible composition of Provider Specific Cloud Services, starting from a high-level description of a composition built upon Agnostic Services, acting as a guide for the selection of actual Services. Such Agnostic Services are automatically selected by the Expert System by applying a set of Non Functional Requirements (NFRs) expressed by the user.

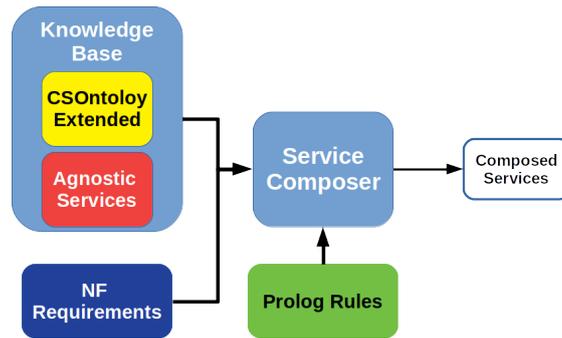


Figure 1: General design of the Expert System

The general design of the Expert System is reported in Figure 1. In order to work, the framework requires three different kinds of inputs, allowing it to apply logical rules and create the Service Composition:

- A knowledge base represented by an Ontology of Cloud Services, comprehending Comprehending TBox and ABox statements that can be used by the Expert System to identify the best suiting Services and implement their composition. This role will be covered by and extension of the CSOntology, which will be called **CSOntologyExtended** from here on, and that will be discussed in Section 3.1. In order to apply logical rules to such a semantic knowledge base, the software Thea [8] will be used to obtain a Prolog representation of it, which can be used to apply rules accordingly.
- A semantic description of Agnostic Services and compositions thereof, in terms of their functionalities and internal workflow. Such descriptions will be provided through OWL-S. As for the Cloud Services knowledge base, also the OWL-S representation will be processed by Thea in order to obtain a set of Prolog facts to work on.
- A set of logical Rules expressing the NFR, used to select the most suitable Services and compositions thereof, within the Knowledge Base and using the agnostic compositions as a base template. Such logical rules will be expressed in Prolog. The SWI-Prolog implementation environment is used to run the rules, in particular the ones described in Section 4.

The output of the System is instead represented by the exact composition of Cloud Provider Services to use to obtain the high-level functionalities required by the user, explicating the sequence of calls among the composing Services which is necessary to build the overall workflow of the composition.

3.1. The Cloud Services Knowledge Base

As already stated in Section 3, the Knowledge Base represented by the CSOntologyExtended has been built upon the CSOntology briefly introduced in Section 2 and described in detail in [4]. Figure 2 provides an overall view of the original CSOntology and of the actual CSOntologyExtended: on the right the ontology before the extension and on the left the result after the

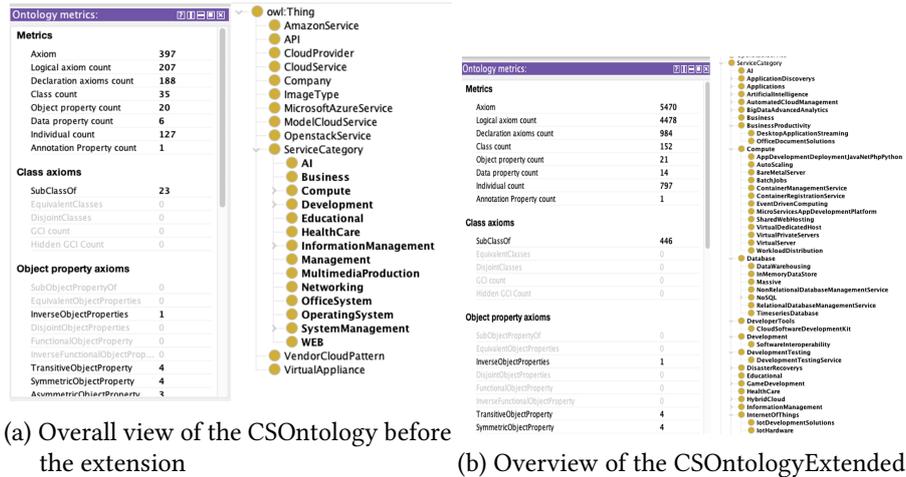


Figure 2: CSOntology and its extension

extension. In order to create the CSOntologyExtended, new Providers, Service Categories and Service Instances have been taken in consideration. In particular, the dataset provided by the Public Project **Cloud Comparer**¹ has been used a source to extend the Cloud Categorization and introduce new Services and relative parameters.

Cloud Comparer aims at describing a series of Cloud and Web Services, divided into Providers and Categories, so that similar Services offered by different Providers can be compared. The dataset can be downloaded as a JSON file, structured as shown in Listing 1.

Listing 1: Excerpt of the JSON dataset provided by Cloud Comparer

```

"category": { "name": "Compute", "ref": "" },
"service": { "name": "Virtual Server", "ref": "",
  "Properties": [
    "Number of instance templates available", "GPU acceleration", "Custom instance
      creation feature",
    "CPU Limits", "Memory Limits", "Temporary Storage Limits"
  ] },
"aws": [ { "name": "Amazon EC2",
  "ref": "https://aws.amazon.com/ec2/",
  "icon": "Compute_AmazonEC2.png",
  "Properties": ["39", "Yes", "No", "1 - 40", "0,5 - 244 GB", "Up to 48 TB (
    Multiple Disks)"} ] ]

```

This part of the dataset shows the Services belonging to the **Compute** category, of type **BareMetal**, and compares different alternatives. As an instance, Amazon Web Services (AWS) offers the Amazon EC2 Bare Metal Instance, which has some specific declared characteristics that can be compared with other offers from different providers. The information derived from the Cloud Compare project have been used to extend and populate

¹<http://comparecloud.in/>

the CSOntology. In particular, Python scripts exploiting the **owlready2**² library have been created.

In order to correctly populate the ontology, new classes have been defined as subclass of the **ServiceCategory** class. The new classes comprehend categories listed in the Cloud Compare dataset, such as *Compute*, *Mobile Services*, *Artificial Intelligence*. Individuals have been created considering the actual Services described by the dataset, such as the Azure Bare Metal Serve described in listing 2.

Listing 2: Excerpt of the JSON dataset provided by Cloud Compare defining a Service

```
"azure": [{ "name": "Azure Bare Metal Servers (Large Instance Only for SAP Hana)",
  "ref": "https://docs.microsoft.com/en-us/azure/virtual-machines/workloads/sap/hana-overview-architecture",
  "icon": "Azure Virtual Machine.png",
  "Properties": [
    "39",
    "Yes",
    "No",
    "1 - 40",
    "0,5 - 244 GB",
    "Up to 48 TB (Multiple Disks)" ]}]
```

New instances have been used to populate the ontology, such as *Shared Web Hosting*, *Virtual Server*, *Auto Scaling* and *Batch Jobs*, considered as instances of *Compute*.

A Service description, as provided by Cloud Comparer, contains several properties. These have been translated into DatatypeProperties, having the specific category class as a domain. Indeed, each category has its own properties, which have been dynamically created by analysing the JSON. Figure 3 shows class *AutomatedDisasterRecoveryServices*, its specific properties and instances. *AutomatedDisasterRecoveryServices* is a subclass of *DisasterRecovery*, and it is the domain of the *Ref* (url of a reference website) and *offeredBy* (connecting the Service to its Cloud Provider) properties, and comprehends several instances, among which **azureBackup** and **azureSiteRecovery**. Among other instances, it is also possible to identify *AgnosticAutomatedDisasterRecoveryServices*. Such an individual describes an agnostic Service that represents the entire class: *ServiceCategory*'s subclasses all have one of such agnostic Service individuals, which are used to create agnostic compositions with OWL-S.

The CSOntologyExtended ontology contains a **CloudService** class, with two subclasses, **AgnosticCloudService** and **VendorSpecificCloduService**. *AgnosticCloudService* is the type to which all agnostic Services refer, while vendor specific Services are instances of *VendorSpecificCloduService*.

So, each individual is considered as an instance of two separate classes: one is a subclass of *ServiceCategory*, the other can be either *AgnosticCloudService* or *VendorSpecificCloduService*. An individual of the *AgnosticCloudService* class is considered as the domain of the **isServiceEquivalent** ObjectProperty, whose range is instead represented by *VendorSpecificCloduService*. This property expresses equivalence among Services, and is used to identify vendor specific Services corresponding to agnostic Services. An example is show in Figure 4, where the prop-

²<https://pypi.org/project/Owlready2/>

erty has been instantiated for the individual `Agnostic_ApplicationService`. Such individual belongs to two different classes, **AgnosticCloudService** and **ApplicationServices**: the former identifies `Agnostic_ApplicationService` as an agnostic Service, while the latter declares it to be an Application Service. The equivalent individuals can either be Vendor Specific Services (such as **IbmCloudFunctions** in the example) or Agnostic Services (such as **logicApps** in Figure 4).

3.2. Description of Agnostic Services

The second input of the Expert System is represented by the description of Agnostic Services, which is used to allow it to identify an optimal configuration, or more than one, of Vendor Specific Cloud Services, starting from an initial agnostic composition. OWL-S is exploited to represent such agnostic Service compositions.

In order to describe how OWL-S is exploited in this approach, we are going to describe an agnostic composition which is named **watchFilm**. This agnostic composition allows to watch films after logging in to a Remote platform, which analyses the user's behaviour through a Big Data service to enhance her experience. So, the watchFilm Service is composed of a **Login** service, and a **SelectedForYou** Service, which proposes a catalogue of films to watch, according to the past selections made by the user. SelectedForYou is in turn composed of simpler agnostic Services, **BigData&AI**, **scalableDatabase** and **agnostic_Compute**. These Services could be further broken down, but for simplicity we will stop at this level. Figure 5 shows such a composition: each ellipse represents a Service, and the arrows describe a "composed of" relationships. Let's suppose a User requested the watchFilm Service. This would in turn rely



Figure 3: Example of Properties defined for the AutomatedDisasterRecoveryServices class



Figure 4: Use of the `isServiceEquivalent` object property

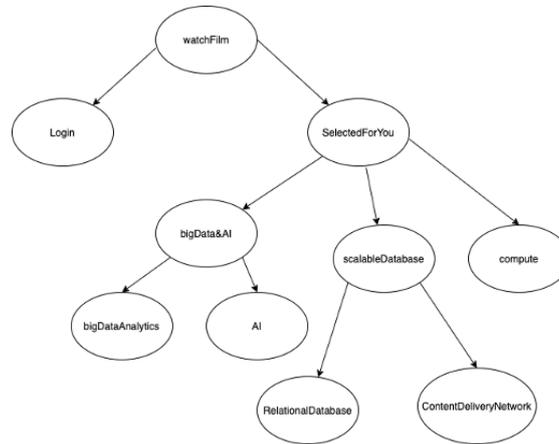


Figure 5: watchFilm agnostic Service Composition

on the Login Service to identify the User, in order to retrieve her personal information. Such information would be used by the selectedForYouService to retrieve a list of suggested films, which in turns relies on scalableDatabase as a data source for films, BigData&AI to analyse the User’s preferences, and agnostic_Compute to run the actual calculation.

This specific composition is obtained through OWL-S, using the Services defined in the CSOntologyExtended as a reference. In order to define the composition, the **SERVICE** class defined by OWL-S is populated with individuals representing the agnostic Services constituting it, as shown in Figure 6. Each of the shown individuals is connected to an instance of the **COMPOSITEPROCESS** class defined by OWL-S through the ObjectProperty **describedBy** (also defined in OWL-S). Viceversa, each **COMPOSITEPROCESS** is connected to a **SERVICE** individual through the **describes** ObjectProperty. In our specific case, **selectedForYou_COMPOSITEPROCESS** is a composed process that describes the **selectedForYou_SERVICE**. The selectedForYou_COMPOSITEPROCESS is in turn constituted by the Services listed in Figure 6, through the **hasParticipant** ObjectProperty, as shown in Figure 7.

All the Services used as participants in this composition are being defined in OWL-S as a further composition of simpler Services, but the ones that cannot be broken down are directly connected to an agnostic Service from the CSOntologyExtended ontology. This is the case for the Agnostic_IdentityAccessManagement Service, which is used as a participant ah the watchfilm_COMPOSITEPROCESS directly.

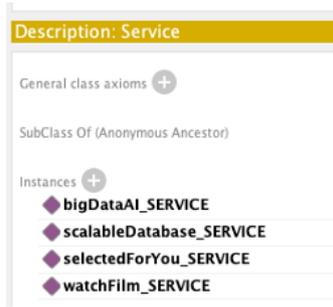


Figure 6: Use of the Service class from OWL-S



Figure 7: Use of the **describes** and **hasParticipant** properties for the selectedForYou Service

3.3. Description of Non-Functional Requirements

In order to describe Non-Functional Requirements (NFR), a new class has been defined in the ontology, which has been then populated with individuals representing several kinds of requirements, such as scalability (*nfrScalability*), security (*nfrSecurity*), use of a specific kind of Database (*nfrRelationalDB* for a Relational Database) and so on.

In order to connect a Service Category with a specific NFR, the **hasTargetService** has been created, and it has been explicitly exploited for the definition of the NFR class itself. As it is shown in Figure 8, a series of connections have been created between the **nfrScalability** individual and the Services satisfying Scalability requirements. If we take in consideration the Scalability requirement, which is satisfied when the architecture of a system is dynamic

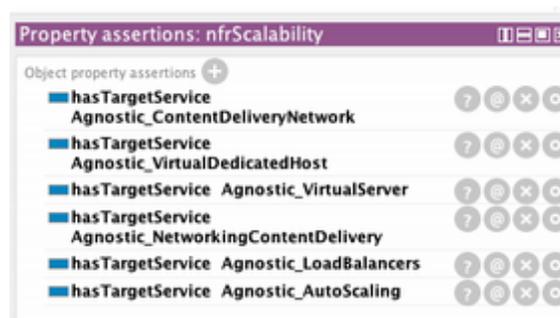


Figure 8: Use of the Scalability NFR

and additional servers can be added as needed, then a first rule that can be implemented in the knowledge base states that a Service composition will be scalable if all of its constituting Services (either simple or composite) are also scalable. The logical rules that the expert system will use in order to create the Services Compositions will take in consideration such definitions, which will in turn be represented by ad-hoc Prolog rules.

4. Definition of Logical Rules in Prolog

Once the semantic description of the Services composition is ready, it is directly transformed into a series of Prolog fact through the **Thea** software.

A set of low level rules have been defined, which represent the baseline to describe higher level relationships and complex inferences. These low level rules are divided into three categories:

- Rules to identify the Class to which a Service belongs.
- Rules to check if a Service belongs to a specific Vendor and it is this recognisable as a Vendor Specific Service.
- Rules to identify Agnostic Services.

The OWL-S structure is then transformed into a Prolog Functor, by using a recursive rule that navigates composite processes until it reaches an Atomic or Simple Process, as defined by the OWL-S standard.

In the example described in Section 3.2, watchFilm was a COMPOSITEPROCESS, constituted by the Login and selectedForYou Services, the latter being again composite. The recursive rule generating the resulting functor is provided in Listing 3.

Listing 3: Prolog Rule to create a Functor for a Service Composition

```

functorAgnosticCompositeService(S,Res,F):- classAssertion(agnosticCloudService,S), add(S,
    [],Res), F =.. [S].
functorAgnosticCompositeService(S,Res,F):-compositeProcess(S),findall(P,
    serviceParticipants(S,P),Part), functorAgnosticCompositeService(Part,Res,_),flatten(
    Res,FRes),F =.. [S|FRes].
functorAgnosticCompositeService([Sh],Res,F):-functorAgnosticCompositeService(Sh,_,F),Res
    = [F].
functorAgnosticCompositeService([Sh|St],Res,F):-functorAgnosticCompositeService(Sh,_,F1),
    functorAgnosticCompositeService(St,_,F2), Res = [F1,F2],F = [F1,F2].

```

The rule analyses each Service and its components by using the findall call, and it recursively explores all the new identified Composite Process. The recursion stops when the exploration does not provide new Composite Processes to further analyse.

Once a Service composition is retrieved, it is necessary to verify if it satisfies the NFRs expressed by the user.

First, all the Services of the composition are analysed to check the agnostic requirements introduced in the ontology as described in Section 3.3. The Prolog rule used for this is shown in Listing 4. The rule analyses a Service S and a Requisite R, to verify if it is satisfied through the hasTargetService property. Using Prolog predicates such as **setof** it would be possible to have a complete list of satisfied requirements.

Listing 4: Prolog Rule to check agnostic NFR

```
agnosticMatch(S,R):- functorAgnosticCompositeService(S,_,F), subterm(T,F),
    propertyAssertion(hasTargetService,R,T).
```

Once the agnostic composition has been checked and it satisfies the user's requirement, it is possible to choose matching Vendor Specific Services, by specifying further requirements on the characteristics of the Service. This is done by examining the DataProperties associated to the Vendor Specific Services, which have been defined in the Service ontology, as described in Section 3.2. Such properties are directly transformed by Thea in Prolog facts, which can be immediately used in logical rules.

Let's consider again the composition obtained for the watchFilm Service. In order to request the Vendor Specific Services that satisfy the user requirements, as an instance on the BigData&AI and ScalableDatabase Services, the rules shown in Listing 5 have been applied.

Listing 5: Prolog Rules to check constraints on BigData&AI and ScalableDatabase Services

```
matchBigDataConstraint(S, Functor):-
    agnosticToVendor(agnostic_BigDataQueryAsAService,R1),
    propertyAssertion(memoryLimits_GB,R1,Exp1),
    intValue(Exp1,Val),Val>300,
    functorCompositeService(agnostic_BigDataQueryAsAService,R1,F1),
    agnosticToVendor(agnostic_MachineLearning,R2),
    propertyAssertion(gpuAcceleration,R2,Exp2),
    boolValue(Exp2,true),
    functorCompositeService(agnostic_MachineLearning,R2,F2)
    -> functorCompositeService(bigDataAI_COMPOSITEPROCESS,[F1,F2],Functor),
    S = bigDataAI_COMPOSITEPROCESS.
```

The BigData&AI service is required to have at least 300GB of available RAM, and a dedicated GPU for Machine Learning. Through the **agnosticToVendor** rule, the Prolog engine can identify Vendor Specific Services corresponding to an Agnostic one, provided that they satisfy a set of specified requirements. In the example reported in Listing 5, **agnosticToVendor** is first used on the **agnostic_BigDataQueryAsAService** Service, which is an agnostic Service providing Big Data functionalities, to identify Vendor Specific Services that satisfy the 300 GB minimum memory requirement. Then, the same rule is applied to the **agnostic_MachineLearning** individual, identifying generic Machine Learning Services, by finding Vendor Specific Services that offer Machine Learning functionalities that, at the same time, have GPU support. The identified services are composed together by the final **functorCompositeService** functor.

5. Conclusion and Future Works

In this paper, an Expert System has been presented with the capability of providing users a composition of Vendor Specific Cloud Services, starting from an agnostic definition of such a composition and a set of logical rules, used to identify Services which satisfy specific non Functional Requirements. The agnostic composition is described by combining an ontology of Cloud Services, which has been obtained by extending the already existing CSOntology

with information extracted from the Cloud Compare dataset, and the structure provided by OWL-S. Several classes and new individuals have been added to the original CSOntology, here renamed as CSOntologyExtended, by using the information available on the Cloud Comparer website, which provides an in depth comparison among Cloud Services from several Vendors. In particular, the JSON representation of the Cloud Services exposed by Cloud Comparer have been automatically extracted and used to populate the new CSOntologyExtended.

The framework leverages the Thea software to translate the Ontological knowledge base into a Prolog set of facts and assertions, which represent the base of executable logical rules. The logical rules which have been defined allow for the creation of Prolog functors, that express the composition of Services, which are specifically selected in order to provide the exact requirements requested by the user. The resulting Service compositions can satisfy multiple requirements at the same time. At the moment, the logical rule that express the NFRs chosen by the user are still quite limited and hard coded, as they are not generated automatically, but they require to be expressed either as new Prolog rules, or as SWRL rules that can be evaluated by an inferential engine. However, in the future we will study a means to dynamically create such rule, according to the User's actual requirements, by means of a graphical interface that can help her to express such requirements. Also a real system design as a service, useful also to evaluate the scalability and the usability of the tool, will be realised.

References

- [1] R. Buyya, C. Vecchiola, S. T. Selvi, *Mastering cloud computing*, Tata McGraw-Hill Education, 2013.
- [2] D. Catteddu, *Cloud Computing: benefits, risks and recommendations for information security*, Springer, 2010.
- [3] B. Di Martino, A. Esposito, S. Nacchia, S. A. Maisto, U. Breitenbücher, An ontology for oasis toasca, in: *Workshops of the International Conference on Advanced Information Networking and Applications*, Springer, 2020, pp. 709–719.
- [4] B. D. Martino, G. Cretella, A. Esposito, G. Carta, An owl ontology to support cloud portability and interoperability, *International Journal of Web and Grid Services* 11 (2015) 303–326.
- [5] B. Di Martino, D. Petcu, R. Cossu, P. Goncalves, T. Máhr, M. Loichate, Building a mosaic of clouds, in: *Euro-Par 2010 Parallel Processing Workshops*, Springer, 2011, pp. 571–578.
- [6] B. D. Martino, G. Cretella, A. Esposito, 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, volume 2, IEEE, 2013, pp. 182–187.
- [7] B. Mark, H. Jerry, L. Ora, M. Drew, M. Sheila, N. Srin, P. Massimo, P. Bijan, P. Terry, S. Evren, S. Naveen, S. Katia, *OWL-s: Semantic markup for web services*, <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/>, 2004.
- [8] V. Vassiliadis, J. Wielemaker, C. Mungall, Processing owl2 ontologies using thea: An application of logic programming., in: *OWLED*, volume 529, Citeseer, 2009.