Semantic Abstraction and Modelling of Heterogeneous Application Deployments

Zoe Vasileiou [a,*], Georgios Meditskos [b], Stefanos Vrochidis [a] and Ioannis Kompatsiaris [a]
[a] *Information Technologies Institute, Centre for Research and Technology - Hellas, Greece*
E-mails: zvasilei@iti.gr, stefanos@iti.gr, ikom@iti.gr
[b] *School of Informatics, Aristotle University of Thessaloniki, Greece*
E-mail: gmeditsk@csd.auth.gr

Abstract. As the complexity of applications increases, more and more organizations migrate to cloud and High-Performance Computing (HPC) infrastructures. However, those resources are diverse and heterogeneous with different standards per cloud provider. Thus, the authoring of the deployment topology is a daunting task making the definition of a unified and interoperable model of uppermost importance. In this paper, we present TOSCA-S, an ontology for capturing key notions of the TOSCA meta-model, following existing Semantic Web standards and best practices in ontology design. More specifically, the ontology reuses the conceptual model of the DOLCE+DnS Ultralite (DUL) foundational ontology, making use of the Descriptions and Situations (DnS) design pattern for defining an abstraction layer for the representation and mapping of applications and infrastructures to ontological entities. The core model focuses on capturing information at higher levels of abstraction, enabling the conceptual description of artefacts, services, code and platforms that fosters advanced context-aware searching, matchmaking, validation and reuse. The proposed framework is part of the SODALITE platform that aims to provide an optimised, highly resilient heterogeneous execution environment enabling operational transparency between Cloud and HPC infrastructures.

Keywords: Pattern-oriented ontology design, Semantic Interoperability, Ontologies, Cloud Computing, HPC

1. Introduction

Cloud computing [1] is a developing paradigm of distributed computing that has reached a substantial level of maturity, therefore has become the most important buzzword in industries. Especially, now-a-days, the applications are getting increasingly more complex such as AI training and HPC applications that require different kind of components and consume large amount of resources. Cloud computing eases the deployment of those applications since it provides a shared pool of resources that users can allocate on-demand according to their existing needs. The users do not need to spend time in installing software in the infrastructure, and also, to invest in expensive IT infrastructure, but only pay per use. In such a manner, cloud computing is more appealing to the users providing flexibility, scalability, better economic plan and other benefits.

This rising adoption of cloud is leading the conventional cloud homogeneous infrastructures to a heterogeneous path. Many applications are requiring different kind of resources for better scaling and performance such as hardware accelerators and distributed environments. Cloud-based HPC solutions are preferred and offered by various commercial vendors that provide multiple frameworks such as Kubernetes for container environments, and SLURM for HPC. However, the cloud APIs and interfaces offered by the cloud providers are different and have many in-

*Corresponding author. E-mail: zvasilei@iti.gr.

compatibilities. The users of HPC applications need to link diverse services, configure and deploy them by using different cloud APIs that inevitably induces interoperability issues because of the cloud APIs that are specialized per cloud provider. To mitigate those issues, there is a high demand to define a unified model for abstracting the Cloud-HPC resources.

As the National Institute of Standards and Technology (NIST) [2] stated, the Cloud computing is composed of three service models and four deployment models. The three service models are Software as a Service (SaaS), Platform as as Service (PaaS), and Infrastructure as as Service (IaaS) [3]. The four deployment models are the private cloud, community cloud, public cloud and hybrid cloud. Those deployment models are defined according to where the infrastructure for the deployment resides and by whom is controlled [4].

Each cloud service model satisfies a unique set of business requirements. The IaaS is the base layer of Cloud Computing which provides the infrastructure such as storage, and network without requiring any physical hardware onsite. PaaS offers more than infrastructure, namely middleware such as database management systems and other runtimes into the cloud environment. SaaS is the top layer of cloud computing and allows users to access application that can be accessed via a web browser. Further, there is the X as a Service (XaaS). XaaS [5] represents "anything as a service". In this paper, we focus on IaaS [6] that provides cloud-services, such as server, storage and network, where the two other layers, SaaS and PaaS, rely on. In IaaS, the DevOps user should write the deployment model for provisioning the infrastructure resources as optimally as possible which is not an easy task. Indicatively, the user should describe the set of the Virtual Machines (VMs), the Virtual Network linking the VMs, the images from which the VMs are running, and potential data stores connected to the VMs [7]. Additionally to this, the user should select the suitable infrastructure services across private, public, and hybrid clouds.

In principle, cloud customers usually are reluctant to be bound to one service provider [8], since for various reasons, such as geographic location of the cloud data centers, the performance offered by the services, or Service Level Agreements, might not totally satisfy the user. Especially, nowadays, that the applications are distributed and require a heterogeneous computing system, the users in their quest to achieve high performance, they adopt solutions with multiple cloud providers.

Henceforth, in a typical Cloud Computing deployment scenario, users should select the resources from different service providers. There are many different IaaS providers, such as Amazon Web Services(AWS), Microsoft Azure, and OpenStack, that rely on unique sets of architectures and APIs [9]. For instance, if a user selects to migrate a software resource from one provider to another, the configuration needs to be customized so as to fit the target. As such, choosing the appropriate resources and migrating from one provider to another is a cumbersome and time-consuming task, delaying the deployment plan of the user, the known vendor-lock-in problem. There are many issues in semantic cloud portability and interoperability [10] [8] in the multi-cloud environment, thus there is a converge for standardization [11]. For alleviating this complex problem, many standard Infrastructure as a Code (IaC) modelling languages have been developed such as OCCI, CIMI and TOSCA focusing on decoupling the cloud applications from the specific idiosyncrasies in the target platforms. However, using different standards, there are still interoperability issues leading to non-reusable cloud resources; thus, the definition of an abstraction layer in a formal, machine-readable format is paramount. Topology and Orchestration Specification for Cloud Applications (TOSCA) is a language released by OASIS, and has widespread acceptance. For reaping the benefits of this standard, a modelling layer can be defined through Semantic Web Technologies, based on the TOSCA language.

Semantic Web is an extension of the current Web and aims at establishing a common framework for sharing and reusing heterogeneous resources. Semantic Web technologies have become popular for enhancing interoperability on various domains, and to a great degree in cloud computing [12][13]. The advantages of using ontologies for achieving interoperability in cloud computing was well initiated in [14], one of the first endeavors to establish a thorough taxonomy for cloud computing. In cloud computing, interoperability is more indispensable than ever, thus many ontologies have been developed for representing the domain in an unambiguous manner. Additional to interoperability, these ontologies aim at alleviating different challenges in Cloud Computing such as service description, and service discovery [15].

The primary goal of the cloud ontologies is the interoperability. The moSAIC project defined an ontology [16] that describes the cloud concepts and definitions according to NIST. The PaaSport project [17] aims at addressing interoperability in the Cloud PaaS market, by following a similar ODP approach with SODALITE, and discovering PaaS offerings. Other research works [8] [18] [19], additionally to the interoperability and the functional require-

ments, represent also non-functional requirements such as the price and the response time. However, despite the growing interest in ontology-based solutions in Cloud computing, little focus has been given on providing modular and reusable solutions in order to make an extensible and standarised model. In the light of the interoperability, various research works [20][21] also focus on providing semantic frameworks that reasoning the ontologies for discovering resources and validating the deployment models.

Other modelling efforts, that do not adopt a semantic approach, are following a Model Driven Engineering approach by offering an intuitive way to the modelers in designing their models. CloudCAMP [22] is compliant with TOSCA and provides a visual environment for authoring abstract models without requiring significant domain expertise. Alien4Cloud[1] eases the design and portability of applications by leveraging TOSCA. ModaClouds [23] framework abstracts the model from the targeted cloud environment and allows quality assessment at design time such as estimating the cost for migration.

To best of our knowledge, none existing modelling framework includes a wide range of assistance support. Most of the decision support systems for IaC are offering solely cloud discovery services. However, a non-expert user needs various kinds of support in order to design a deployment model with minimal effort and less issues to arise during the deployment time. In pursuit of that objective, SODALITE is using Semantic Web Technologies for creating an abstraction layer for the cloud resources. Defining an abstracted conceptual model with those technologies, the symbolic inference capabilities of the Knowledge Base are leveraged for unveiling any hidden knowledge and yield smart suggestions. An innovative aspect of our ontology is the adoption of the meta-modeling capability of OWL2 language since it allows to capture the complex relationships of the TOSCA models, and also cloud-based HPC resources and applications. None of the existing cloud ontologies follows this meta-modelling approach for the creation of a conceptual layer.

SODALITE offers a reasoning framework that assists the users with multiple intelligent capabilities. In particular, this intelligence incorporates (i) context-aware content suggestions (ii) optimization recommendations for better performance (iii) validation for detecting inconsistencies (iv) simplification of the model by permitting omission of some model part. For the validation, the Shapes Constraint Language (SHACL) was used as it allows to decouple the class definitions from their constraints, while for the suggestions, the SPARQL query language was adopted for querying over the RDF Knowledge Graphs.

To this end, we propose TOSCA-S ontology upon which the SODALITE reasoning framework relies. TOSCA-S is used as the underlying abstraction layer to represent cloud and HPC resources, and applications in a unified manner, allowing their further extension and reuse by different designers. This work is developed under SODALITE project, for assisting application developers and infrastructure operators to deploy, optimize and execute their heterogeneous applications in a simpler and faster way. The ontology has been developed by using existing Web Standards and following the best practices in the ontology design, by the use of:

- **Ontology Design Patterns:** As described in [24], Ontology Design Patterns(ODPs) are small, modular and re-usable solutions in recurrent modeling problems. TOSCA-S ontology is based on DnS ontology design pattern[25] providing native support for modularization of domain-specific ontologies.
- **Meta-modelling:** By reaping the expressivity of OWL2, the latest W3C standard, we use meta-modelling where the same identifier is used for both an instance and a class. This practice enables the modelling of TOSCA concepts as there are entities playing multiple roles.

The paper is organized as follows: In Section 2, tha background knowledge is described. In Section 3 the related work is described. In Section 4, the scenario that motivates our research is described along with some challenges. In Section 5, the position of the reasoning framework within the SODALITE conceptual architecture is presented. In Section 6, the proposed ontology is described along with its specification and a deployment model for the snow use case, extending our domain ontology. In Section 7, various reasoning cases are presented in order to show indicatevely the intelligence that can be offered to the user authoring IaC. In Section 8, the evaluation of the TOSCA-S ontology is presented. In Section 9, we conclude our work.

[1] https://alien4cloud.github.io/

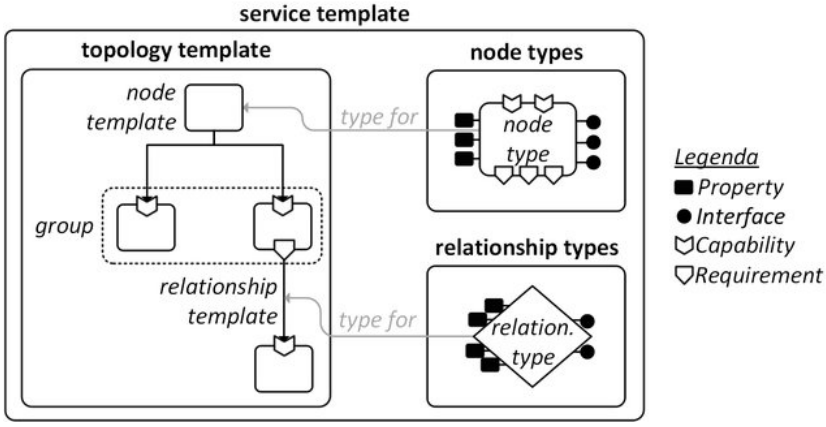


Fig. 1. The TOSCA Metamodel [28]

2. Background

2.1. OASIS TOSCA

TOSCA [26] is an emerging standard in OASIS for providing various management facilities in order to reduce the complexity in deploying composite applications. Its main objectives [27] are: a) Automated deployment and management of composite applications b) Portability of applications descriptions and their management c) Interoperability and reusability of applications components. For delivering those benefits, it provides a conceptual metamodel that allows to formalize the structure of cloud applications as a topology graph in Figure 1. The topology template specifies the overall structure of the cloud application defined by a service template. A service template is used to specify the structure of an application, namely the topology template, which is a congregation of node and relationship templates defining a cloud application. Furthermore, the TOSCA language provides a set of pre-defined normative set of types defining the schema of the resources; in particular, it provides several types such as node, relationship and capability types which are the building blocks of templates. Node types could be, for example, Compute, Database e.t.c. and Relationship types could describe how templates are connected to one another such as a host relationship. Both Node and Relationship types contain properties and interfaces. The types are reusable entities deriving from existing types and extend their properties, interfaces, attributes and other concepts. The types, in the TOSCA metamodel, follow a class-level hierarchy as one type can inherit other type. Thus, the experts can extend the normative types and define the semantics of the infrastructure resources in their own custom types.

For a better comprehension of the TOSCA metamodel, a simple logical diagram of a Web Application topology is presented in Figure 2. Three node templates are used, the `web_server`, `mysql`, and `db_server`. The Web application (`web_server`) is dependent on a database (`mysql`) which is hosted on a `db_server` server. All the templates are instances of TOSCA normative types. The `web_server` is instance of a SoftwareComponent type, has some properties such as from which image to be built, and depends on the `mysql` database as denoted in its requirements. Similarly, the `my_sql` database is instantiated from a `DBMS.MYSQL` type, has assigned properties such as the username and password for access and is hosted on a `db_server`. The `db_server` is of a Compute type, and exposes the capabilities of the node, namely hardware characteristics. The minimum required capabilities for a host are defined in the corresponding type. In our case, the required capabilities of the `mysql` host are defined in the `DBMS.MYSQL` type. For brevity, not all required relationships are included, as for example, the `web_server` is hosted in a container and this relationship is omitted. In Listing 1, the topology is depicted in the TOSCA language. Some properties are getting values from the TOSCA input parameters instead of hardcoded values.

2.2. Reused Ontology

The **Descriptive Ontology for Linguistic and Congnitive Engineering (DOLCE)** [29], developed as part of the *WonderWeb* project [30], is a foundational ontology that has a clear cognitive bias by capturing concepts related

```yaml
topology_template:
  inputs:
    #omitted for brevity

   node_templates:
    web_server:
        type: SoftwareComponent
        properties:
          image_name: web_server_image
          ports: ['8080:8080']
          docker_network_name: 'my_network'
        requirements:
           - dependency: mysql

    mysql:
         type: tosca.nodes.DBMS.MySQL
         properties:
           user: { get_input: mysql_user }
           password: { get_input: mysql_rootpw }
           port: { get_input: mysql_port }
         requirements:
           - host: db_server

    db_server:
         type: tosca.nodes.Compute
         capabilities:
           disk_size: 10 GB
           num_cpus: 1
           mem_size: 2 GB
```

Listing 1: TOSCA Topology Example for a web application

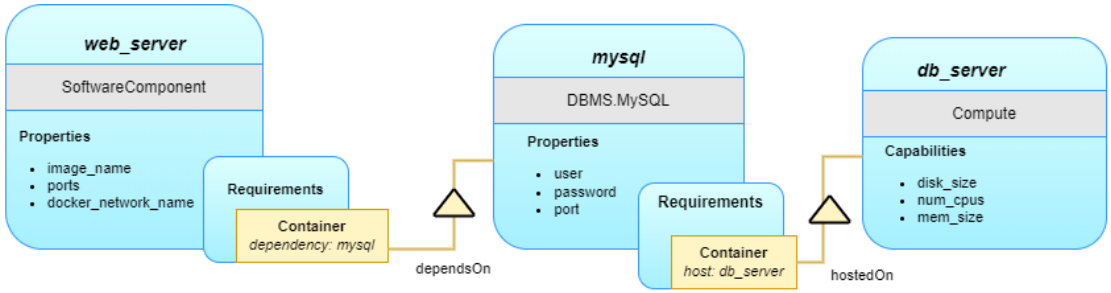


Fig. 2. Logical diagram of an excrept from a web application topology

to natural language and human common sense. DOLCE has already proven to be a good modelling basis for many core ontologies.DOLCE+ DnS Ultralite [2] is a lightweight version of DOLCE.

For promoting reusability and modularity, we used Description & Situations Ontology Pattern [25] which is a plugin to the DOLCE Ultralite (DUL) foundational ontology. Description and Situations ODP is part of DUL. It represents cognitive artifacts for being used in many rational activities such as modular conceptualization, perspective thinking and planning. The main purpose of the DUL is to provide a set of upper level concepts that can be the basis for easier interoperability among domain level ontologies. We adopted this pattern since it provides formally precise representations of different, contextualized views on concepts that can be used on all the tiers of TOSCA-S Ontology. The SODALITE ODP, as presented in Section 6.2.1, is a special instantiation of the DnS pattern.

The DnS core design pattern is illustrated in Figure 3 and allows the representation of the following conceptualisations:

– **Situation:** A situation defines a set of domain entities that participate in a specific pattern instantiation. It is interpreted through a description (*satisfies*)

[2] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl

– **Description:** A description contains the descriptive context of the situation by defining the concepts that classify the domain entities.
– **Concept and Parameter:** A concept classifies domain entities describing their interpretation in a particular situation. Each concept might have one or more parameters which allow additional nested information to be encapsulated.

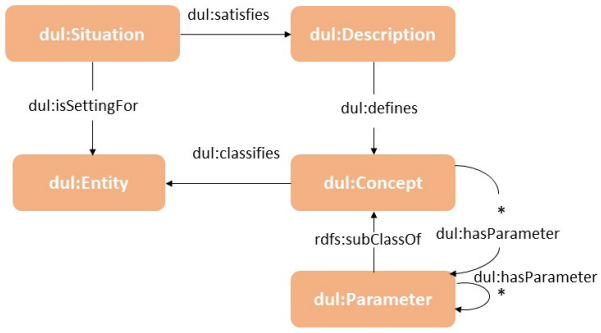


Fig. 3. Core DnS pattern in DUL

3. Related Work

Many ontologies have been developed in different areas of Cloud computing[15]. However to best of our knowledge, none of them represent the knowledge in a reusable and modular way for easier maintenance, extension, and reasoning. TOSCA supports XaaS since it is a standard that offers interoperability and permits you to define custom resource types for any cloud and HPC model definition; thus, our TOSCA-S ontology is based on this standard. Due to the nature of the deployment model in TOSCA, which has many tiers interconnected with each other through inheritance, it is important to define a unified ontology.

3.1. Cloud computing ontologies

Various ontologies have been developed modelling functional and/or non-functional requirements usually covering partially the kinds of the Cloud Computing, namely IaaS, SaaS, and PaaS.

In an ontology for OASIS TOSCA[16], the TOSCA specification has been semantically annotated so as information for the domain of the application to be modelled; however, only the structural aspects of the specification are captured and the focus is not given in representing the knowledge in a reusable and modular way since no upper level ontology is used or any other practice. Therefore, the model is not abstract for catching every aspect of an application topology and its resources. Regarding the PaaS, the PaaSport project [17] developed a cloud broker that addresses the vendor-lockin problem in Cloud PaaS market by defining an ontology for recommending the best-matching PaaS offering to the Application developer. The ontology represents capabilities of PaaS offerings, requirements of applications, and Service Level Agreements between the cloud providers and the cloud consumers. This ontology extends the DOLCE+DnS Ultralight (DUL) design pattern offering interoperability and extensibility which enable other similar efforts could map to this upper general concepts. Upon this ontology, semantic matchmaking algorithms are running for finding the best PaaS offering. In [31], the Cloud Functional and Non-functional Features (CloudFNF) ontology is presented for modelling functional and non-functional features of cloud services developed in a functional-based way. The ontology was designed based on functionalities so as to overcome issues such as overlapping concepts, unbalances granularity and poor arrangement of concepts. Nonetheless, more focus has been given on which and how the concepts are distributed rather than the interoperability. With the upper goal to implement a service discovery system, the CoCoOn [32] ontology models only IaaS concepts and primarily focuses

Table 1
Overview of Related Cloud ontologies

Name	Ref.	IaaS	PaaS	SaaS	Extensibility
Di Martino et al., 2020	[16]	+	-	-	-
Vassiliades et al., 2018	[17]	-	+	-	+
al-sayed et al., 2020	[31]	+	+	+	-
Zhang et al., 2019	[32]	+	-	-	-
Moscato et al., 2011	[19]	+	-	-	-
Castañé et al., 2020	[33]	+	-	-	-
Rekik et al., 2015	[18]	+	-	-	-
SODALITE Ontology	Our work	+	+	+	+

Legend: +(available), -(not available)

on non-functional characteristics such as price and QoS of the Cloud environments; nonetheless, the ontology only captures the structural aspects of the IaaS resources, and their QoS aspects without any generalization. The mOSAIC ontology [19] is one of the seminal cloud ontologies in the interoperability category since was designed for being compliant with various standards (e.g., OCCI, NIST). This ontology, based on OWL description logic language, captures IaaS concepts for managing heterogeneous resources within a multi-cloud environment and also represents non-functional requirements such as QoS, and SLAs. The resources are designed based on the OCCI language standard. The cloud lightning ontology (CL-ontology) [33] extends the moSAIC ontology in order to incorporate heterogeneous resources and HPC environments. It supports resource management, specific hardware accelerators, and other types of resources such as virtual machines. The Cloud description ontology [18] covers all the three cloud models, and models both the functional and non-functional properties of services. This ontology captures concepts such as the kind of the deployment model such as federated cloud environments, and characteristics of resources, and platforms.

In Table 1, the above-mentioned ontologies are compared based on the ability to model IaaS, PaaS and SaaS concepts, and also to be extensible. A comparison is done on the modelling of the cloud service models since it is essential more concepts to be modelled. The extensibility of the ontologies is also examined since it is indispensable an ontology not to be monolithic, but expandable in order to be adopted more easily and tailored to other cases. Regarding the modelling of the cloud service models, the SODALITE Ontology, by adopting TOSCA, all those models can be modelled supporting, in such way, the "anything as a service". Also, it is extensible through the use of the ODP abstraction layer promoting modularity, and of the meta-modelling enabling the reuse of the ontology semantics regarding subsumption hierarchies.

3.2. Reasoning frameworks

There are various semantic frameworks, developed in the context of cloud computing, in order to support the users in discovery of resources, matchmaking mechanisms and other context assistance methods.

A reasoning framework has been designed for federated environments, namely distributed cloud systems with multiple heterogeneous infrastructures. For those environments, the Open-Multinet (OMN) [34] set of ontologies has been defined based on an upper level ontology. Those ontologies represent the attributes and the types of the resources as well as services available within the federation. The lifecycle of the resources and services is also modeled. The semantic representation of those resources enables the discovery by the user based on the rule-based technology of SPARQL queries. The user requirements are matched with the policies set by infrastructure providers. Additional to the discovery, the OMN documents written by the users are validated against inconsistencies.

Some frameworks have focused on the interoperability on the semantic level by transforming between Cloud APIs. In order to support interoperability for IaaS cloud resources, three ontologies [35] have been defined for three cloud resource description standards, namely TOSCA, OCCI and CIMI. Those ontologies are mapped to a common upper level ontology named Linked-CR. By the use of semantic rules, the cloud resource descriptions are translated from one standard to another, in such a way, permitting the organizations to interconnect heterogeneous clouds

Name	Ref	Transformation	Discovery	Validation	Assistance
Willner et al., 2017	[34]	-	+	+	-
Yongsiriwit et al., 2016	[35]	+	+	-	-
Challita et al., 2017	[36]	+	-	+	-
Parhi et al., 2015	[20]	-	+	-	-
Ahamed et al. , 2022	[37]	-	+	-	-
Liu et al., 2014	[38]	-	+	-	-
Zhou et al., 2019	[21]	-	+	-	-
SODALITE Ontology	our work	-	+	+	+

Legend: +(available), -(not available)

using those three standards. This semantic framework paves the way for the seamless translation of cloud providers' resource description. Additionally, discovery of resources is also supported through SPARQL rules running over the knowledge base. Another framework promoting interoperability in multi-cloud is FClouds [36]. It contains a catalogue of formal models that supports transformations between the cloud APIs. The transformation is available for OCCI, TOSCA, GCP and AWS models. The FClouds models are verified based on some properties such as sequentiality and reversibility.

Many frameworks provide resource selection and discovery through the use of Semantic Technologies. In [20] a multi-agent semantic-based framework for the description and discovery of infrastructure resources is proposed. It enables the service providers in registering new cloud services and assists the users in discovering the suitable services according to their functional and non-functional requirements. The discovered resources are ranked by an algorithm based on semantic rules written in the Semantic Web Rule Language (SWRL) rules. Another multi-agent framework has been developed in [37] that exploits cloud service description annotated with cloud ontology. The user can discover Cloud IaaS services based on QoS parameters produced by a Semantic Query Processor. In [38], an ontology-based cloud service discovery is proposed that enables the user to match services based on both functional and non-functional requirements. A matchmaking algorithm is running for satisfying the user's functional goals that checks for similarities in case of not exact matching. Also, the services are selected based on non-functional requirements and ranked by preference through a weighted process. A resource discovery framework is proposed in [21] that is using an HPC resource ontology model for the cross-regional software and hardware resources distributed in a supercomputing center. The semantic model has been extended by Quick Service Query List (QSQL) representing a resource index list containing semantic relationships. Based on the QSQL structure extended by the WordNet database, a published resource is matched to the best ontology concept in index list in order to achieve fast resource discovery.

In Table 3.2, the aforementioned reasoning frameworks are compared hinged on four capabilities: (i) *Transformation:* represents the conversion from one Cloud API to another. Most of the frameworks providing this conversion support widely-used open standard languages. This capability promotes interoperability as multiple cloud APIs are supported. (ii) *Discovery:* is about matchmake and select resources according to various functional and non-functional features (iii) *Validation:* ensures the consistency of the model, written by the user, based on criteria that differ in each framework depending on its purpose (iv) *Assistance:* pertains to suggestions, provided to the user, during the design of the deployment model such as potential concepts that could be assigned to an application component. All those frameworks foster interoperability as they are using an ontology; however, to best of our knowledge, none framework offers concomitantly discovery, validation and authoring assistance for enabling simpler and easier Infrastructure as a Code (IaC).

4. Motivating scenario and challenges

In recent times, a cornucopia of tools and IaC languages have been developed for mitigating the vendor lock-in issue. Thereby, the users should put much coding effort on developing their deployment plans; especially the deployment of composite tasks, namely those targeting Cloud, HPC and Edge infrastructures, demand deep knowledge of

the platforms, usually leading to errors during the deployment. By capitalizing on the state-of-the-art techniques in the Ontology Engineering, SODALITE is an H2020 project that aspires to provide a semantic abstraction layer that enables the interoperable description of application and infrastructure services and supports the non-expert DevOps users during the design phase of their deployment models.

To exemplify, a composite deployment model could be a Machine Learning (ML) application such as the snow use case developed in SODALITE. The snow use case enables the capillary observation of the continuous health of the mountains by applying advanced image processing on large collections of images derived from multiple sources. Geo-located images are crawled automatically from the web, the images are classified based on the mountain presence, and masks are applied for identifying snow. To achieve such tasks, various processing pipelines are needed constituting by VMs, storages, and other resources. For simplification, focus will be given on an excrept of the use case. One of its components is the Mountain classifier indicating if a mountain is included in the image. For predicting this mountain presence, many components are needed with various relationships with each other. In particular, those components are: (i) a storage for saving the image training data, (ii) a service for training the model, (iii) a storage for saving the trained model, and (iv) a service running the actual prediction. As depicted in Figure 4, the two services are connected through the same docker network, and are built from docker images saved in a docker registry. Also, each service is hosted on a docker engine which accordingly is hosted on a VM.

For such deployment, the users faces several challenges such as the following:

- Host the services to the nodes that can satisfy the required capabilities. For example, an ML application might require a specific number of GPUs.
- Ensuring the relationships across the components are correctly set. For instance, a training ML service should be dependent to a database for retrieving the training dataset.
- Apply the optimal flags in the applications for better performance. For example, compiler optimizations based on the capabilities of the infrastructure.

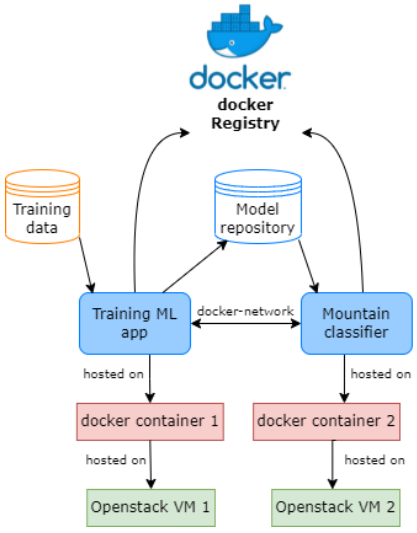


Fig. 4. Deployment model of the Snow Mountain Classifier

5. Architecture

SODALITE aims at facilitating the work of DevOps teams in designing infrastructural code that considers the peculiarities of the heterogeneous architectures by developing a *Semantic Reasoning Framework*. The position of this framework in the SODALITE conceptual architecture along with its interactions with the other components is

depicted in Figure 5. The Integrated Development Environment (IDE) [39] is the main interaction point with the user. It provides Domain Specific Language editors for the design phase of the deployment. The users are assisted during this phase by requesting assistance, and getting feedback in the form of suggestions and validation errors. This assistance provided to the user derives from the Intelligent services. The Intelligent services are based on the reasoning that is running upon the interlinked information in the *Knowledge Base* where the *TOSCA-S ontologies* [40] and the deployment models are saved in the form of RDF Knowledge Graphs. Finally, once the users are contented with their models, they can deploy them through the Orchestrator to the targeted infrastructure.

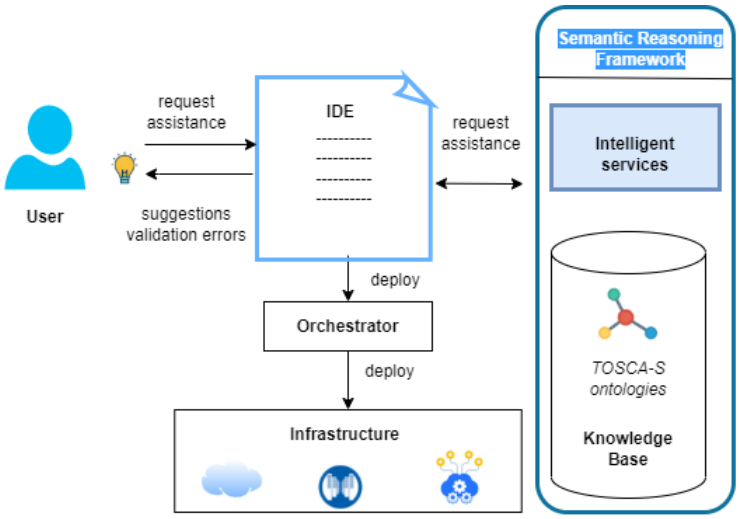


Fig. 5. Architecture of the SODALITE Reasoning framework

6. TOSCA-S Ontology

6.1. Specification

Following the NeOn methology[41], the requirements were elicited from the domain experts through three industrial use cases that the ontology network should fulfill. The requirements are expressed in the form of Competency Questions (CQs).

6.1.1. Competency Questions

Sample CQs that the data should be able to answer:

1. Which are all the available application topologies?
2. Which are the metadata of an application topology (timestamp, filename etc.)?
3. Does each application topology contain at least one template?
4. Which are the node templates that are subclasses of a specific node type?
5. Which are the properties of a node template?
6. Are all the Software Components run by a Compute Node type ?
7. Is a Software Component assigned to a host with capabilities according to its resource type definition?
8. Is a Web Application hosted on a web server ?
9. Is an application that requires Container-level virtualization technology hosted on container runtime node?
10. Is an application that requires Container-level virtualization technology connected to a storage and network with capabilities according to its resource type definition?
11. Is the corresponding input defined for each property that gets value from a TOSCA input?

12. Which are all the available resources models?
13. Which are the metadata of a resource model (timestamp, filename etc.)?
14. Does each resource model contain at least one type?
15. Which are all the operations of an interface type?

6.1.2. Formal requirements

The TOSCA-S Ontology should be able to:

1. represent an application topology including:
   (a) its metadata (e.g. timestamp, filename)
   (b) its templates (e.g. node, relationship templates) instantiated from resource types
   (c) the concepts of its templates (eg. properties, requirements)
2. represent a resource model including:
   (a) its metadata (e.g. timestamp, filename)
   (b) its types (e.g. node, relationship, artifact types)
   (c) the concepts of its types (e.g. properties, requirements)

6.2. Ontology description

TOSCA-S conceptual model, available online[3], captures and interlinks TOSCA-based descriptions of cloud resources and applications in a multi-tier manner. Both the resources and the instantiantions are captured as custom and reusable patterns. Namely, the infrastructure engineer designs the resources, while the Application engineer designs the instantiantions. This model makes the most of the existing Semantic Web technologies and the best practices in the Ontology development. Hence, the ontology has been implemented in OWL2[4], the latest official language recommended by W3C standard. TOSCA-S conceptual model takes the full benefit of the OWL2 expressiveness, for instance, by applying meta-modelling. With meta-modelling, a concept can be both an instance and a class enabling entities to play more than role which is very important for representing TOSCA concept in multi tiers. As for best practices in ontology engineering, an Ontology Design Pattern(ODP) Approach has been adopted for having modular and reusable building blocks since this modular knowledge can be maintained, and reasoned more easily. More precisely, TOSCA Metamodel contains different levels of abstractions with repeatable patterns (such as normative types, and custom -level node types). Given this kind of representation, the ODP approach is the best candidate as the amount of modelling work is eliminated for implementing common features and the reuse is promoted.

The semantic models are:

- **SODALITE Metamodel:** This is the core metamodel that promotes the modularity and reusability of the ontology elements. It inherits the Descriptions and Situations pattern. This Ontology Design Pattern (ODP) is used to capture knowledge throughout all tiers.
- **Domain ontology:** It provides the vocabulary for capturing the normative types of TOSCA, the custom resource types, and the instantiantions of resource types (templates).

`soda` is the namespace prefix of the sodalite metamodel, while `tosca` is the namespace prefix of the tosca domain ontology. For the sake of clarity, in the current paper, the prefixes are kept in a contracted form. For the expanded version, please check Table 2.

[3]https://github.com/SODALITE-EU/semantic-models
[4]https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

Table 2
Prefixes and namespaces used in the TOSCA-S ontology

prefix	ontology(namespace)
soda	https://www.sodalite.eu/ontologies/sodalite-metamodel/
tosca	https://www.sodalite.eu/ontologies/tosca/
DUL	http://www.loa-cnr.it/ontologies/DUL.owl#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
sesame	http://openrdf.org/schema/sesame#directSubClassOf
dcterms	http://purl.org/dc/terms/

6.2.1. *SODALITE Metamodel*

The SODALITE Metamodel extends the core DnS ontology pattern that described in the background section. A `SodaliteSituation` represents a type or a template. Each `SodaliteSituation` has a descriptive context(`:hasContext`), the `SodalitedDescription`, which describes the properties, capabilities, interfaces e.t.c. of the situation. The description specifies(`:specification`) one or more concepts(`SodaliteConcept`), such as properties and requirements. Each `SodaliteConcept` can have one or more parameters(`SodaliteParameter`). Each `SodaliteParameter` incorporates nested knowledge as it can contain one or more parameters.

The capabilities of the SODALITE ODP are shown through an example in Figure 7. It demonstrates the instantiation of the pattern for capturing the definition of a TOSCA base type. The correspondence between the TOSCA elements with the ODP concepts is illustrated.

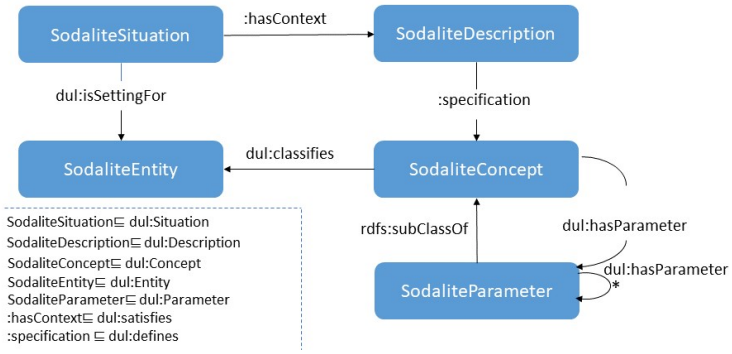


Fig. 6. SODALITE Ontology Design Pattern

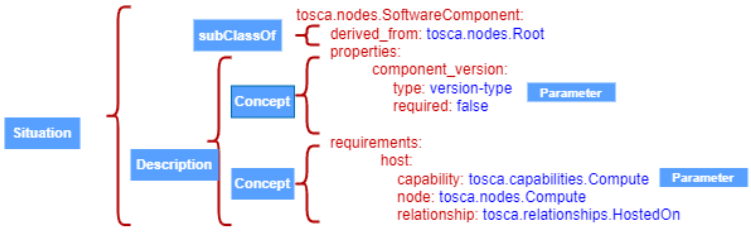


Fig. 7. Example TOSCA node type and high-level assignment of SODALITE ODP concepts

6.2.2. Domain Ontology

The domain ontology provides the necessary vocabulary so as the context to be captured in the application domain. This ontology provides the definitions of the TOSCA Metamodel. Namely, the base capability, data, node, interface, and relationship types have been defined. All those definitions have been captured through the SODALITE ODP as described in the previous subsection.

To exemplify, this TOSCA domain ontology includes the definition of the base types, additional vocabulary of TOSCA, and metadata information for modeling the topology:

- **Base types** All the TOSCA base types are modeled, namely artifact, capability, data, node, relationship, and policy types. All those types follow a hierarchy, as for instance, all the types are subclasses of the `tosca.entity.Root` type.
- **TOSCA vocabulary** Various properties pertinent to modelling the context of the types and templates is included like attributes, capabilities, interfaces, operatiosn and requirements.
- **Topology information** Descriptive information of the application topology is modelled such as the date time created, which templates it contains.

6.2.3. Tiers

The SODALITE ontology models software applications, cloud, HPC and Edge infrastructures, performance optimization and the deployment and lifecycle of the applications. All this knowledge is captured in three tiers through the ODP of the SODALITE Metamodel

- **Tier 0:** This tier contains the domain ontology including the definitions of the TOSCA Metamodel. This is the static schema of the ontology. More precisely, mainly the TOSCA normative types constitute the vocabulary, that is used in the rest tiers, including various kind of base types such as node, relationship, capability, and other types. Those types could be, for instance, a 'Compute' or a 'Network' type.
- **Tier 1:** This tier includes all the custom types designed by the Infrastructure Experts. Those custom types are non-normative and extend the existing base types of Tier 0. As in Tier 0, the custom resource definitions contain its semantics such as which properties are required, the permitted types that can serve as a host etc.
- **Tier 2:** This tier includes all the instances of Tier 0 or Tier 1 types, named as "Templates". Those instances are created by Application Ops Experts. Those experts design the application components, namely templates, and its relationships with each other which is the Topology as described in Section 2.

6.2.4. Metamodeling

Meta-modelling [42] is the practice of using a model to describe another model as an instance. One feature of metamodeling is that properties can be assigned to classes. An inspiration for metamodeling is that a model usually plays more than one role in an application. The same identifier can be a class in one role and instance in another, this is also called as punning[43]. By espousing meta-modelling, contextualised views can be represented on complex situations, creating reusable pieces of knowledge that otherwise cannot be expressed by standard ontology semantics. Other goal of the metamodeling is to mimic other modeling systems' behavior, such as object oriented programming and a property value to be set in all the members of a class.

In our case, TOSCA metamodel contains subsumption hierarchies as one node type can inherit another type. The description information of the node types can be captured only if the types can be treated as instances as well. For propagating the description of a node type to its inheritee node type, namely the descriptive context of the domain for the `:hasContext` property, depicted in Figure 8, this can be achieved through a property chain axiom of OWL2 through the `rdfs:subClassOf` relation. This chain property axiom is assigned to the `:hasInferredContext` property that will be included in the reasoning cases of Section 7.

For instance, an indicative example of the use of metamodeling in the TOSCA-S ontology is illustrated in Figure 8. Both the `tosca.nodes.SoftwareComponent` and `sodalite.nodes.DockerHost` node types play the role of an instance since they are associated with a property assertion, `properties` and `requirements` respectively. The `sodalite.nodes.DockerHost` is subclass of the `tosca.nodes.SoftwareComponent`, so it has also the role of a class, and its description contains both the requirements and the properties since the properties are inherited by the `tosca.nodes.SoftwareComponent` description.

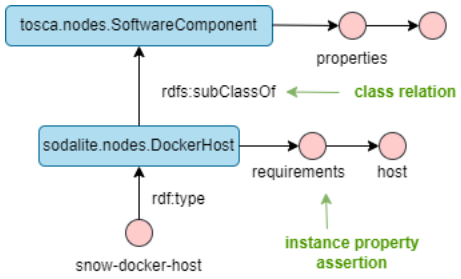


Fig. 8. Metamodeling example

6.3. Instantiating an Application using the TOSCA-S ontology

In order to have a better comprehension of the ontology design pattern, and the division through the three tiers, in the current section, it is described the developed ontology by instantiating the snow industrial use case [5], as described in Section 4. In Figure 9, it is an excerpt of the logical diagram of the snow use case topology. In this figure, a `skyline_extractor` component is a ML software application extracting skylines from images, an instance of the `sodalite.nodes.DockerizedComponent` custom type. This node template overrides some properties of its type, and has as a requirement to be hosted on the `snow-docker-host` template. Accordingly, `snow-docker-host` is a docker engine, an instance of `sodalite.nodes.DockerHost` custom type, and has as a requirement to be hosted on a `snow-vm`. The `snow-vm` is an *OpenStack* virtual machine (vm), and the `security-rules-snow` is the security configuration protecting the vm. A similar flow with the `skyline_extractor` is followed for `snow-vm` and `security-rules-snow` node templates.

In Figure 10, an ontology that corresponds to this logical diagram is depicted. As mentioned in Section 6.2, the abstraction layer for the descriptions is based on an ODP that governs all the tiers.

Analysis of Figure 10:

- **Tier 2**
  All the four node templates from the logical diagram are instantiated. As an example, we will elaborate on the `skyline_extractor` application. This application, an instance of the `sodalite.nodes.DockerizedComponent` type, is a `soda:SodaliteSituation` that has a description context (`soda:hasContext`). The description (`soda:SodaliteDescription`) is related with two properties and one requirement through `tosca:properties` $\sqsubseteq$ `dul:defines` and `tosca:requirements` $\sqsubseteq$ `dul:defines` assertions accordingly. Regarding property instances, as an example, there is a property that classifies (`dul:classifies`) 'alias' in the figure, and has a datatype property assertion (`tosca:hasDataValue`) with a 'snow skyline extractor' value.
- **Tier 1**
  All the custom node types definitions, from which the node templates are instantiated, are modelled in tier 1. As in tier 2, each node type is a `soda:SodaliteSituation` that has a description context (`soda:hasContext`). In our example, we see three additional concepts that are modelled compared to Tier 2, namely attributes, capabilities, and interfaces. Since the node types have more nested descriptions, `dul:hasParameter` enables those more complex descriptions. The node types are subclasses of the normative tosca types in tier 0. The use of meta-modelling can be showed through this example, as we see that a node type can be both a class and an instance. For example, `sodalite.nodes.DockerizedComponent` is a class, since derives from `tosca.nodes.SoftwareComponent`, while is also an instance by having descriptive context through property assertion.

[5] https://www.sodalite.eu/water-availability-prediction-mountains-images

– **Tier 0**
All the TOSCA normative type definitions are modelled in this tier. All those types inherit from the `tosca.entity.Root` class. For example, `tosca.nodes.Root` represents all the node types, and the `tosca.capabilities.Root` represent all the capability types. Similarly with Tier 1, punning capabilities of OWL 2 can be noted, as all those classes have descriptive context, but for brevity, it is omitted in this figure.

At the bottom of Figure 10, it is the legend diagram depicting the instance types of all the SODALITE ODP parts.

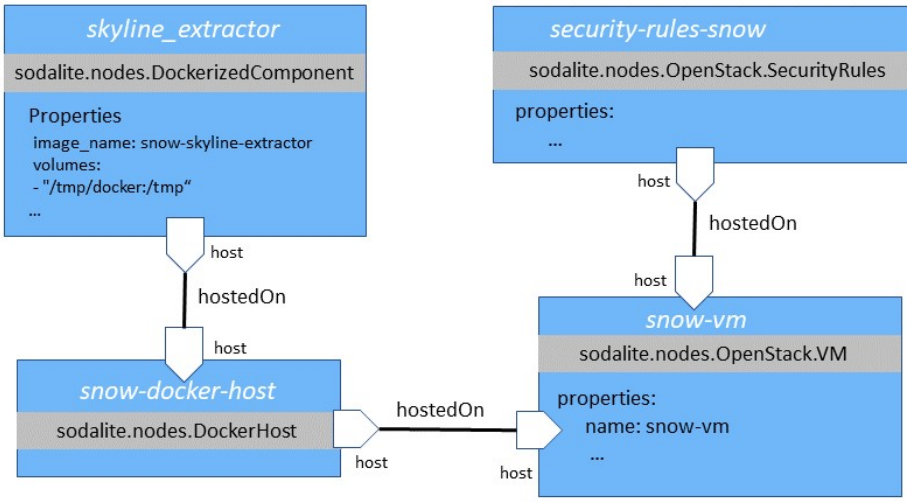


Fig. 9. Excerpt of the TOSCA representation of snow application

7. Rule-based reasoning for simpler and smarter IaC

In the current section, the inference capabilities of the TOSCA-S Ontology are presented. Indicatively, three kinds of reasoning are included covering the most prominent aspects that bestow on the increase of the simplicity of modelling applications and infrastructures: (i) validation (ii) suggestion (iii) abstraction. The reasoner is available online[6].

7.1. Topology Validation

A TOSCA application topology contains nodes that are linked with each other via relationships. Each relationship specifies that a requirement of the source node must be actually satisfied by (a capability of) the target node. For validating an application topology [44], all the elements forming the relationship should be checked, namely its source (requirement of a node), the relationship itself (a relationship template), its target (a node or a capability of a node). The scope of this paper is not to present all the conditions validating the topology, but show an example including two conditions for the validation of the requirements. Given that r_a is a requirement assigned to a node template, and r_d is the requirement definition in its node type:

(a) **name(r_a) = name(r_d)**
For each requirement (r_a) assigned to a node template, there should be a corresponding requirement definition (r_d) in its node type. This is the base condition that should be satisfied before proceeding

[6]https://github.com/SODALITE-EU/semantic-reasoner

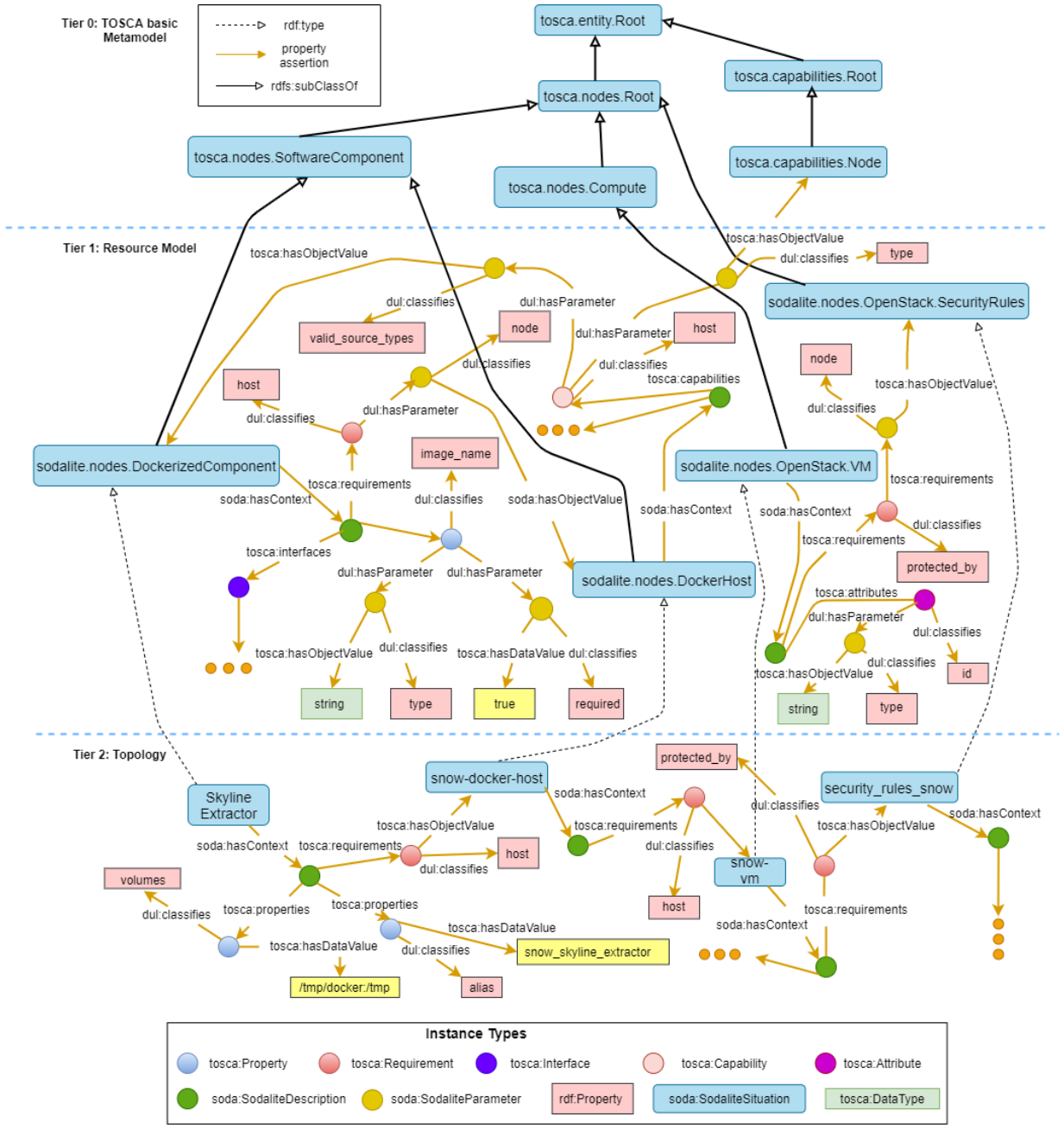


Fig. 10. Excerpt ontology from the snow use case

with other checks in the application topology. In Figure 11, it is depicted a node template, named as `snow-skyline-extractor`, with a name of a requirement assignment (`host2`) not included in its node type hierarchy. The `sodalite.nodes.DockerizedComponent` node type has only *host* as requirement name, and none of its ancestors definitions includes the *host2* requirement name. A SPARQL query in Listing 2, performs this check for the requirement assignment (r_a).

(b) **type(r_a.node) $\geqslant$ type(r_d.node)**
Given that *node* is specified in r_d, the type of the template specified in the corresponding r_a must be equal or extend the node type defined in r_d. The SPARQL query of Listing 3 performs this validation. Line 4 retrieves the node type of the template, lines 6-9 retrieve the r_d.node requirement definition, and lines 11-13 check if the type of r_a inherits the r_d.node.

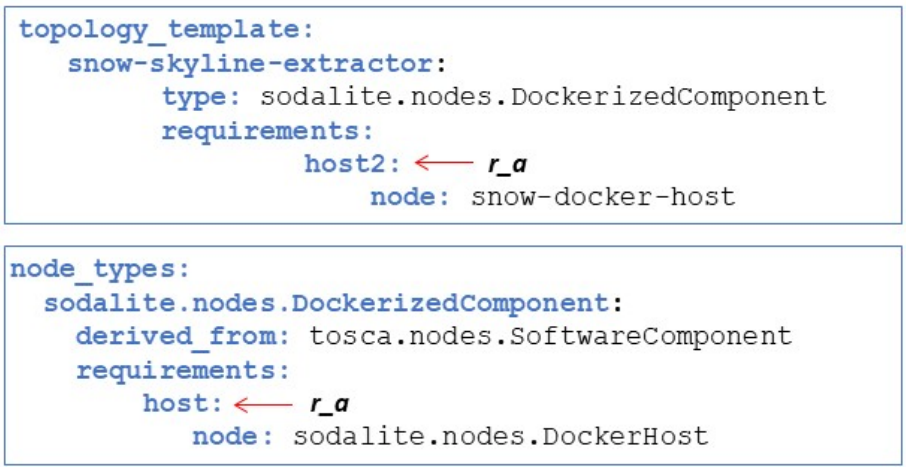


Fig. 11. Invalid requirement assignment name

```
1  select  ?template ?templateType ?r_a
2  where {
3      ?aadm :includesTemplate ?template .
4
5      ?template a soda:SodaliteSituation ;
6              sesame:directType ?templateType ;
7              soda:hasContext [tosca:requirements
8                               [DUL:classifies ?r_a]] .
9
10     ?templateType soda:hasInferredContext ?ctx;
11             rdfs:subClassOf tosca:tosca.entity.Root.
12     ?ctx tosca:requirements [DUL:classifies ?r_a] .
13 }
```

Listing 2: Query checking **(a)** condition of topology validation

7.2. *Optimization suggestions*

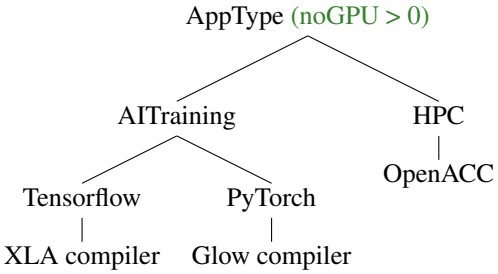
The performance of an application that is deployed using IaC on heterogeneous infrastructure target is uppermost. Cloud and High Performance Computing applications are executed on an architecture consisting of diverse execution platforms such as CPUs, GPUs, and FPGAs with different memory hierarchy making it complex for the application developer to fully exploit the acceleration potential of the architecture. The capabilities of the infrastructure such as number of GPUs, SSD availability, type of CPU architecture are already included in the application model. Through ontological reasoning over this model, by using SPARQL queries, those capabilities are retrieved for helping the application expert select the optimal flags/settings for the application.

```
select ?template  ?v  ?r_a
where {
        ?aadm soda:includesTemplate ?template .
        FILTER (contains(str(?aadm), ?aadmId)) .

        ?template a soda:SodaliteSituation ;
            sesame:directType ?templateType .
        ?templateType soda:hasInferredContext ?ctx.
        ?templateType rdfs:subClassOf tosca:tosca.entity.Root.
        ?ctx tosca:requirements ?r .
        ?r DUL:classifies ?r_a.
        ?r DUL:hasRegion ?v .
        FILTER NOT EXISTS {
                ?template soda:hasContext
               [tosca:requirements [DUL:classifies ?r_a]] .
        }
}
```

Listing 3: Query checking **(b)** condition of topology validation

According to the capability values, specific optimizations can be enabled, for example:



As a proof of concept, an example will be presented about an AI Training application that uses the Tensorflow framework. In Figure 12, the application deployment model of this application is depicted in TOSCA. `skyline extractor` node template is hosted on a `vm` node template that offers specific capabilities. Since number of GPUs is positive, the XLA optimizing compiler for Machine Learning can be enabled and also the Extract, Transform, Load (ETL) pipeline can be optimised by applying data prefetching and caching. The number of GPUs can be retrieved through a SPARQL query, depicted in 13. Lines 3-5 retrieves the capabilities of the host. Lines 11-15 retrieve the value of the property containing the number of the gpus.

7.3. Abstracted model

By omitting information in the deployment modeling, the model can be further simplified. In this section, it will be presented how a requirement assignment (r_a) can be excluded from the application deployment model.

Within the requirement section, a target node template is added under *host* keyword indicating on which container the application is going to be installed. Abovementioned, an implicit *hostedOn* relationship exists between the nodes. In Figure 14, an excerpt of the resource, and the application model from the snow use case are depicted. The `skyline-extractor` node template has the requirements section greyed out implying that the host target has been omitted.

By applying reasoning, missing requirements in the application deployment model can be detected, the compatible node type in r_d is retrieved, and finally the available matching node template is added to the model. This can be implemented in three steps:

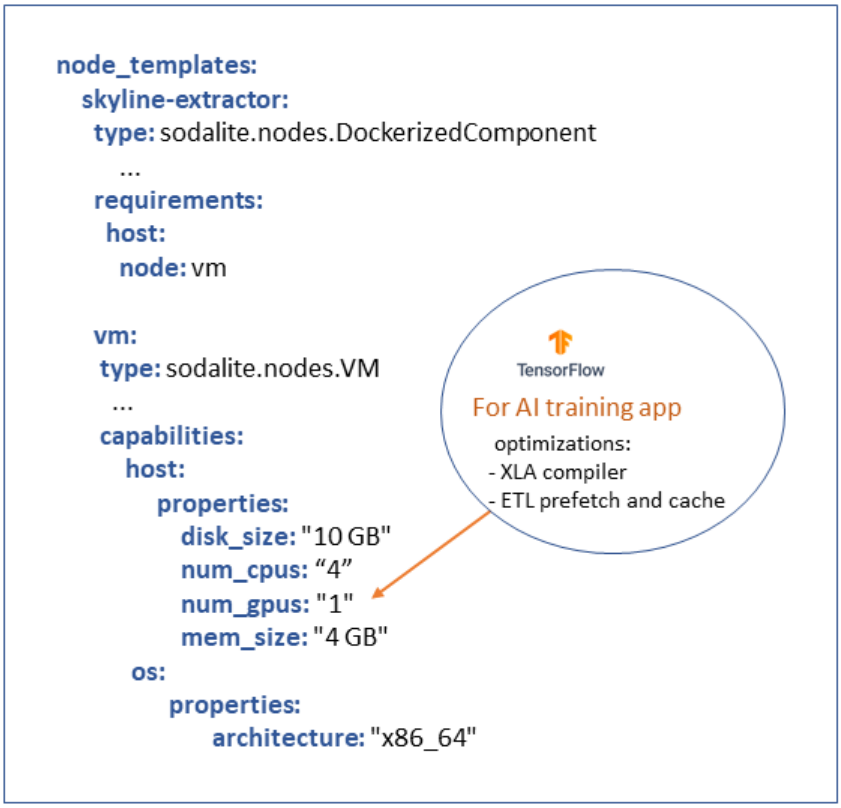


Fig. 12. AI Training application deployment model in TOSCA

```
select ?ngpu
where {
  ?template
  (soda:hasContext/tosca:requirements/tosca:hasObjectValue)+/
  soda:hasContext/tosca:capabilities  ?capability .

  ?template sesame:directType ?nodeType .
  ?nodeType rdfs:subClassOf tosca:tosca.entity.Root .

 #Retrieve number of gpus
 ?capability DUL:classifies tosca:host .
 ?capability tosca:properties ?property .
 ?property DUL:classifies ?property_gpus .
 FILTER (strends(str(?property_gpus), "num_gpus")) .
 ?property tosca:hasDataValue ?ngpu .
}
```

Fig. 13. Part of SPARQL Query retrieving number of gpus

1. **Detect missing requirements** As a first step, it should be detected if the user has not completed a requirement. In Listing 4, lines 3-4 restrict the results for the specific application deployment model. Lines 13-15 restrict the results for the omitted requirements in the model.

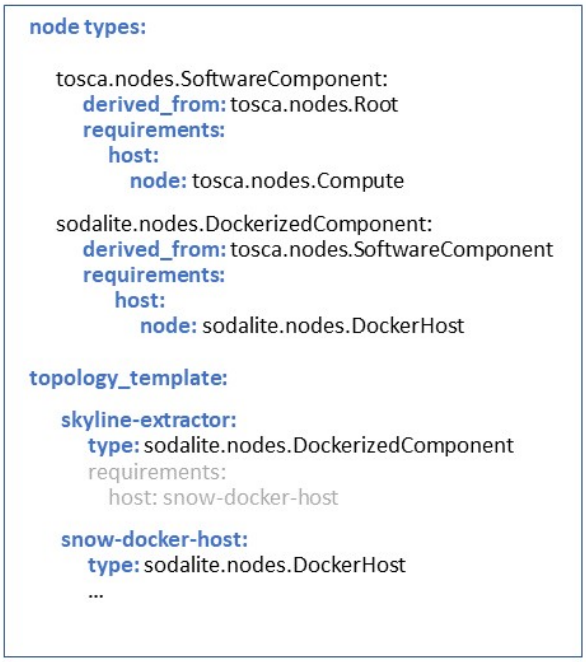


Fig. 14. Excerpt from snow use case

2. **Find the matching node type in r_d**
   All the compatible node types according to the r_d definition of the template are retrieved. As already described in 7.1 section, for each requirement (r_a) assigned to a node template, there is a requirement definition (r_d) in its node type. Given that the same requirement definition can be present not only in the direct node type definition but also in the superclasses, more than one node type could be returned. This is inferred through the `hasInferredContext` which is a chain property as described in Section 6.2. In Figure 14, `skyline_extractor` is of `sodalite.nodes.DockerizedComponent` type. This type has as a host requirement value, `sodalite.nodes.DockerHost`. `sodalite.nodes.DockerizedComponent` is of `tosca.nodes.SoftwareComponent` normative type which has as a host requirement value, `tosca.nodes.Compute`. Thus, according to the node type hierarchy, `sodalite.nodes.DockerHost` and `tosca.nodes.Compute` are inherited as valid node types for host r_a. `sodalite.nodes.DockerHost` is the lowest in the node type hierarchy; therefore, the templates that could serve as a host for the `skyline extractor` should be of `sodalite.nodes.DockerHost` type. In Listing 4, it is the corresponding SPARQL query, lines 6-12 retrieve the compatible node types (variable `?v`) from the requirement definition for the corresponding omitted r_a assignments from step 1.
3. **Resolve missing host requirements**
   The matching node templates of the node type, that found in the second step, are retrieved. If only one template is found, then the requirement of the model can be resolved by adding the template to the r_a of the model through an update operation. Otherwise, if more than one templates are found, then all the matching templates could be returned as suggestions. In Listing 5, it is the SPARQL query retrieving the templates of the node type(`?var`) of step 2.

```
select ?template  ?v  ?r_a
where {
        ?aadm soda:includesTemplate ?template .
        FILTER (contains(str(?aadm), ?aadmId)) .

        ?template a soda:SodaliteSituation ;
            sesame:directType ?templateType .
        ?templateType soda:hasInferredContext ?ctx.
        ?templateType rdfs:subClassOf tosca:tosca.entity.Root.
        ?ctx tosca:requirements ?r .
        ?r DUL:classifies ?r_a.
        ?r DUL:hasRegion ?v .
        FILTER NOT EXISTS {
                ?template soda:hasContext
               [tosca:requirements [DUL:classifies ?r_a]] .
        }
}
```

Listing 4: Sparql query for omitted requirements

```
select distinct ?template {
   ?template a soda:SodaliteSituation;
   rdf:type ?type .
   ?type rdfs:subClassOf ?var
}
```

Listing 5: Sparql query for matching templates

8. Evaluation

8.1. Qualitative evaluation

8.1.1. Compliance with the requirements of the use cases

To ensure that the semantics of the use cases are captured, we have defined a group of Competency Questions (CQs) based on the functional requirements that the data should be able to answer. Getting inspired by [45] where the authors demonstrate several cases for the validation of the RDF graphs by using SHACL and the benefits of adopting this language, we adopted this validation approach. Based on the approach in [46], each constraint corresponds to a unit test that contributes to validate the ontology. In general, a specific ontology should be able to answer CQs. Table 6 includes a group of indicative CQs along with their SHACL constraint translation.

8.2. Structural ontology assessment

In [47], the OOPS! (OntOlogy Pitfall Scanner!) was proposed, a diagnosis tool that detects common pitfalls in ontologies along with tips about how to repair them. OOPS! splits the pitfalls into three categories, namely critical, important, and minor. Some pitfalls were detected for the DOLCE upper level ontology; since this ontology is imported, only the pitfalls for the TOSCA-S ontology will be considered. No critical pitfall was detected for the TOSCA-S ontology. Only important and minor cases were detected. Most pitfalls were about missing annotations, namely ontology elements lack annotation properties that label them, and about missing domain or range in properties.

Table 3
Advanced metrics produced by the Ontometrics tool

Base Metrics	Axioms	12166
	Class count	197
	Object property count	129
	Data property count	24
	DL expressivity	SRIN(D)
	SubClassOf axioms count	283
	SubObjectPropertyOf axioms count	70
	SubPropertyChainOf	1
Schema metrics	Attribute richness	0.14
	Inheritance richness	1.45
	Relationship richness	0.38
Knowledgebase metrics	Average population	18.12
	Class richness	0.179

For assessing the domain coverage of our ontology, it was submitted to the Ontometrics [7] [48] online platform. An extension of this tool is presented in [49] where the ontometrics is presented as Ontology Metrics as a Service, and most other tools are mainly outdated. This tool evaluates the quantitative quality metrics of an ontology. Various kind of metrics are produced by Ontometrics tool. Ontometrics groups the results in different categories; The base, schema and knowledgebase metrics were used for the evaluation. The base metrics include simple metrics showing the quantity of ontology elements such as the count of the properties, classes and other kind of concepts. Also, the kind of DL expressivity is included in the base metrics indicating which variant[8] of the Description Logic is used. The schema metrics measure the design of the ontology for indicating its richness, inheritance and other metrics of the schema design. The schema metrics included in this evaluation are: (i) the *attribute richness* is defined as the average number of attributes per class. The higher this number is, more knowledge is delivered (ii) the *inheritance richness* is the distribution of information across different levels of the ontology's inheritance tree. An ontology with high inheritance richness indicates that th abstraction level is high (iii) *relationship richness* represents the diversity of the types of the relations in the ontology. Both inheritance and non-inheritance relationships are taken into account, as an ontology that includes only inheritance relationships represents less information compared with an ontology with diverse relationships. The knowledgebase metrics is based on the instance metrics for measuring the amount of real-world knowledge represented by the ontology. This kind of metrics included in the evaluation are: (i) the *average population* that indicate the average distribution of instances across all classes. If the number is low, it might indicate that the knowledge represented is insufficient. (ii) the *class richness* shows how instances are related to classes in the ontology schema. It gives a general idea about how well the instances utilize the ontology schema.

For evaluating the TOSCA-S ontology with the Ontometrics, the Knowledge Base was populated with the snow use case deployment model. The results for the base, schema and knowledgebase metrics are presented in Table 3. Regarding, the base metrics, mainly metrics related with the properties and the inheritance were included showing the quantity of the axioms in the knowlwedge base. From the schema metrics, we can infer that the inheritance richness is high indicating that the ontology covers a wide range of concepts, and notably captures the subsumption hierarchies that govern the TOSCA metamodel. As for the *KnowledgeBase* metrics, the average population is very high since, as already mentioned, the punning feature of OWL2 was adopted, by the TOSCA-S ontology, treating the subjects with the same name both as classes and individuals. The class richness is quite low since all the class knowledge is not utilized as only the TOSCA normative types related with the snow use case were instantiated.

[7] https://ontometrics.informatik.uni-rostock.de/
[8] https://handwiki.org/wiki/Description*logic*

Table 4
Save application model response times(secs) per use case.

	#templates	Save	Save with override	Delete
Clinical	16	5.8	16	5.8
Vehicle	8	3	6	2.4
Snow	23	15	22	10

Table 5
Query response times(secs)

properties	0.28
attributes	0.25
capabilities	0.19
requirements	0.32
types	0.25
full information for a type/template	0.21
requesting valid nodes that can satisfy a requirement	0.4

8.3. Quantitative evaluation

8.3.1. Use Cases

The ontology has been evaluated through three industrial use cases[9] of our European SODALITE project. University of Stuttgart, Politecnico di Milano and Adaptant company have developed the Clinical, Snow and Vehicle use cases respectively. Clinical use case is a typically HPC application simulating spinal operations which supports in-silico clinical trials of bone-implant systems in Neurosurgery, Orthopedics and Osteosynthesis. Snow use case, the application used as a main example in this paper, is a GPU application for predicting water availability by applying advances image processing on snow mountain images derived by multiple sources. Vehicle use case is an application, targeting Cloud and Edge environments, that monitors the vehicular data , and according to the values adapts the services dynamically.

The TOSCA-S knowledge graphs for supporting the three use cases consists of 165K triples in Knowledge Base. For providing intelligent assistance in the authoring of resources and applications, it is important to provide scalable reasoning for better user experience. Knowledge bases are fast in create and read operations and slower in delete operations. Saving an application model includes various queries for locating resources and validating the model such as cases in section 7. In Table 5, the response times for saving and deleting the application model per use case are depicted. As the number of templates increases, more time is needed. When a model is resaved in the Knowledge Base, all its templates are overriden, and delete operations are included, thus 'Save with override' column depicts higher response times.

In Table 4, the response times for some query types providing content-assistance are provided. The performance of those services has been positively evaluated by the end users. By conducting an experiment with 4 participants using the SODALITE IDE, we get as a feedback that the average time needed for developing the deployment model was 18.75 minutes, and for redeploying and reconfiguring the application was 37.5 minutes. Those query types are for getting:

1. the properties (from a template or type)
2. the attributes (from a template or type)
3. the capabilities (from a template or type)
4. the requirements (from a template or type)
5. all the types (data, node, relationship etc.)
6. full description from a type/template, namely all its associated concepts are returned
7. node templates that are satisfying the requirement definition of a type in order to find valid hosts, networks, storages etc.

[9] https://www.sodalite.eu/use_cases

Table 6
Indicative CQs and SHACL translation

Competency question	SHACL constraint

Does each application topology contain at least one template?	ex:AADMShape
	a sh:NodeShape ;
	sh:severity sh:fatalError ;
	sh:targetClass soda:ApplicationDeployment ;
	sh:sparql [
	sh:message "AADM contains no template" ;
	sh:select """
	SELECT distinct $this {
	$this a soda:ApplicationDeployment .
	FILTER NOT EXISTS {
	$this DUL:isSettingFor ?template.
	}
	} """;
	] .

Does each Application Deployment Model contain the required metadata?	ex:AADMMetadataShape
	a sh:NodeShape ;
	sh:severity sh:fatalError ;
	sh:targetClass soda:ApplicationDeployment ;
	sh:sparql [
	sh:message "AADM does not contain metadata";
	sh:select """
	SELECT distinct $this ?time ?user ?name {
	$this a soda:ApplicationDeployment.
	FILTER NOT EXISTS {
	$this soda:createdAt ?time .
	$this soda:createdBy ?user .
	$this soda:hasName ?name .
	}
	} group by $this ?time ?user ?name""";
	] .

Are all the Software Components run by a Compute node type?	ex:SoftwareComponentShape
	a sh:NodeShape ;
	sh:severity sh:fatalError ;
	sh:targetClass soda:SodaliteSituation ;
	sh:sparql [
	sh:message "A software component should always be hosted on a compute node" ;
	sh:select """select distinct ?host {
	FILTER NOT EXISTS {
	$this a soda:SodaliteSituation ;
	a tosca:tosca.nodes.SoftwareComponent;
	(soda:hasContext/tosca:requirements
	/tosca:hasObjectValue)* ?host .
	?host a tosca:tosca.nodes.Compute.
	}
	}""";
	]

9. Conclusions

This paper presented TOSCA-S ontology for modeling deployment plans of TOSCA-compliant languages in an abstract and inter-operable way. The creation of the ontology was motivated by the lack of a standard language in cloud computing leading to the vendor-lockin problem. Interoperability is accomplished through modular and reusable ontological components by adopting best practices in ontological engineering. Namely, the knowledge is captured in a uniform way since our ontology is based on the DnS pattern, a foundational ontology, and, at the same time, by using meta-modelling we have contextualized descriptions promoting reusability. Also, we present how this modeling approach enables the knowledge inference though reasoning, providing, hereby, assistance in the design of the deployment models through suggestions, validation support and abstraction. Regarding the assessment of the ontology, both qualitative and quantitative evaluation have been carried out. Regarding the quantitative evaluation, it is worth to be noted that it has been measured through three industrial use cases developed under the SODALITE project. The results showed that the framework can be used in real-world use cases.

Currently, we support most of the TOSCA specification. In future, we plan a) to test our ontology on the EGI infrastructure[10] b) enrich our reasoning services for supporting runtime optimizations c) to support more validation cases.

10. Acknowledgements

This study was supported by the EC funded project SODALITE (H2020-825480).

[10]https://www.egi.eu/

References

[1] I. Odun-Ayo, M. Ananya, F. Agono, and R. Goddy-Worlu. Cloud computing architecture: A critical analysis. pages 1–7, 07 2018.

[2] P. M. Mell and T. Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, USA, 2011.

[3] Naren.J, S.K. Sowmya, and P. Deepika. Layers of cloud – iaas, paas and saas: A survey. *International Journal of Computer Science and Information Technology*, Vol. 5 (3):4477 – 4480, 06 2014.

[4] D. Rountree and I. Castrillo. Chapter 3 - cloud deployment models. In *The Basics of Cloud Computing*, pages 35–47. Syngress, Boston, 2014.

[5] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu. Everything as a service (xaas) on the cloud: Origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628, 2015.

[6] S. Bhardwaj, Jain L., and S. Jain. Cloud computing: A study of infrastructure as a service (iaas). *Journal of Engineering and Information*, 2:60–63, 2010.

[7] P. Harsh, F. Dudouet, R. Cascella, Yvon Jegou, and C. Morin. Using open standards for interoperability - issues, solutions, and challenges facing cloud computing. 07 2012.

[8] Z. Zhang, C. Wu, and D. W.L. Cheung. A survey on cloud interoperability: Taxonomies, standards, and practice. *SIGMETRICS Perform. Eval. Rev.*, 40(4), 2013.

[9] K. T. Tran. *Efficient complex service deployment in cloud infrastructure*. PhD thesis, Networking and Internet Architecture [cs.NI]. Université d'Evry-Val d'Essonne, 2013.

[10] C. Ramalingam and P. Mohan. Addressing semantics standards for cloud portability and interoperability in multi cloud environment. *Symmetry*, 13(2), 2021.

[11] S. Ghazouani and Y. Slimani. A survey on cloud service description. *Journal of Network and Computer Applications*, 91:61–74, 2017.

[12] H. Brabra, A. Mtibaa, L. Sliman, W. Gaaloul, and F. Gargouri. Semantic web technologies in cloud computing: A systematic literature review. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 744–751, 2016.

[13] F. Imam. Application of ontologies in cloud computing: The state-of-the-art. 2016.

[14] L. Youseff, M. Butrico, and D. Da Silva. Toward a unified ontology of cloud computing. In *2008 Grid Computing Environments Workshop*, pages 1–10, 2008.

[15] J. Agbaegbu, O. T. Arogundade, S. Misra, and R. Damaševičius. Ontologies in cloud computing; review and future directions. *Future Internet*, 13(12), 2021.

[16] B. Di Martino, A. Esposito, S. Nacchia, S. Maisto, and Uwe Breitenbücher. An ontology for oasis tosca. In *Web, Artificial Intelligence and Network Applications*, pages 709–719. Springer International Publishing, 2020.

[17] N. Bassiliades, M. Symeonidis, P. Gouvas, E. Kontopoulos, G. Meditskos, and I.P. Vlahavas. Paasport semantic model: An ontology for a platform-as-a-service semantically interoperable marketplace. In *Data Knowl. Eng.*, pages 81–115. Springer International Publishing, 2018.

[18] M. Rekik, K. Boukadi, and H. Ben-Abdallah. Cloud description ontology for service discovery and selection. In *2015 10th International Joint Conference on Software Technologies (ICSOFT)*, volume 1, pages 1–11, 2015.

[19] F. Moscato, R. Aversa, B. Di Martino, T. Fortiş, and V. I. Munteanu. An analysis of mosaic ontology for cloud resources annotation. *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 973–980, 2011.

[20] M. Parhi, Binod Kumar Pattanayak, and Manas Ranjan Patra. A multi-agent-based framework for cloud service description and discovery using ontology. In Lakhmi C. Jain, Srikanta Patnaik, and Nikhil Ichalkaranje, editors, *Intelligent Computing, Communication and Devices*, pages 337–348, New Delhi, 2015. Springer India.

[21] A. Zhou, K. Ren, W. Li, X.and Zhang, and X. Ren. Building quick resource index list using wordnet and high-performance computing resource ontology towards efficient resource discovery. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 885–892, 2019.

[22] A. Bhattacharjee, Y. D. Barve, A. S. Gokhale, and T. Kuroda. Cloudcamp: Automating cloud services deployment management. 2019.

[23] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C. Nechifor, D. Petcu, A. Gericke, and C. Sheridan. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, pages 50–56, 2012.

[24] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7:1–7, 2015.

[25] A. Gangemi and P. Mika. Understanding the semantic web through descriptions and situations. In *CoopIS/DOA/ODBASE*, 2003.

[26] OASIS. Topology and Orchestration Specification for Cloud Applications. https://www.oasis-open.org/committees/tosca.

[27] A. Brogi, J. Soldani, and P. Wang. Tosca in a nutshell: Promises and perspectives. In *ESOCC*, 2014.

[28] OASIS. Topology and Orchestration Specification for Cloud Applications (TOSCA) Simple Profile in YAML, Version 1.3. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html.

[29] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with dolce. *Phys. Rev. E.*, 69:026113, 2002.

[30] C. Masolo, S. Borgo, A. Gangemi, and A. Guarino, N.and Oltramari. WonderWeb deliverable D18 ontology library (final). Technical report, IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web, 2003.

[31] M. al sayed and F. Omara. Cloudfnf: An ontology structure for functional and non-functional features of cloud services. *Journal of Parallel and Distributed Computing*, 2020.

[32] Q. Zhang, A. Haller, and Q. Wang. Cocoon: Cloud computing ontology for iaas price and performance comparison. In *The Semantic Web – ISWC 2019*, pages 325–341. Springer International Publishing, 2019.
[33] G. G. Castañé, Huanhuan X., Dapeng D., and J. P. Morrison. An ontology for heterogeneous resources management interoperability and hpc in the cloud. *Future Generation Computer Systems*, 88:373–384, 2018.
[34] A. Willner, M. Giatili, P. Grosso, C. Papagianni, and I. Morsey, M.and Baldin. Using semantic web technologies to query and manage information within federated cyber-infrastructures. *Data*, 2(3), 2017.
[35] K. Yongsiriwit, M. Sellami, and W. Gaaloul. A semantic framework supporting cloud resource descriptions interoperability. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 585–592, 2016.
[36] S. Challita, F. Paraiso, and P. Merle. Towards formal-based semantic interoperability in multi-clouds: The fclouds framework. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 710–713, 2017.
[37] B. Bazeer Ahamed and Murugan Krishnamoorthy. *Invocation of Multi-Cloud Infrastructure Services in Web-Based Semantic Discovery System*, pages 3–18. Springer International Publishing, Cham, 2022.
[38] L. Liu, X. Yao, L. Qin, and M. Zhang. Ontology-based service matching in cloud computing. In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 2544–2550, 2014.
[39] J. Gorroñogoitia, Z. Vasileiou, E. Imperiali, I. Kumara, and G. Meditskos D. Radolović. A smart development environment for infrastructure as code. *SWForumWorkshop 2021*, 2021.
[40] G. Meditskos, A. Karakostas Z. Vasileiou, S. Vrochidis, and I. Kompatsiaris. A pattern-based semantic lifting of cloud and hpc applications using owl 2 meta-modelling. In *4th Special Session on High Performance Services Computing and Internet Technologies*, 2020.
[41] M. C. Suárez-Figueroa, A. Gomez-Perez, E. Motta, and A. Gangemi. *Ontology Engineering in a Networked World*. 2012.
[42] D. Allemang and J. Hendler. *Semantic web for the working ontologist - modeling in RDF, RDFS and OWL*. 01 2008.
[43] N. Jekjantuk and J. Z. Gröner, G.and Pan. Modelling and reasoning in metamodelling enabled ontologies. In *Proceedings of the 4th International Conference on Knowledge Science, Engineering and Management*, page 51–62, Berlin, Heidelberg, 2010. Springer-Verlag.
[44] A. Brogi and A. Tommaso. Sommelier: A tool for validating tosca application topologies. In *Communications in Computer and Information Science series*, volume 880, pages 1–22, 2018.
[45] J. E. Labra Gayo, E. Prud'hommeaux, I. Boneva, and D. Kontokostas. Validating rdf data. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 7:1–328, 2017.
[46] E. Blomqvist, A. Seil Sepour, and V. Presutti. Ontology testing - methodology and tool. In *Knowledge Engineering and Knowledge Management*, pages 216–226. Springer Berlin Heidelberg, 2012.
[47] M. Poveda Villalón. Ontology evaluation: a pitfall-based approach to ontology diagnosis. 2016.
[48] B. Lantow. Ontometrics: Putting metrics into use for ontology evaluation. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. SCITEPRESS - Science and Technology Publications, Lda, 2016.
[49] A. Reiz, K. Dibowski, H.and Sandkuhl, and B. Lantow. Ontology metrics as a service (omaas). pages 250–257, 2020.
[50] A. Mustafa M., H. Hesham A., and A. Omara. Towards evaluation of cloud ontologies. *Journal of Parallel and Distributed Computing*, 126:82 – 106, 2019.
[51] K. Siegemund, U. A., J. Pan, E. Thomas, and Y. Zhao. Towards ontology-driven requirements engineering. 2011.