SOftware Defined AppLication Infrastructures managemenT and Engineering

First version of ontologies and semantic repository

D3.1

CERTH
31.7.2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825480.

Deliverable data			
Deliverable	First version of ontologies and semantic repository		
Authors	Georgios Meditskos (CERTH), Zoe Vasileiou (CERTH), Panagiotis Mhtzias (CERTH), Anastasios Karakostas (CERTH), Stefanos Vrochidis (CERTH), Jesús Gorroñogoitia (ATOS)		
Reviewers	Kalman Meth (IBM), Román Sosa González (ATOS)		
Dissemination level	Public		
History of changes	Panagiotis Mitzias (CERTH)	Outline created	7.10.2019
	All	Initial partner contributions	19.12.2019
	All	Additional partner contributions	14/01/2020
	Georgios Meditskos	Version ready for review	15/01/2020
	Kalman Meth, Román Sosa González	Review forms available	21/01/2020
	All	Final	28/01/2020
	Georgios Meditskos	Updated ToC to address reviewers' recommendations	03 /05/2020
	All	Partner contributions	01/06/2020
	Georgios Meditskos	Version ready for review	01/07/2020
	Kalman Meth, Román Sosa González	Review forms available	08/07/2020
	All	Final	20/07/2020

Acknowledgement

The work described in this document has been conducted within the Research & Innovation action SODALITE (project no. 825480), started in February 2019, and co-funded by the European Commission under the Information and Communication Technologies (ICT) theme of the H2020 framework programme (H2020-ICT-16-2018: Software Technologies)

Table of Contents

List Of Images

List Of Tables

Executive Summary

The present deliverable reports on the work carried out within T3.1 “Application Semantic Modelling” and T3.2 “Infrastructure Semantic Modelling”, relevant to the development of the SODALITE ontologies (abstraction layer) and the representation and mapping of cloud applications and infrastructures to ontological entities. In addition, it describes the first version of the Semantic Reasoner module that populates the SODALITE Knowledge Base (KB) with the resource models and Abstract Application Deployment Models (AADMs) composed through the textual editor (Application Developer IDE). It also provides the REST API that different modules use to get information from the KB.

More specifically, the deliverable presents the current content of the SODALITE semantic models and the abstract conceptual model (Ontology Design Pattern [1]) that has been adopted to build them. Based on the requirements set forth by WP2 and the dependencies incurring from the interaction with the other WPs (mainly with WP4 in the first year of the project), the purpose, scope, intended uses and the requirements of the SODALITE ontology were identified. These specifications, along with the modelling insights from the relevant literature, served as guidelines for building the first version of the ontologies that currently comprises modules for capturing Cloud application and resources modelled using TOSCA node types and node templates. All this information is used to build the SODALITE knowledge graphs that capture and interlink cloud-related information.

In addition, we present the preliminary version of the reasoning layer whose purpose is: a) to populate the Knowledge Base (KB) of SODALITE with application and resource models defined by end users; b) to provide the basic reasoning infrastructure to be used in WP4 (T4.4) for implementing advanced reasoning services, such as semantic retrieval and validation services; and c) to provide the REST API needed to retrieve and store information in the KB.

Finally, we describe the specifics of the textual editor (Application Developer IDE) of SODALITE that allows end users to define AADMs by reusing components and resources from the KB. More specifically, we describe the DSL used to define AADM in an abstract level, along with technical details on the interaction between the IDE and the semantic infrastructure of SODALITE in terms of exchange models and REST API.

Glossary

Acronym	Explanation
AADM	Abstract Application Deployment Model
AOE	Application Ops Experts The equivalent process from the ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes is Operation processes and maintenance processes
API	Application Program Interface
CP	Content Pattern
CAMP	Cloud Application Management for Platforms
DL	Description Logic
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
DnS	Descriptions and Situations
DSML	Domain Specific Modeling Language
MDE	Model Driven Engineering
DSL	Domain Specific Language
DUL	DOLCE Ultralite
EMF	Eclipse Modelling Framework
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IDE	Integrated Development Environment
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
KB	Knowledge Base
OASIS	Organization for the Advancement of Structured Information Standards
ODP	Ontology Design Pattern
OWL	Web Ontology Language
PaaS	Platform as a Service
QE	Quality Expert The equivalent process from ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes: Infrastructure management and Configuration management processes
QoS	Quality of Service
RDF	Resource Description Framework

RDFS	Resource Description Framework Schema
RDF4J	Resource Description Framework for Java (open source java framework)
RE	Resource Expert The equivalent process from ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes is Quality Management and Quality assurance processes
REST	Representational State Transfer
RM	Resource Model
SaaS	Software as a Service
SHACL	Shapes Constraint Language
SPARQL	SPARQL Protocol and RDF Query Language
SPE	Semantic Population Engine
SRE	Semantic Reasoning Engine
SWRL	Semantic Web Rule Language
TOSCA	Topology and Orchestration Specification for Cloud Applications
WP	Work Package
XML	Extensible Markup Language

1 Introduction

Cloud computing has revolutionised IT and has become a popular paradigm for the provision of computing infrastructure with convenient and scalable access to computing resources. One of the most important challenges in this domain is *interoperability*, since numerous vendors have introduced different paradigms and services, making the cloud landscape diverse and heterogeneous. The two noteworthy dimensions of interoperability in the cloud domain – connectivity and usability – have been divided into five layers [2]: 1) *Transport Interoperability*: the exchange of data using physical networks, such as the Internet; 2) *Syntactic Interoperability*: the structure and coding of the data; 3) *Semantic Interoperability*: the intended meaning of the data; 4) *Behaviour Interoperability*: service behaviour under certain conditions; 5) *Policy Interoperability*: conformance of interacting systems to applicable laws, regulations and organizational policies.

In addition, the ability to move cloud services between different cloud environments, standardised service definitions, and be vendor independent, can help technologies based on cloud computing reach the next level. Portability between two systems must be considered separately for data and for applications, since the factors of relevance differ substantially between these. Data portability, on the one hand, is moving the data and/or applications from one system to another and having it remain usable or executable. Application portability, on the other hand, is the ability to migrate an application from one cloud service to another or between a customer's environment and a cloud service.

Many cloud resource management standards have been proposed to cope with different aspects of interoperability and portability. For example, TOSCA[1] (Topology and Orchestration Specification for cloud Applications), OCCI[2] (Open Cloud Computing Interface), and CIMI[3] (Cloud Infrastructure Management Interface) are among the most known standards that focus on the interoperable description of application and infrastructure cloud services, the relationships between parts of the service, dependencies between application components, cloud computing management tasks, etc. However, cloud resources that have been modelled and described using different standards still face interoperability problems, since the provided modelling languages and semantics differ, resulting in heterogeneous schemata and vocabularies that create cloud silos and non-reusable cloud resources. In order to enable the development, management, faster execution and reuse of complex applications, both the application's components and infrastructures should be modelled in a standardised, machine-readable format and abstraction level.

Semantic Web technologies and, particularly, ontologies and reasoning can promote interoperability and intelligent decision support mechanisms for various cloud-based services, providing effective interoperability among the Cloud based systems and resources. A great body of work has focused on the semantic representation of cloud-based services, resources and infrastructures (see Section 2.1.3 for more details). While the majority of the existing solutions are based on the semantic annotation of resources, APIs, operations, infrastructures, etc., in SODALITE we follow a knowledge-intensive, Ontology Design Pattern (ODP) paradigm to build the Modelling Layer and achieve semantic interoperability. The SODALITE's rich conceptual meta-model enables the formal abstraction of applications and infrastructures, allowing the semantic annotation and interlinking of functional and non-functional requirements, software dependencies, service capabilities and QoS, reusing existing general-purpose ontologies and extend them to our domain. The models focus on capturing information at higher levels of abstraction, enabling the conceptual description of artefacts, services, code and platforms that will foster advanced context-aware searching, matchmaking, validation and reuse.

[1] http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html
[2] https://occi-wg.org/
[3] https://www.dmtf.org/standards/cloud

However, although an ontological model is preferable at a higher level, developers usually prefer a detailed, concrete syntactic representation. A DSL can provide a more programmer-oriented representation of descriptions, offering a lightweight language abstraction level in an appealing manner to the software engineering community, hiding the complexity of the ontology language and the conceptual schema. Through reasoning and intelligent mapping services, the DSL can be then transformed to the rich underlying conceptual model. To this end, SODALITE's Modelling Layer is enriched with a textual editor (Application Developer IDE) that allows end users to describe applications and resources using a DSL.

This deliverable aims to present the first version of the SODALITE Modelling Layer, which consists of the Semantic Knowledge Base, the Semantic Reasoner and the SODALITE IDE to support users in defining resource and application models.

1.1 Objectives

In the following sections we clearly describe the objectives and motivations of using the two key technologies of this deliverable, namely ontologies and IDE. More specifically, we describe the concepts and principles that underpin our work, the expected results, and we associate them with the methods and tools developed in the project to meet the overall objectives. More details on the innovation and the baseline tools are provided in sections 2.3 and 6, respectively, whereas D2.4 "Guidelines for Contributors to the SODALITE Framework" elaborates on the quality of the developed artefacts and the guidelines for software and release management.

1.1.1 Ontologies

In SODALITE, one of the objectives is to use ontologies as the means to achieve knowledge interoperability, defining layers of conceptual abstractions to represent models and domain knowledge. More precisely, through the use of ontologies, SODALITE aims to:

- **Follow a common, extensible and formal standardised model to describe cloud-related concepts:** Ontologies are used in order to define the interoperable abstraction layer in SODALITE, allowing the formal representation of applications and infrastructures, the semantic annotation and interlinking of functional and non-functional requirements, software dependencies, service capabilities and QoS. To this end, SODALITE capitalizes on and reuses existing general purpose ontologies, adapting them to the application domain. As we describe in section 3, the OWL 2 ontology language is used (W3C recommendation) for defining the SODALITE meta-model and the underlying semantics.
- **Manage and share information in the form of interconnected resources in RDF knowledge graphs for capturing structural and semantic relationships:** Ontologies are semantic data models that define the types of things that exist in our domain and the properties that can be used to describe them. They are essentially generalized data models, defining only general types of things that share certain semantics and properties, but they don't include information about specific individuals in our domain. The instantiation of ontologies with real-world data that aggregate knowledge of real-world entities and their relationships forms the Knowledge Graphs. Using Knowledge Graph, information is represented as a network of relationships, instead of as separate tables, capturing both structural and semantics relationships in an unambiguous manner. Section 4 elaborates on the generation of the SODALITE Knowledge Graphs and the assertions of abstract relationships among cloud components.
- **Reuse of automated sound and complete reasoning tools for knowledge enrichment and consistency checking:** Reasoning over knowledge graphs aims to identify errors and infer new conclusions from existing data. New relations among entities can be derived through knowledge reasoning and can feed back to enrich the knowledge graphs. By capturing knowledge in the form of interoperable RDF knowledge graphs with formal

semantics, we can reuse existing logic-based frameworks and rule languages to enrich SODALITE with an interpretation and validation layer. For example, custom reasoning logic can infer semantic validation errors and smell detection [3], following the semantics of TOSCA and the interconnection defined in the Knowledge Graphs

- **Foster interlinking with external datasets:** Apart from capturing relationships in a knowledge base only among the entities that are known in the system, another important aspect is the ability to semantically enrich the Knowledge Graphs with external knowledge (Linked Open Data), fostering advanced searching and reasoning services. For example, Yago [4] is a prominent RDF knowledge graph extracted from Wikipedia that can be interlinked with local knowledge graphs in order to improve the semantic content. For example, Entity Linking and Word Sense Disambiguation (e.g. BabelNet[4], WordNet[5]) on the textual descriptions of TOSCA components can be used for advanced searching and classification services, improving user experience.

1.1.2 IDE

The main objective of adopting an IDE in SODALITE is the integration of the main application development environment with the main application deployment and governance environment, both toolsets coexisting and interacting within the same framework. In particular, the IDE aims to support the following features across the project lifetime:

- **DSML development and management:** The IDE offers not only a complete development environment, but also a complete framework for DSML development and management, and also for model driven engineering, supporting inter-DSML transformation and code generation. A significant baseline of MDE and DSML technologies, including some de-facto standards are supported by the IDE (see section 2.2.1)
- **Multiview specification of application deployment topology**: The IDE supports the abstract specification of application deployment topologies, adopting both textual and visual notations, and permit to visualize different facets of the same deployment model as viewpoint representations. Textual notation suits the fast modeling needs of skilled application owners, while visual notation facilitates the understanding of complex topologies and improves their communication among other team roles.
- **Inter-DSML reference and resolution**: The IDE supports inter-DSML reference and resolution that largely simplifies the design of the SODALITE meta-models, which can be split into specialized meta-models, which reference each other. In this way, users adopting specialized roles can use customized interlinked DSML meta-models for describing infrastructures, resources, application deployment topologies, optimizations and models for other concerns in a simpler manner.
- **Semantic-driven modeling assistance:** The IDE leverages the semantic knowledge base (KB) to assist the modelers with hints and suggestions during the process of describing their application deployment models. KB inference and reasoning capabilities can be exploited to conduct semantic validation of application deployment models in order to spot errors and provide recommendations. Moreover, the KB can suggest to modelers available infrastructure resources that satisfy the requirements expressed by the application components, either at design (through the IDE), deployment or runtime.
- **Centralised management dashboard:** Through the IDE, application owners have a centralised unique dashboard to design their deployment models, deploy them and manage their application lifecycle.

[4] "BabelNet." https://babelnet.org/.
[5] "WordNet | A Lexical Database for English." https://wordnet.princeton.edu/.

1.2 Overview of SODALITE architecture

For the purpose of introduction, we present a short synopsis of the SODALITE architecture that has been described in the previous public deliverable D2.1 (Section 3). For full details, please check the functional description, inputs, outputs, and dependencies of each component.

SODALITE aims to provide developers and infrastructure operators with tools that abstract their application and infrastructure requirements to enable simpler and faster development, deployment, operation, and execution of heterogeneous applications on heterogeneous, software-defined, high-performance, cloud infrastructures. To this end, SODALITE aims to produce:

- A pattern-based abstraction library that includes application, infrastructure, and performance abstractions.
- A design and programming model for both full-stack applications and infrastructures based on the abstraction library.
- A deployment framework that enables the static optimization of abstracted applications onto specific infrastructure.
- Automated run-time optimization and management of applications.

The SODALITE platform is divided into three main layers, each covered by a separate work package (WP). These layers are the Semantic Modelling layer (WP3), the Infrastructure as Code layer (WP4), and the Runtime layer (WP5). Figure 1 below shows these layers together with their relationships. This deliverable focuses on the Semantic Modelling layer, describing the specifics of the layer and the progress that has been achieved in the first year of the project.

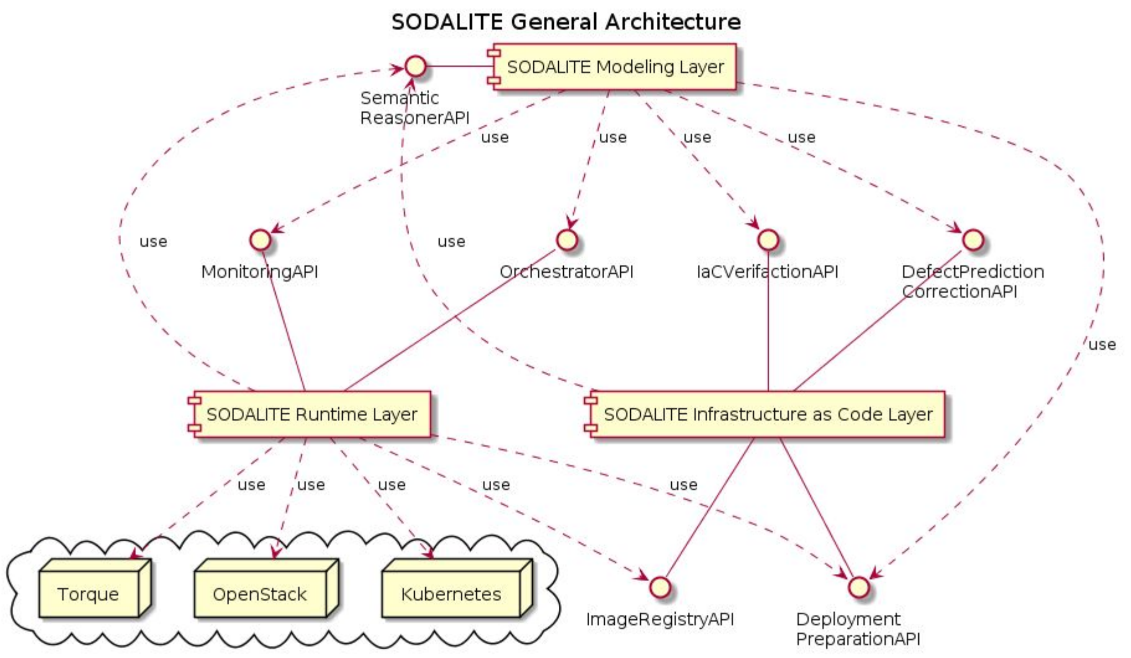


Figure 1. SODALITE overall Architecture

1.2.1 Semantic Modelling

The components of the SODALITE Semantic Modelling Layer are depicted in the following figure (Figure 2).

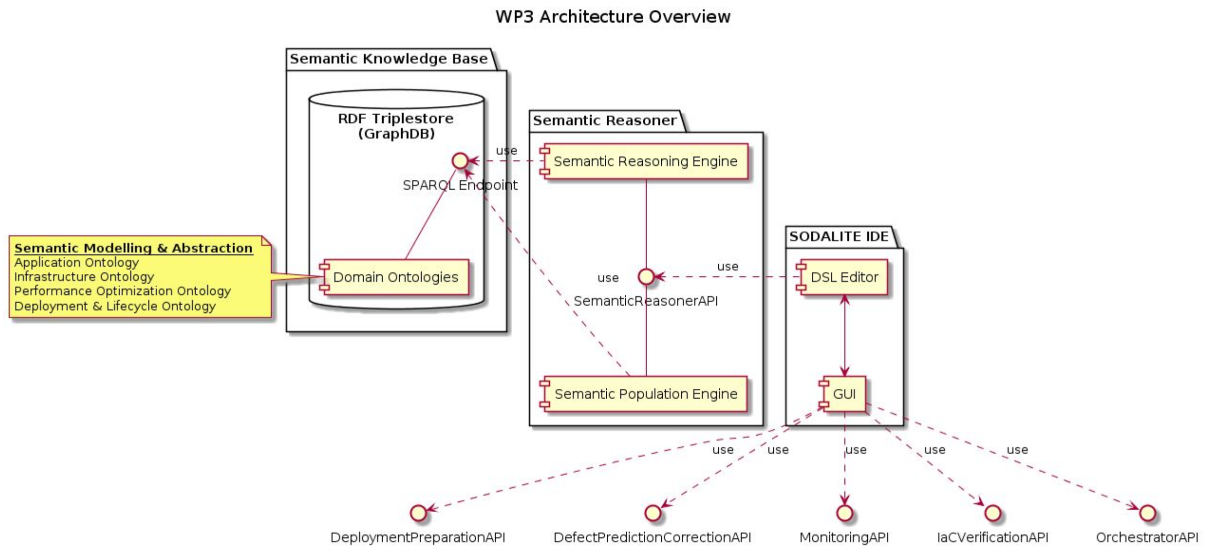


Figure 2. SODALITE Semantic Modelling components (WP3)

The main objective of the Semantic Modelling layer is to provide the framework for semantically representing abstractions of a) cloud applications, capturing higher-level information that will enable the conceptual description of artefacts, code, functional and non-functional requirements, software dependencies etc., and b) cloud infrastructures, available services and service capabilities in terms of functionalities, resources and business characteristics offered, and QoS. These semantic abstractions are realised in the form of RDF Knowledge Graphs, aiming at the formal representation and linking of application and infrastructure requirements that enables semantic reasoning framework to be developed on top of the RDF graphs to support search, discovery, validation and reuse.

To this end, three main modules have been defined:

1. The **Semantic Knowledge Base** (KB), which is SODALITE's semantic repository (RDF triple store) that hosts the models (domain ontologies), created in WP3.
2. The **Semantic Reasoner**, which is a logical middleware that facilitates the interaction with the KB through the REST API (Semantic Reasoning Engine module - SRE), as well as the population of the KB with information coming from the SODALITE IDE users (Semantic Population Engine module - SPE).
3. The **SODALITE IDE** that provides the GUI and the DSL Editor to assist end users in composing resource and application models.

In the next section, we provide more details about the technologies and standards we have used to implement these components and their modules.

1.3 Structure of the Document

This deliverable is structured as follows:

- Section 2 presents background and related work on the domains of a) knowledge representation and ontologies (section 2.1), describing the basic semantics that underpin ontologies, W3C standards for creating and sharing ontologies, existing ontology-based solutions in the domain of software engineering, as well as best practices in the ontology development process, and b) Domain Specific Languages and existing IDE solutions (Section 2.2). We also present the key innovations of the developed technologies (Section 2.3), compared to the state of the art presented in Section 2.
- Section 3 presents the Semantic Models of SODALITE, i.e. the abstract Ontology Design Pattern (ODP) that is used to capture definitions of application and resources. The section describes the modelling components, i.e. the different logical modelling layers / tiers

defined, and the SODALITE Knowledge Base (KB). It also elaborates on the development status and the functionality supported in the first year of the project, as well as next steps.

- Section 4 describes the Semantic Reasoner of SODALITE that runs on top of the KB that supports ontology population and checking. We describe the reasoning services that have been implemented to support basic (OWL 2 native) reasoning functionality and the mapping of TOSCA-related definitions to the conceptual model of SODALITE. The section concludes with the development status and next steps.
- Section 5 presents the IDE that has been developed to assist Resource Experts (REs) and Application Ops Experts (AOE).
- Section 6 describes the WP3 technology stack, i.e. the set of technologies, frameworks and standards that are (re)used and extended to develop the components of the Semantic Modelling layer.
- Section 7 concludes the deliverable, presenting next steps.

2 Related Work

2.1 Knowledge Representation and Ontologies

In literature, ontologies have been widely used as an effective way for modelling domain information because they can represent and organize information, the context and relationships more accurately, especially considering the necessity of dynamic changes. In addition, they offer easy expandability by merging, expanding and combining parts of existing ontologies.

Ontologies are models used to capture knowledge about some domain of interest. Formally speaking, ontologies are *explicit formal specifications of shared conceptualizations* [5],[6]. They represent abstract views of the world, including the objects, concepts, and other entities that are assumed to exist in some area of interest, their properties and the relationships that hold among them. Their expressivity and level of formalisation depend on the knowledge representation language used.

Within the Semantic Web, which is an extension of the current Web that aims to establish a common framework for sharing and reusing data across heterogeneous sources, ontologies play a key role. The Semantic Web vision is to make the semantics of web resources explicit by attaching to them metadata that describe meaning in a formal, machine-understandable way. In this effort, the Web Ontology Language (OWL) [7] has emerged as the official W3C recommendation for creating and sharing ontologies on the Web that is based on Description Logics (DLs).

2.1.1 Description Logics

Description Logics (DLs) [8] are a family of knowledge representation formalisms characterised by logically grounded semantics and well-defined reasoning services. The main building blocks are concepts representing sets of objects (e.g. Nodes), roles representing relationships between objects (e.g. dependencies), and individuals representing specific objects (e.g. virtual_machine). Starting from atomic concepts, such as Property, arbitrary complex concepts can be described through a rich set of constructors that define the conditions on concept membership. For example, the concept ∃hasProperty. Property describes those objects that are related through the hasProperty role with an object from the concept Property; intuitively, this corresponds to all those individuals that have at least one property. A DL knowledge base K typically consists of a TBox T (terminological knowledge) and an ABox A (assertional knowledge). The TBox contains axioms that capture the possible ways in which objects of a domain can be associated. For example, the TBox axiom Compute ⊑ Root asserts that all objects that belong to the concept Compute, are members of the concept Root too. The ABox contains axioms that describe the real world entities through concept and role assertions. For example, Compute(vm) and hasProperty(vm, name) express that vm has a property, which is described by the 'name' instance. Table 1 summarises the set of terminological and assertional axioms in DLs.

Table 1. Terminological and assertional axioms

Name	Syntax	Semantics
Concept inclusion	$C \sqsubseteq D$	$C^I \subseteq D^I$
Concept equality	$C \equiv D$	$C^I = D^I$
Role Equality	$R \equiv S$	$R^I = S^I$
Role inclusion	$R \sqsubseteq S$	$R^I \subseteq S^I$
Concept assertion	$C(a)$	$a^I \in C^I$
Role assertion	$R(a,b)$	$(a^I, b^I) \in R^I$

The semantics of a DL language is formally defined through an interpretation I that consists of a nonempty set Δ^I (the domain of interpretation) and an interpretation function $\cdot^I$, which assigns to every atomic concept A a set $A^I \subseteq \Delta^I$ and to every atomic role R a binary relation $R^I \subseteq \Delta^I \times \Delta^I$. The interpretation of complex concepts follows inductively. Table 2 shows the syntax and semantics of some of the most common DL constructors.

Table 2. Examples of concept and role constructors

Name	Syntax	Semantics
Top	$\top$	Δ^I
Bottom	$\perp$	$\emptyset$
Intersection	$C \sqcap D$	$C^I \cap D^I$
Union	$C \sqcup D$	$C^I \cup D^I$
Negation	$\neg C$	$\Delta^I \setminus C^I$
Universal Quantification	$\forall R.C$	$\{a \in \Delta^I \mid \forall b. (a,b) \in R^I \rightarrow b \in C^I\}$
Existential Quantification	$\exists R.C$	$\{a \in \Delta^I \mid \exists b. (a,b) \in R^I \wedge b \in C^I\}$
Inverse	R^-	$\{(b, a) \in \Delta^I \times \Delta^I \mid (a,b) \in R^I\}$
Transitive Closure	R^+	$\bigcup_{n>1} (R^I)^n$
Composition	$R \bigcirc S$	$R^I \circ S^I$

2.1.2 RDF, OWL and OWL 2

The Resource Description Framework[6] (RDF) standard was originally released as a W3C Recommendation in 1999, and was updated in 2004 and in 2014. The RDF standard consists of two major components: a data model and language for representing data, and syntax standards for expressing, exporting, and parsing the data model and language. The RDF data model is based on graphs, as opposed to the tuples that underlie traditional relational data models. In RDF, a data graph is constructed by the union of a number of three part assertions, called triples. A triple consists of a subject, a predicate, and an object, in which the subject is an entity about which some data is expressed, the predicate can be seen as the typing of the related data, and the object is the actual related data relevant to the subject [9]. The RDF Schema (RDFS) standard, released along with the second generation of RDF in 2004 (and updated in 2014), defines classes and properties that extend the base RDF vocabulary and provides support for more expressive knowledge modelling semantics. Using the RDFS vocabulary it is possible to model complex data structures, including basic ontologies.

[6] http://www.w3.org/RDF/

The Web Ontology Language[7] (OWL) was developed simultaneously with RDFS to provide better support for such higher-level expressiveness. OWL is a knowledge representation language widely used within the Semantic Web community for creating ontologies. The design of OWL and particularly the formalisation of the semantics and the choice of language constructors have been strongly influenced by DLs. OWL comes in three dialects of increasing expressive power: OWL Lite, OWL DL and OWL Full. OWF Full is the most expressive of the three: it neither imposes any constraints on the use of OWL constructs, nor lifts the distinction between instances (individuals), properties (roles) and classes (concepts). This high degree of expressiveness comes however at a price, namely the loss of decidability that makes the language difficult to implement. As a result, the focus has been placed on the two decidable dialects, and particularly on OWL DL, which is the more expressive of the two.

Despite the rich primitives provided for expressing concepts, OWL DL has often proven insufficient to address the needs of practical applications. This limitation amounts to the DLs style model theory used to formalise its semantics, and particularly the *tree model property* [10] conditioning DLs decidability. Therefore, OWL can model only domains where objects are connected in a tree-like manner. This constraint is quite restrictive for many real-world applications, including the ambient intelligence domain, which requires modelling general relational structures.

Responding to this limitation and to other drawbacks that have been identified concerning the use of OWL in different application contexts throughout the years, the W3C working group produced OWL 2 [11]. OWL 2 is a revised extension of OWL, commonly referred to as OWL 1. It extends OWL 1 with qualified cardinality restrictions. Another prominent OWL 2 feature is the extended relational expressivity that is provided through the introduction of complex property inclusion axioms (property chains). To maintain decidability, a regularity restriction is imposed on such axioms that disallow the definition of properties in a cyclic way. Three profiles, namely OWL 2 EL, OWL 2 QL and OWL 2 RL, trade portions of expressive power for efficiency of reasoning targeting different application scenarios.

2.1.3 Semantic Web Technologies in Cloud

Semantic Web technologies and, particularly, ontologies and reasoning can promote intelligent decision support mechanisms for various Cloud based services, providing effective interoperability among the Cloud based systems and resources.

A great body of work has focused on the semantic representation of Cloud-based services, resources and infrastructures. OpenCrowd's Cloud Taxonomy[8] is an online, freely navigable taxonomy that categorizes Cloud Services according to both their service model (IaaS, PaaS or SaaS) and application context. It enables users to discover and access Cloud services so that they can further navigate to respective home pages. Moving beyond just a static model, the Cloud Taxonomy is interactive, where users can contribute comments and recommend additional products to include, aiming at encouraging the dialog between Cloud computing services vendors and developers. The authors in [12] focus on semantic-based QoS management and monitoring for cloud-based systems and propose a new framework that combines semantic technologies and distributed data stream processing techniques. In [13], a framework is described that facilitates the discovery of Cloud services (IaaS). [14] emphasises on migrating cloud applications between cloud platforms. A Service Oriented Cloud Computing Architecture (SOCCA) is proposed where cloud computing resources are componentized, standardized and combined in order to build a "cross-platform virtual computer".

Various European research projects deal with issues related to using semantics for PaaS portability and interoperability. The Cloud4SOA[9] project focuses on resolving interoperability and portability issues existing in current Cloud infrastructures and on introducing a user-centric approach for

[7] https://www.w3.org/TR/owl-features/
[8] http://cloudtaxonomy.opencrowd.com
[9] http://www.cloud4soa.com/

applications that are built upon and deployed using Cloud resources. The mOSAIC[10] project aimed at creating, promoting and exploiting an open-source Cloud application programming interface and a platform targeted for developing multi-Cloud oriented applications.

The PaaSport project [15] focuses on resolving cloud platform interoperability and cloud application portability issues that exist in the Cloud PaaS market through a flexible and efficient deployment and migration approach. To this end, PaaSport combines Cloud PaaS technologies with lightweight semantics in order to specify and deliver a thin, non-intrusive Cloud broker (in the form of a Cloud PaaS Marketplace), to implement the enabling tools and technologies, and to deploy fully operational prototypes. PaaSport uses three semantic models [16], the offering model, which represents the description of a cloud computing platform, the application model which represents the cloud-based requirements of the application or software/resource dependencies on the hosting cloud platform, and the SLA model which represents the agreement between the application owner and the PaaS provider. For offering extensibility, the PaaSport ontology uses DnS ontology as an upper-level ontology.

In [17], the authors proposed an ontology representation of the TOSCA standard to fill the gap between the structural aspect and the domain of the applications. The aim is to semantically annotate the TOSCA specification with meaningful information which can give a clearer vision of the concepts involved in each of components, rules and relationships defining them. To this end, the paper proposes the definition and subsequent creation of a TOSCA structural ontology that, joint with the appropriate domain ontologies, can help give a fully semantic level to a modelled application, thus giving new opportunities for improving its development, deployment, and portability in the cloud. The focus is mainly given in capturing the structural relationships of the standard rather than generating an abstract model to represent different aspects of the applications.

Open-Multinet (OMN) [18] is a bundle of ontologies for providing a common standardized model for federated cloud environments. The ontology consists of nine ontologies, an upper ontology and eight descendant ontologies. Those ontologies can be used formally to describe a federation of e-infrastructures , including the attributes and types of resources as well as services available within the federation. Also, the lifecycle of a collection of resources and services is also modeled. OMN framework converts tree-based resource specification models to semantic models allowing easy discovery of cloud resources.

FClouds [19] is a framework for semantic interoperability in multi-cloud environments. It provides cloud formal models for cloud structure and cloud api operations and then reason over them for proving some properties. Having defined the semantics of each cloud, transformations functions can ensure interoperability from one cloud API to another. Until now, transformations are available for OCCI, TOSCA GCP and AWS.

An ontology model has been proposed in [20] for efficient discovery of high performance computing resources/services distributed in cross-regional supercomputing centers. An upper ontology is used, named HPCRO, modeled with the 5W1H principle. The domain ontology facilitates the description of cross-regional hardware and software resources. For example, computing capabilities such as network and accelerators are models for hardware resources, and application libraries, operating systems and compilers for software resources. Furthermore, the semantic model has been extended by Quick Service Query List representing a resource index list containing semantic relationships. The WQIRL algorithm is used, combining QSQL and WordNet, for fast resource discovery, which uses WordNet database for mapping a published resource to the best ontology concept in index list.

In [3] an (anti) pattern ontology and SWRL detection rules are used to form the Knowledge Base and propose recommendations to the cloud API developer through SPARQL queries. Four

[10] http://www.mosaic-cloud.eu/

ontologies are modeled, namely, two ontologies for representing the features of OCCI and REST API and a pattern and (anti) pattern ontology.

The methodology proposed in [21] defines an iterative approach for solving interoperability problems. It uses semantic web annotations, semantic web services, ontology and the AI planning methods. Briefly, the user selects a use case, performs interoperability analysis for choosing an ontology of interoperability problems, and an AI planner. This paper does not propose an ontology but a set of steps for creating a service for solving interoperability problems for use cases that the user prefers to execute.

The work in [22] describes the development of two ontologies to capture functionalities, features and interoperability problems among APIs of different providers of PaaS. The domain of the first ontology is the representation of resources and operations in APIs of PaaS in order to map resources of different APIs. This ontology is mostly based on the mOSAIC ontology; The concepts are derived from many projects and some cloud computing standards (TOSCA, OCCI, CDMI). From TOSCA, it models properties, capabilities, interfaces, operation and requirements. The second ontology represents technical and semantic interoperability problems of commercial platforms as a service offers.

2.1.4 Meta-modelling

Meta-modelling is the name commonly given to the practice of using a model to describe another model as an instance [23]. Applications of meta-modelling in Ontology Engineering are manifold, including the representation of provenance or versioning information, as well as the documentation of modelling decisions. One feature of meta-modelling is that it must be possible to assign properties to classes in the model. This way it is possible to assert the membership of classes in meta-classes and interconnect them via meta-roles [24]. However, putting properties on classes typically violates the separation of class and individual that allows a model to be described in OWL DL.

Ontology languages differ with respect to their support for meta-modelling. While it is supported by OWL Full, this high expressivity leads to undecidability [25]. One variant, which is also supported by OWL 2 DL, is called punning [26]. Punning allows for using the same identifier, e.g. for an individual and a class. The class and its corresponding individual are, however, treated as entirely independent.

A number of motivations for meta-modelling exist. One such motivation is that a model often needs to play more than one role in an application: A particular concept should be viewed as a class in one role but as an instance in another role. For example, as we describe in Section 3.3, TOSCA nodes need to be treated both as classes and as individuals: in the former case, by capturing nodes as classes, we reuse the ontology semantics regarding subsumption hierarchies. However, descriptive information regarding nodes, e.g. capabilities and interfaces can be captured only by treating nodes as instances as well, in pattern instantiations. In addition, by espousing meta-modelling, we allow the representation of contextualised views on complex situations, affording reusable pieces of knowledge that cannot otherwise be expressed by the standard ontology semantics. Moreover, the meta-model enables the reuse of the encapsulated semantics across domains with similar scope but different implementation frameworks, by translating them into the respective framework language.

2.1.5 Ontology Design Patterns

As ontology engineering became more broadly used, knowledge engineers needed ways to optimize and accelerate parts of the ontology development process. One of the approaches was employing Ontology Design Patterns (ODPs) - small, modular, and reusable solutions to recurrent modelling problems - and templates based on these patterns or other representation regularities in the ontology[27]. ODPs provide a consistent way for developing ontologies [28]. They can be viewed as modularised foundational ontology fragments that serve as design snippets for good modelling practices. They also can be viewed as a way of bottom-up pattern finding that is then reused across the ontology and offered to others as a 'best practices' design solution for some modelling aspect.

There are different types of ODPs. For example, the Logical Pattern deals with the absence of some features of representation languages, such as to capture n-aries relations in OWL. The Content Pattern (CP) [29] is one of the most frequently used ODP. It can be considered roughly analogous to a software design pattern, with the added benefit that it includes a reference base implementation (in the form of an OWL 2 building block) ready for immediate customisation. CPs are small ontologies that mediate between use cases (problem types) and design solutions. They are used as modelling components: ideally, an ontology results from a composition of CPs, with appropriate dependencies between them, plus the necessary design expansion based on specific needs. SODALITE's conceptual model reuses the DnS [30] ontology pattern of DOLCE+DnS Ultralite (DUL) ontology[11] [31], so as to capitalise on the high axiomatisation provided by DUL and achieve a better degree of knowledge sharing, reuse and interoperability.

2.2 IDE

2.2.1 User-centered DSL workbenches

Domain specific languages (DSL) are those intended for the purpose of representing knowledge in a particular application domain; they are designed as an abstraction that focuses on the conceptualization of such a domain. Concrete realities in the domain are conceptualized as model instances that are compliant with the DSL meta-model. A framework that assists modelers in creating these DSLs, maintain and transform them is called a DSL workbench[12]. Workbenches offer full fledged features that facilitate the authoring of DSL compliant models, including customizable syntax highlighting, context-aware code assistance and completion, syntactic and semantic validation, error marking and quick fixing, code generation, reference resolution, navigation in outline views, folding model sections, model comparison, auto formatting, DSL testing and others [32]. DSLs are specified through meta-models or grammars and can adopt different modeling notation, including textual, graphical, tabular, etc. [32][33]. Several textual based workbenches, including pure textual, and textual projectional, have been proposed and implemented [34][35], including Xtext [36], Textual Editing Framework (TEF)[13], Textual Concrete Syntax, TCS [37], EMFText [38]**,** MetaModeling System (MMS)[14], Monticore [39], Rascal MPL [40], Spoofax [41], and others. Pure text workbenches support free textual edition of DSL models, which are parsed and validated before they are converted into an abstract syntax tree (AST) model. Projectional workbenches provide an editable projection of the AST, which is edited by the modeler.

XText is one of the most popular industrial, mature, textual-based workbenches in the Eclipse ecosystem. DSLs are specified in Xtext with concrete syntax grammars, which are translated into Ecore models (the abstract syntax), while a modeling workflow engine file generates the Java model source code tree and the DSL editor code, which is integrated within the Eclipse UI. This engine also generates Xtend classes for processing customized validation, code-assistance, etc, and XPand templates for code generation.

TEF is an academic prototype of textual workbench based on Eclipse, that uses a Textual Syntax Language (TSL) to define the DSL grammar, which is translated into Ecore using EMF codegen. Several DSL editors, namely textual and tree-based are generated. Other similar Eclipse EMF based textual workbenches are TCS and EMFText.

[11] http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite

[12] "Language Workbenches: The Killer-App for Domain Specific ...." 12 Jun. 2005, https://martinfowler.com/articles/languageWorkbench.html. Accessed 29 Jun. 2020.

[13] https://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html

[14] www.jetbrains.com/mps

MMS is a projectional textual workbench. This approach may offer multiple projections for the same AST. Despite the different approach, MMS offers a similar textual based authoring experience to modelers, although combined with symbolic and tabular notations. For instance, different entities in the DSL could be injected into the model by selecting them from a drop-down list of options. Unlike Eclipse-based textual workbenches that require compilation and plugin deployment to use the generated DSL editors, MMS generated editors are immediately available after reloading the changes in the DSL AST specification.

Monticore is a Java based workbench available for Eclipse and IntelliJ that, like Xtext, offers a context-free rich grammar for defining the DSL syntax, rules and terminal symbols. From the grammar definition, Monticore generates (like Xtext and similar workbenches) the lesser, parser, DSL editor and code generation (based on templates) components. Rascal, like above workbenches, offers a complete framework for DSL metaprogramming manipulation, transformation and generation of source code, with primitives located at the language itself. Spoofax is another Eclipse based textual DSL workbench, similar to others mentioned above. It includes a Spoofax meta-language and editor for declarative language definition, generators that produce lessers, parsers, syntactic validators, compilers and interpreters, DSL editors for Eclipse and IntelliJ, and programmatic API to extend the default DSL management behavior.

A key feature supported by some of these DSL textual workbenches (e.g. XText, Rascal, Spoofax) is the possibility to compose more complex DSLs by aggregating distinct, separate, simpler DSLs.

Besides textual notation, other DSL workbenches that support graphical and tabular notations are also popular, and some have evolved to high matured state in the Eclipse IDE, considered by some as the de-facto standard ecosystem for open source DSL meta-modeling and Model Driven Engineering [42]. Among those graphical DSL workbenches available in this ecosystem, we can highlight Sirius [43] and Graphiti[15], both leveraging the Eclipse Graphical Modeling Framework (GMF). Both workbenches leverage Eclipse GEF[16] and GMF[17] APIs, adopting a Model-View-Controller (MVC) strategy, to generate graphical notations and editors for DSL authoring, releasing modelers from the intrinsic programmatic complexity of these Eclipse frameworks.

Graphiti offers a simplified API, on top of GEF and GMF, that supports the definition of features, instead of adopting the MVC approach, and provides CRUD operations for the managing of the business models and the graphical elements associated with these features. However, it still remains a programmatic API, although it largely simplifies the development of graphical DSL editors compared to the low-level GEF and GMF APIs.

Sirius offers a complete workbench for DSL graphical modeling that leverages GMF. Using Sirius, modelers can design DSL meta-models and associated visual editors without any programming. Sirius supports multiple viewpoint visual representations for the same DSL meta-model. These representations can be enabled/disabled by modelers to offer different perspectives of the same model. Representations can combine graphical elements in a canvas with tables, matrices (crossed-tables) and tree-based hierarchies in separate views. Representations are designed by using a visual tree-based editor. Sirius can be customized programmatically by providing Java based extensions and Acceleo[18] queries.

[42] offers an empirical comparison of Graphiti and Sirius, concluding that Sirius is a preferable choice because if offers a similar (or even better) features set than Graphiti but supporting WYSIWYG edition of DSL visual editors, unlike the Graphiti's programmatic definition of DSL visual elements.

[15] https://www.eclipse.org/graphiti/
[16] https://www.eclipse.org/gef/
[17] https://www.eclipse.org/gmf-tooling/
[18] http://www.eclipse.org/acceleo/

A controversy among practitioners of DSL modeling is the debate about the benefits of adopting a pure textual or graphical notation for DSL model authoring. The main benefit of adopting graphical DSL notation is the effective communication of the model concepts between end-users and domain practitioners [44]. The work in [45] summarizes the main benefits of both approaches based on previous research works, classifying them in the following:

- Textual notation benefits:
    - Analyzability of the textual model by external tools
    - Consistency checking
    - Platform and tool independent
    - Shortening learning curve as textual modeling is similar to programming
- Graphical notation benefits:
    - Representation of thoughts avoids language constructs misconceptions
    - Spatial reasoning improves model comprehensibility
    - Ease programming task through more attractive representations

It also reports the results of an empirical study where they compare textual and graphical notations on the "efficiency, effectiveness and satisfaction of software developers while performing analysability and modifiability tasks on two different applications". It concludes that "subjects in the experiment performed significantly better both for analysability coverage and modifiability efficiency with a textual notation, while none of the measures was significantly in favor of graphical notations. Despite this, subjects showed a slight preference towards the graphical notation of the domain models used in the experiment".

2.2.2 Authoring tools for cloud applications

Several modeling languages have been proposed in recent years for supporting the specification of complex application topologies and their deployment into a variety of infrastructures, including Cloud [46][47][48][49] and HPC [50][51]. The Topology and Orchestration Specification for Cloud Applications, TOSCA [46] can be considered as the de-facto standard in the Cloud realm. TOSCA encompasses in a single language, terminology to define the topologies of both infrastructures and their resources, on the one hand, and an orchestration of application components on the other. Moreover, although it standardizes a textual notation based on YAML[19], it lacks visual notation, leading to the appearance of different authoring tools that use their own non-standardized visual notation. In order to address this limitation, some researchers have designed their proposals to standardize the visual notation for TOSCA.

Vino4TOSCA [52] provides a visual notation that focuses on improving the human-centric communication of cloud topologies and orchestrations. It partially covers the TOSCA specification by focusing on the modeling of topologies of resources with node templates, relationship templates and groups, leaving apart the specification of types. A Web tool prototype for Vino4TOSCA visual modeling, named Valesca[20] was implemented in the project CloudCycle.

Vinery [53] is a Web TOSCA visual editor, also developed in CloudCycle, which can be opened in a Web browser, but it is also included, as a plugin, within the Eclipse and IntelliJ IDEs. Vinery separates modeling concerns. On the one hand, it supports resource experts on the specification of TOSCA types and their management, using the Element Manager. On the other hand, it supports application owners on the definition of their application topologies, using the Topology Modeler. This is done by instantiating types as application components (i.e. templates) and the relationships among them. Aiming at facilitating the collaboration, all defined elements (types, templates, relationships, topologies, etc) are shareable within a common repository, which can be

[19] "YAML Ain't Markup Language (YAML™) Version 1.2 - YAML.org." 1 Oct. 2009, https://yaml.org/spec/1.2/spec.html. Accessed 29 Jun. 2020.
[20] http://www.cloudcycle.org/valesca

publicly exposed on the Web. Created topologies are exportable into CSAR files for deployment into a TOSCA-compliant runtime environment. Experts can populate the repository with new types defined with the Element Manager or imported from existing CSAR files.

The SIDE Workbench [54] developed in the project SWITCH is a Web graphical editor of Cloud applications that enables i) app owners to design their application topologies as compositions of components instantiated from a palette of component types that are retrieved from a common repository, ii) experts to define the abstract infrastructure. Resulting specifications are exported into TOSCA blueprints whose compliance is verified using semi-formal models and reasoners. SIDE Workbench imposes a predefined schema for component specification that constrains its usage within the SWITCH environment.

CloudCAMP DSML [48] supports the generation of IaC deployment models from users' abstract business-oriented requirements. It offers a Web-based editor, leveraging WebGME[21], for the creation of application component topologies by utilizing TOSCA node templates and relationships. These topologies are exported as Ansible scripts, using a MDE approach for code generation that relies on above DSML and then deployed into the IaC environment. CloudCAMP incorporates in its DSML the available node and relationship types that are supported to instantiate application components.

Alien4Cloud [22] offers a Topology Editor, a Web-based editor, which enables application owners to design the deployment topology of their applications as an orchestration of components instantiated from types retrieved from a common TOSCA Catalogue. This catalogue contains types (for nodes, relationships, artifacts, etc), imported from TOSCA descriptors, and topology, which can be reused to create tailored application application topologies. Types and topologies can be clustered into workspaces for restricting their access to authorized users. Types can be graphically modeled using form-based editors. Application topologies can also be graphically modeled in the Topology Editor using a canvas where to drag and drop types from the Catalogue palette.

Ubuntu Juju[23] offers a framework designing the deployment topology of applications into the Cloud. It offers a graphical Web editor for building topology models and a repository, called Charms Store, that contains buy-per-use reusable components called Charms for a variety of use cases. Juju exports topology models into YAML documents called bundles, which can be used in a IaC. However, Juju's bundles are particular to Juju's own orchestrator, and not based on the portable and orchestration-neutral TOSCA language.

2.3 SODALITE Innovation

Following the literature review presented in the previous section, we describe in this section the key innovation and research contributions of our work compared to the state of the art. The technologies adopted and reused to implement the innovative features are presented in Section 6, where we describe the baseline technologies of WP3.

2.3.1 Ontologies

Despite the growing interest in ontology-based solutions for cloud environments, comparatively little focus has been given on:

- Provisioning of modular and reusable ontological components, following best practices in ontology engineering, e.g. Ontology Design Patterns (ODPs) [55]: ODPs constitute small, modular, and reusable solutions to recurrent modelling problems, providing a consistent way for developing ontologies. From an ontology engineering perspective, this modular knowledge can be maintained, comprehended and reasoned over more easily, leading to self-contained, independent and reusable knowledge components, especially in domains

[21] https://webgme.org/

[22] ATOS, "Alien4Cloud 1.1 overview, "December 2017

[23] https://jaas.ai/

where we need to encapsulate different levels of abstractions, e.g. normative types, resources and applications in TOSCA.
- Taking full benefit of the expressiveness and conceptual modelling capabilities of OWL 2, e.g. punning [56][57]: In many domains, there is a need to encapsulate the different levels of abstractions, having entities that play more than one role (e.g. a particular concept should be viewed as a class in one role but as an instance in another role). OWL 2 supports punning (meta-modelling) that allows for using the same identifier, e.g. for an individual and a class. In TOSCA, for instance, nodes need to be treated both as classes and as individuals: in the former case, by capturing nodes as classes, we can reuse the ontology semantics regarding subsumption hierarchies. However, descriptive information regarding nodes, e.g. capabilities and interfaces can be captured only by treating nodes as instances as well.

In SODALITE we propose an ontology-based framework for capturing and interlinking TOSCA-based descriptions of cloud applications and resources. Our approach performs a conceptual lifting of TOSCA meta-model to the Descriptions and Situations pattern (DnS), espousing meta-modelling to handle the specifics of the standard in a multi-tier manner. As we describe in Section 3, our approach captures information in a conceptually uniform manner across three tiers: the descriptive context of node types (Tiers 0 and 1) and node templates (Tier 2). In line with the TOSCA meta-model that defines multi-level concept-object hierarchies (i.e. normative types can be considered as schema types, but they are also instantiated to attach annotations, therefore they are handled as instances), we start from concept hierarchies (Tier 0), where classes obtain descriptive context through property assertions. The concept hierarchy is further extended (Tier 1) and so does the meta-model space where custom resources are treated as instances as well. Finally, Tier 2 contains only instances, closing this alternation of schema and instance definitions.

By implementing the DnS pattern, we ensure the provision of modularised foundational ontology fragments as a reference base implementation (in the form of an OWL 2 building blocks), ready to be customised in TOSCA-based environments. At the same time, the use of meta-modelling allows us to represent contextualised views on complex situations (nodes and templates), affording reusable pieces of knowledge that cannot otherwise be expressed by the standard ontology modelling, e.g. multi-level concept-object hierarchies.

The innovation and contribution of our research can be summarised in the following:
- We propose an ODP-based abstraction layer for TOSCA-compliant descriptions of applications and resources, offering a common modelling strategy for one of the fastest growing standards in OASIS.
- We enhance the proposed conceptual lifting with meta-modelling, fostering the reuse and extension of TOSCA entities in different levels of abstraction (i.e. TOSCA normative types, resource and application models).
- We demonstrate the use of emerging W3C standards for enrichment and validation of TOSCA-compliant Knowledge Graphs.

2.3.2 IDE

SODALITE IDE supports the specification of application deployment topologies, and the resources the application requires on the target infrastructure, ranging from Cloud to HPC, as model instances of the SODALITE DSL. This DSL has been designed as an abstraction that leverages TOSCA to facilitate the export of AADM topologies as TOSCA blueprints into the SODALITE IaC environment (D4.1 IaC Management). As TOSCA is a vast modeling language that shows significant complexity for modelers, the SODALITE DSL has been conceived to reduce this complexity, by adopting design and implementation decisions explained in the following.

The DSL has been designed as a group of interlinkable meta-models that cover different modeling concerns, namely:

- the modeling of application topologies
- the modeling of application optimizations, and
- the modeling of infrastructure resources,

for applications and resources of both Cloud and HPC domains.

Application Ops Experts (AOEs) tackle the modeling of application topologies as model instances of the Abstract Application Deployment Meta-model (AADM). This meta-model defines classes for describing application components, their requirements and relationships, which can be mapped to TOSCA nodes, requirements and relationship templates, respectively.

Optimization Experts (OEs) tackle the design of deployment optimizations for components in AADMs, as instances of the Optimization Meta-model (OM). This meta-model defines classes to apply optimization actions to selected application components, including compilation, AI training and HPC parallelisation optimizations.

Resource Experts (REs) tackle the modeling of infrastructure resources as model instances of the Resource Meta-model (RM). This meta-model defines classes for describing resource types, their capabilities, relationships and interfaces, which can be mapped to the TOSCA nodes, capabilities, relationships and interface types, respectively.

The AADM imports both the OM and the RM so that the application components in an AADM model can refer to optimization models and they can be instantiated as instances of the types defined within RM models. This split of meta-models for the different modeling domains permits these three roles to focus only on the modeling of a particular concern.

Separation of modeling concerns is also a strategy adopted in other related works, specially in Vinery, but also in SIDE Workbench and in Alien4Cloud. SODALITE goes beyond these approaches, specifically designed for Cloud deployment, by extending the modeling support to the HPC domain, and by incorporating specific modeling assistance for the optimization of application deployments.

Authoring DSL models is being supported by the SODALITE IDE, based on Eclipse, and assisted by the semantic repository, namely the SODALITE semantic Knowledge Base (KB). The IDE offers textual and graphical editors for creating AADM models, and textual editors for OM and RM models [24].

Textual editors for AADM, OM and RMs models are intended for skilled modelers that require fast modeling.

Graphical views, tabular and form-based editors for AADM are intended for modelers that prefer a visual modeling in a canvas by dragging and dropping entities from a palette, but also for textual modelers that value visual representations of the AADM models as an improved communication and knowledge exchange channel. They are synchronized with the textual edition so that changes in the textual model are immediately reflected in the graphical views and the form-based property views. Viceversa, changes in the graphical views and/or the form-based editors are reflected in the textual editor. Multiple graphical views of the same DSL model are possible. SODALITE will offer an AADM visual representation that resembles the TOSCA visual notation[25], and another specific one for workflow oriented notation. Graphical modeling of application topologies and infrastructure resources is also supported by other related works, notably by Winery, SIDE Workbench, CloudCAMP DSML, Alien4Cloud, and Jiji. However, they offer pure graphical editors, combining graph, tabular and form-based modeling, but not textual, as Sodalite does, which could offer faster modeling to skilled developers. Moreover, SODALITE combines synchronised textual and

[24] Visual editors are planned pending on the availability of resources for their implementation

[25]https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/os/TOSCA-Simple-Profile-YAML-v1.2-os.pdf

graphical modeling of AADMs, and it offers different viewpoint notations for the same AADM model. SODALITE adopts a baseline technology (i.e. Sirius) that facilitates the creation of multiple viewpoint representations of the same model in order to express different facets.

Some related works also support the import of TOSCA YAML blueprints for populating infrastructure resources into their repositories. SODALITE goes beyond providing a simplified DSL for RM authoring that releases the REs from the complexity of the TOSCA YAML specification.

SODALITE also innovates in the modeling aids offered to the AOEs and REs. Editors provide context-aware intelligent content-assistance to guide modelers through the grammar of the SODALITE meta-models, suggesting possible elements to incorporate to the model at the point of typing. The IDE only requires from AOEs and RMs the provision of the mandatory information they own about the application topology and/or resources, and relies on the shared knowledge contained within the KB to complete the model both at the design and deployment time. The IDE also exploits the KB to assist modelers during the authoring of AADM or RM models, by suggesting them, using context-aware intelligent content-assistance, suitable choices to fulfil certain model entities, including the overriding of properties inherited from supertypes, the resolution of mandatory requirements, the proper selection of component types, etc. AADM and RM models are stored into the SODALITE KB to be reused and shared with other modelers.

The IDE textual editors conduct syntactic validation (i.e. to ensure DSL compliance) during the modeling phase. Semantic validation is also conducted at the time the models are stored into the KB or deployed into the SODALITE IaC environment, by applying deep inference reasoning on the shared knowledge that are referenced in the models. Although conformal and semantic validation is supported in some related works, notably in the SIDE Workbench, to our knowledge, they are not exploiting the reasoning and inference capabilities of a semantic KB. Moreover, SODALITE innovates by extending the validation of the optimization of AADM models and the detection of possible technical defects within them, which is conducted by the SODALITE optimization sub-system. All detected errors and warnings are presented to the user in the IDE, close to the affected component. If quick fixes are suggested by the validation process as well, they are presented to the user, and applied upon selection.

3 SODALITE Conceptual Models

This section describes in detail the conceptual model and modelling decisions that have been taken to implement the ontology-based semantic abstraction layer of SODALITE, which involves the capturing of resources and instantiations of custom and reusable patterns (AADMs). The former are specified by the Resource Experts, while the latter by the Application Ops Experts (see D2.1, Section 2.2. “Actors and use cases”). To this end, the current version of the semantic models of SODALITE includes:

- The **SODALITE meta-model**, i.e. the formal ontology pattern to use in order to capture information on different levels of abstraction.
- The **domain ontology** that provides the vocabulary to capture information in the two modelling layers (tiers) considered in SODALITE, namely Tier1 (resources) and Tier2 (pattern instantiations).

SODALITE capitalises on and combines existing Semantic Web standards and best practises for ontology development. To this end, the ontologies have been implemented in OWL 2 [7], the officially recommended language by W3C for knowledge representation in the Semantic Web. In addition, a key design choice underpinning the engineering of the SODALITE conceptual models has been the adherence to an Ontology Design Pattern (ODP) approach. The aim is to package commonly recurring ontology features as small and reusable building blocks, to be reused by ontology engineers in development. These building blocks emphasise the reusability of the developed domain model, rather than the technical specifics and they can assist in ontology engineering in two ways [1]:

1. By reducing the amount of modelling work needed for implementing common features.
2. By promoting the encoding and reuse of best practice solutions to common modelling problems.

3.1 Tiers

SODALITE follows a modular, 3-tier approach to capture knowledge (Figure 6):

- **Tier 0**: This tier provides the basic TOSCA meta-model, i.e. the representation of TOSCA meta-model in the conceptual model of SODALITE ODP. These are generic types of resources, capabilities, properties, etc. This part is the static schema of the ontology/KB.
- **Tier 1**: Involves the instances that are created by Resource Experts, e.g. custom resource types, their capabilities, associations, etc.
- **Tier 2**: These are the instances that are combined into “patterns” or “templates”, which are reusable combinations of Tier 1 types. These patterns will be created by Application Ops Experts using the DSL in the SODALITE IDE. These instances also define the Abstract Application Deployment Model - AADM.

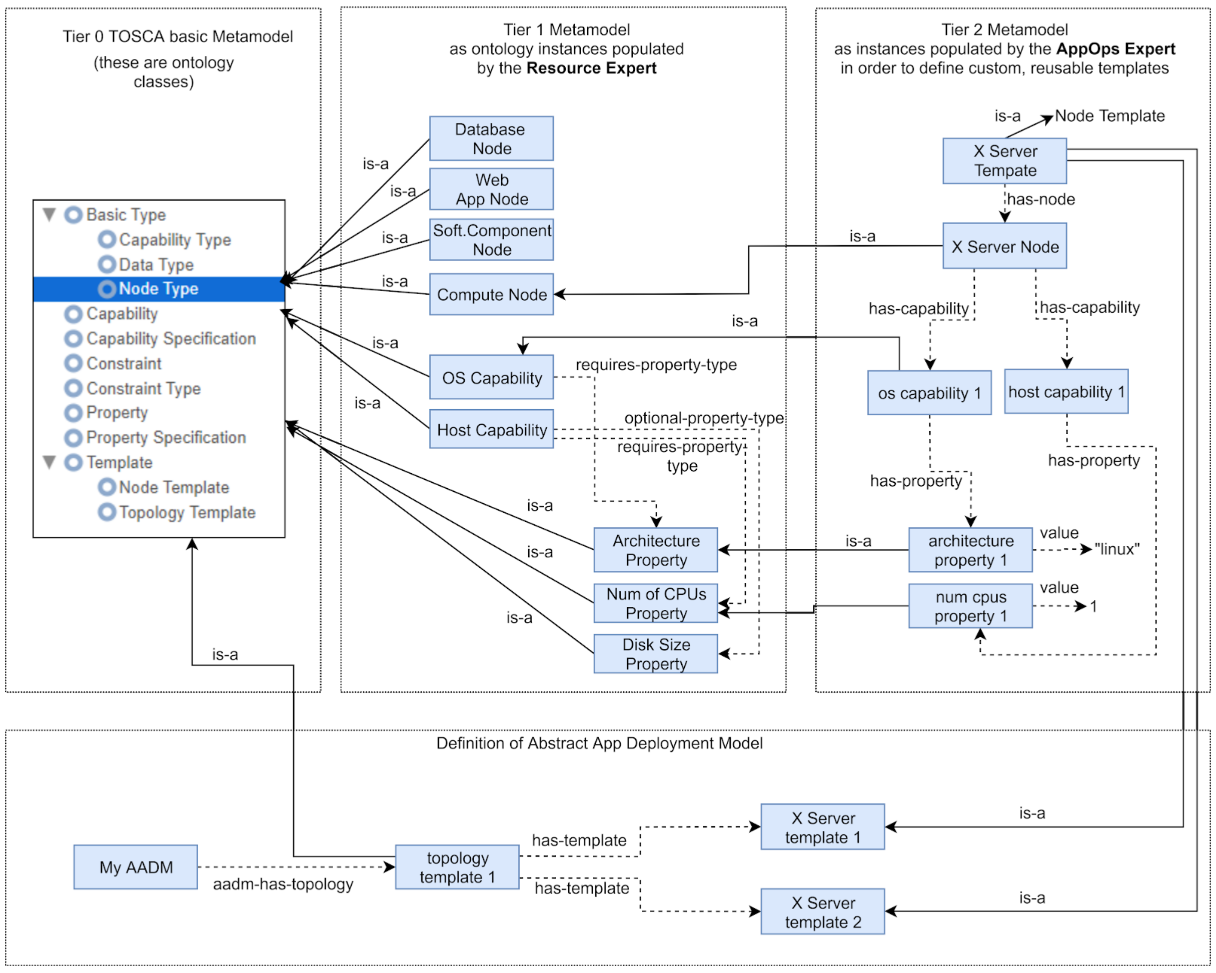


Figure 3. Overview of the SODALITE modelling layers

From an ontology engineering perspective, this modular knowledge representation framework can be maintained, comprehended and reasoned over more easily, leading to self-contained, independent and reusable knowledge components.

The SODALITE conceptual model consists of two core ontologies that are used to capture knowledge in the three tiers:

1. **SODALITE Ontology Design Pattern (ODP)**: It is the core meta-model (reusing the Descriptions and Situations pattern) that describes the way knowledge should be structured in modular, self-contained and reusable modules. This ODP is used to capture knowledge in all tiers (Tiers 0, 1, 2).
2. **TOSCA ontology**: It consists of several instantiations of the SODALITE ODP in order to capture knowledge in Tier 0 (static knowledge about TOSCA normatives).

Tier 1 and Tier 2 are the two dynamic tiers in the sense that their content is derived by mapping input provided by the users (Application Ops and Resource Experts) through the IDE. In all tiers, the SODALITE ODP is used, fostering a unified knowledge representation paradigm that enables the harnessing of potentials of a complete and unified metadata model with dedicated features, such as interoperability, cross-database search and smooth knowledge management.

3.1.1 Descriptions and Situations Pattern (DnS)

To promote a well-defined description and achieve a better degree of knowledge sharing and reuse, the SODALITE ODP is defined as a specialised instantiation of the Descriptions and Situations (DnS) ontology pattern that is part of DOLCE+DnS Ultralite (DUL).

The Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) focuses on cognitive issues and it aims at capturing the ontological categories underlying natural language and human common sense. DnS (Descriptions and Situations) enhances DOLCE's descriptive characteristics even further allowing the context-sensitive "redescriptions" of the types and relations postulated by other given ontologies or ground vocabularies. The current OWL encoding of DnS assumes DOLCE as a ground top-level vocabulary. The DOLCE+DnS Ultralite (abbreviated as DUL) is a light version, which provides simplifications and improvements of some parts of DOLCE and DnS. Its purpose is to provide a set of upper level concepts that can be the basis for easier interoperability among many middle and lower level ontologies.

DnS tries to capture the notion of "Situation" out of a state of affairs, with their interpretation being provided by a "Description":

- **Situation**: a set of domain entities that are involved in a specific pattern instantiation.
- **Description**: serves as the descriptive context of a situation, defining the concepts that classify the domain entities of a specific pattern instantiation, creating views on situations.
- **Concepts and parameters**: Classify domain entities describing the way they should be interpreted in a particular situation. Each concept may refer to one or more *parameters*, allowing the enrichment of concepts with additional descriptive context.

The basic implementation of the DnS pattern in DUL allows the relation of situations (dul:Situation) and descriptions (dul:Description) with domain concepts (dul:Concept) and participants. More specifically, a situation describes the entities of a context, e.g. the components that are involved, and satisfies (dul:satisfies) a description. The description in turn defines (dul:defines) concepts that classify (dul:classifies) the entities of the situation, describing the way they should be interpreted. Each concept may have one or more parameters (dul:hasParameter). Figure 4 presents the DnS core pattern.

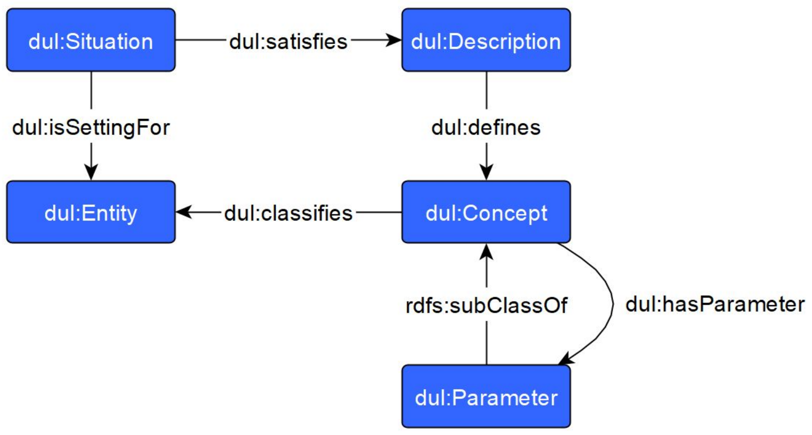


Figure 4. Core DnS pattern in DUL

3.2 SODALITE Meta-Model

The SODALITE meta-model extends the core DnS pattern by defining proper specialisations of the core DUL concepts to model TOSCA definitions (Figure 5). More specifically:

- **SodaliteSituation**: Represents a situation, e.g. a node type (soda:SodaliteSituation ⊑ dul:Situation).

- **SodaliteDescription**: Each situation, e.g. node type, has a descriptive context (soda:hasContext) that describes the properties, attributes, interfaces, requirements, capabilities, etc. of the situation soda:SodaliteDescription ⊑ dul:Description.
- **SodaliteConcept and SodaliteParameters**: Each property, attribute, interface, requirement, capability, etc. has a specification (soda:specification ⊑ dul:defines) which involves one or more SodaliteConcepts with zero, one or more SodaliteParameters (soda:SodaliteConcept ⊑ dul:Concept, soda:SodaliteParameter ⊑ dul:Parameter). Each SodaliteConcept classifies one SodaliteEntity (soda:Entity ⊑ dul:Entity).

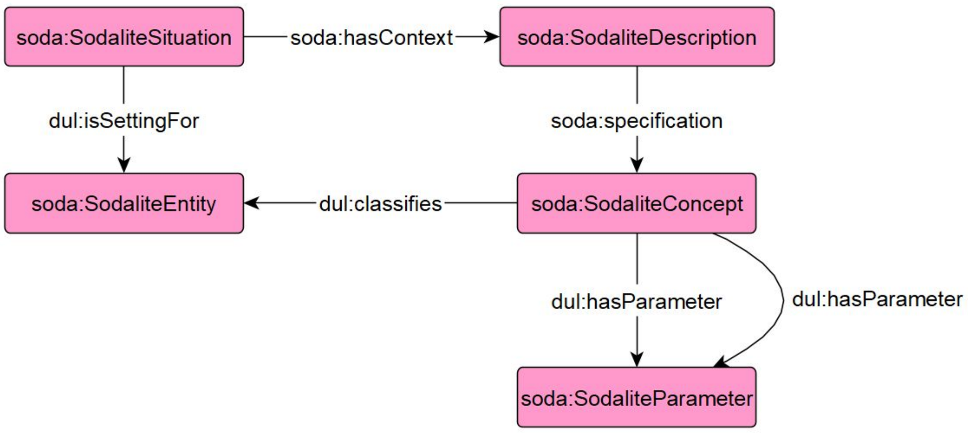


Figure 5. SODALITE meta-model (extension of DUL)

Intuitively, each TOSCA type (e.g. node, capability, etc.) is represented as a SodaliteSituation. Each such situation is associated with a context (SodaliteDescription) that defines additional descriptive information relevant to the type. For example, this description may contain definitions about relevant properties, attributes, requirements and capabilities that a node type may have. As already highlighted, the benefit of using a formal ODP to model knowledge in SODALITE is that the same conceptual model can be used in all tiers, resulting in a unified model (knowledge graphs) with increased modularity, interoperability and easier to define inference logic. In the following, we describe key modelling guidelines for capturing knowledge in different tiers, along with examples.

3.3 TOSCA Domain Ontology

SODALITE ODP is a generic ontology pattern that defines the modelling guidelines to be followed in order to capture domain knowledge. As such, a domain ontology is needed to provide the necessary vocabulary to capture context relevant to the application domain. Figure 6 depicts the domain concepts that have been defined as subclasses of soda:SodaliteConcept to be used in instantiations of the SODALITE ODP. At the same time, the TOSCA domain ontology contains the definitions of Tier 0 (TOSCA basic Meta-model). The current version of the ontology provides the modelling of TOSCA capabilities, datatypes, interfaces, nodes and relationships. Figure 7 presents an excerpt of this hierarchy.

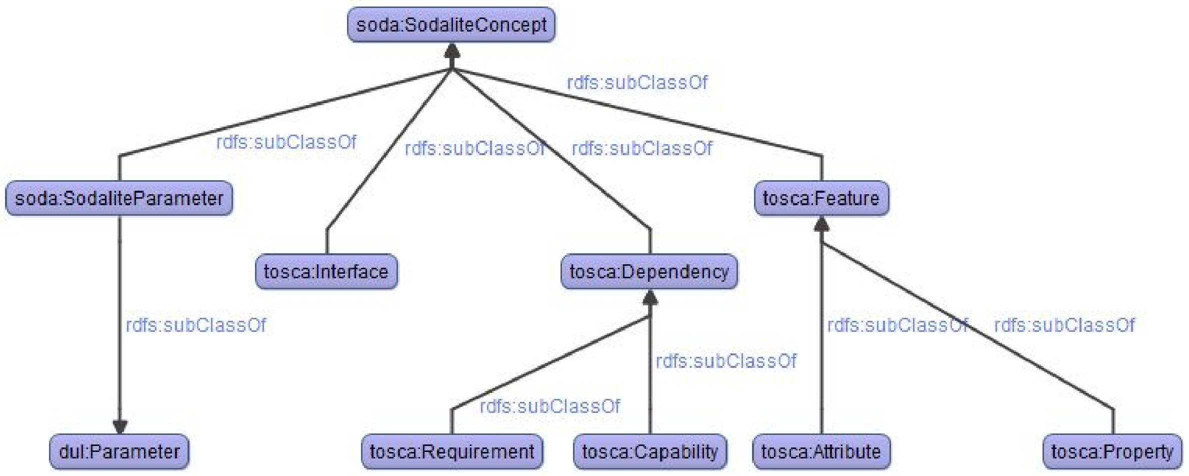


Figure 6. Hierarchy for TOSCA concepts

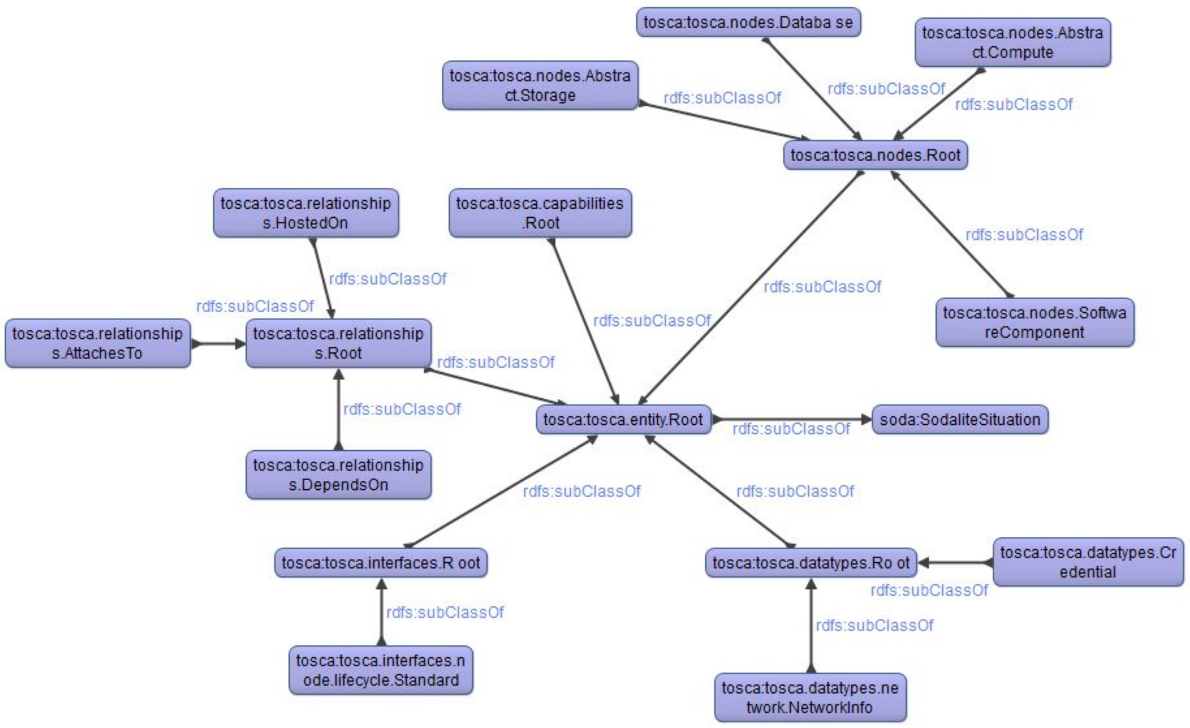

tosca:tosca.nodes.Databa se
tosca:tosca.nodes.Abstra ct.Compute
tosca:tosca.nodes.Abstra ct.Storage
tosca:tosca.nodes.Root
tosca:tosca.relationship s.HostedOn
tosca:tosca.capabilities .Root
tosca:tosca.nodes.Softwa reComponent
tosca:tosca.relationship s.AttachesTo
tosca:tosca.relationship s.Root
tosca:tosca.entity.Root
soda:SodaliteSituation
tosca:tosca.relationship s.DependsOn
tosca:tosca.interfaces.R oot
tosca:tosca.datatypes.Ro ot
tosca:tosca.datatypes.Cr edential
tosca:tosca.interfaces.n ode.lifecycle.Standard
tosca:tosca.datatypes.ne twork.NetworkInfo
rdfs:subClassOf

Figure 7. Hierarchy of Root (nodes, relationships, capabilities, etc.)

The domain ontology also defines a number of properties pertinent to modelling of TOSCA-related context. The following are key properties defined as subproperties of soda:specification, meaning that can be used in soda:SodaliteDescription instantiations:

- tosca:attributes.
- tosca:capabilities.
- tosca:interfaces.
- tosca:operations.
- tosca:properties.
- tosca:requirements.

In addition, the TOSCA domain ontology provides the vocabulary to capture the Abstract Application Deployment Model (AADM), which acts as a container for Tier 1 and Tier 2 definitions. For example, information relevant to versioning, user id submitting the model, submission time, etc. is part of this model. Figure 8 depicts the basic RDF graph schema that captures topology-related information.

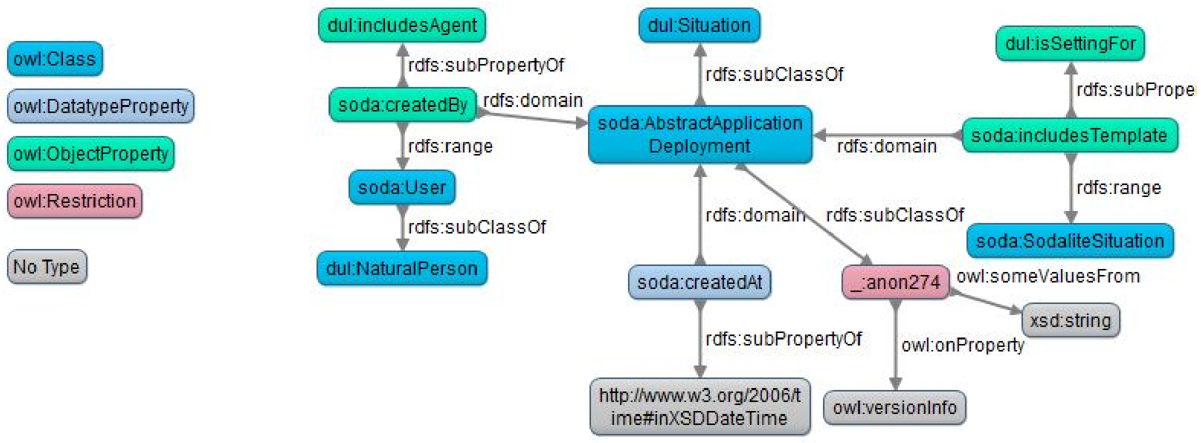
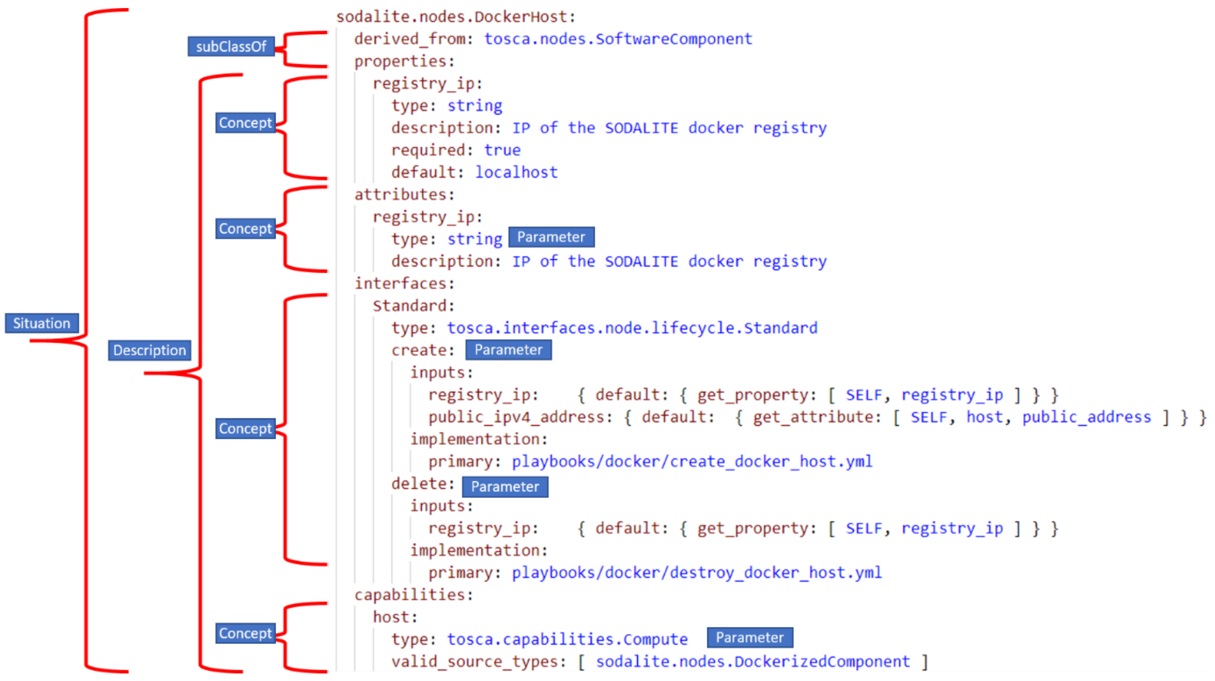


Figure 8. Overview of the AADM model to capture topologies

Example A: Node types (Tier 0 – Tier 1)

We present the capabilities of the SODALITE ODP through an example that demonstrates the instantiation of the pattern to capture a complete definition of a node type. Figure 9 depicts an example of a node type, along with the relations of the various TOSCA elements to ODP concepts. The example actually refers to Tier 1, but the same rationale is followed in Tier 0 for capturing definitions of the TOSCA meta-model (e.g. normative TOSCA node types).



```
sodalite.nodes.DockerHost:
  derived_from: tosca.nodes.SoftwareComponent
  properties:
    registry_ip:
      type: string
      description: IP of the SODALITE docker registry
      required: true
      default: localhost
  attributes:
    registry_ip:
      type: string
      description: IP of the SODALITE docker registry
  interfaces:
    Standard:
      type: tosca.interfaces.node.lifecycle.Standard
      create:
        inputs:
          registry_ip:    { default: { get_property: [ SELF, registry_ip ] } }
          public_ipv4_address: { default:  { get_attribute: [ SELF, host, public_address ] } }
        implementation:
          primary: playbooks/docker/create_docker_host.yml
      delete:
        inputs:
          registry_ip:    { default: { get_property: [ SELF, registry_ip ] } }
        implementation:
          primary: playbooks/docker/destroy_docker_host.yml
  capabilities:
    host:
      type: tosca.capabilities.Compute
      valid_source_types: [ sodalite.nodes.DockerizedComponent ]
```

Figure 9. Example TOSCA node type and high-level assignment of SODALITE ODP concepts

Attributes: Each attribute is captured as an instance of the tosca:Attribute concept that classifies the property we want to model (in this example, ex:registry_ip). For representing the description text, we use the dcterms:description property of Dublin Core[26]. Any other descriptive information is captured through definitions of one or more soda:SodaliteParameters that are associated with the instance of tosca:Attribute (Figure 10).

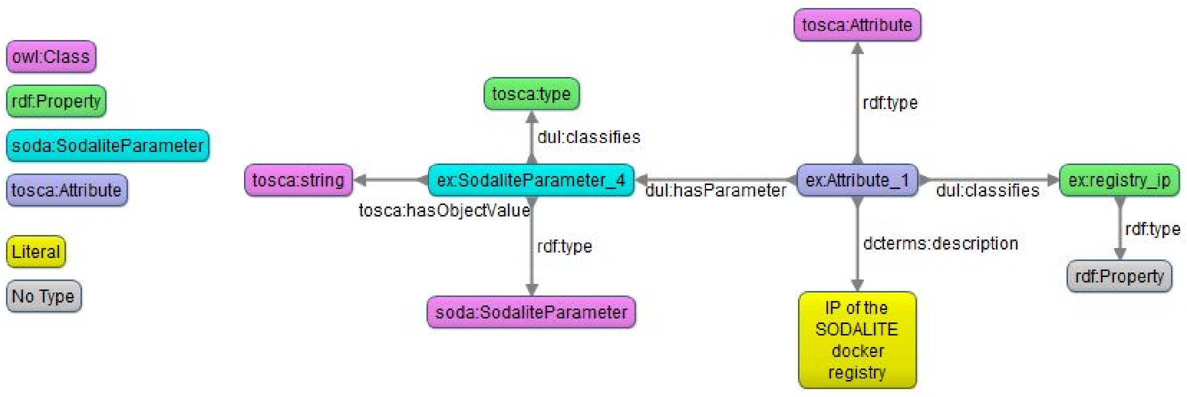


Figure 10. Example attribute definition

Properties: Properties are defined as instances of the tosca:Property concept. Similar to attribute definitions, each property instance classifies the property we want to model. Additional context is captured using dul:hasParameter property assertions, such as the fact that the property is required and that it has a default value (Figure 11).

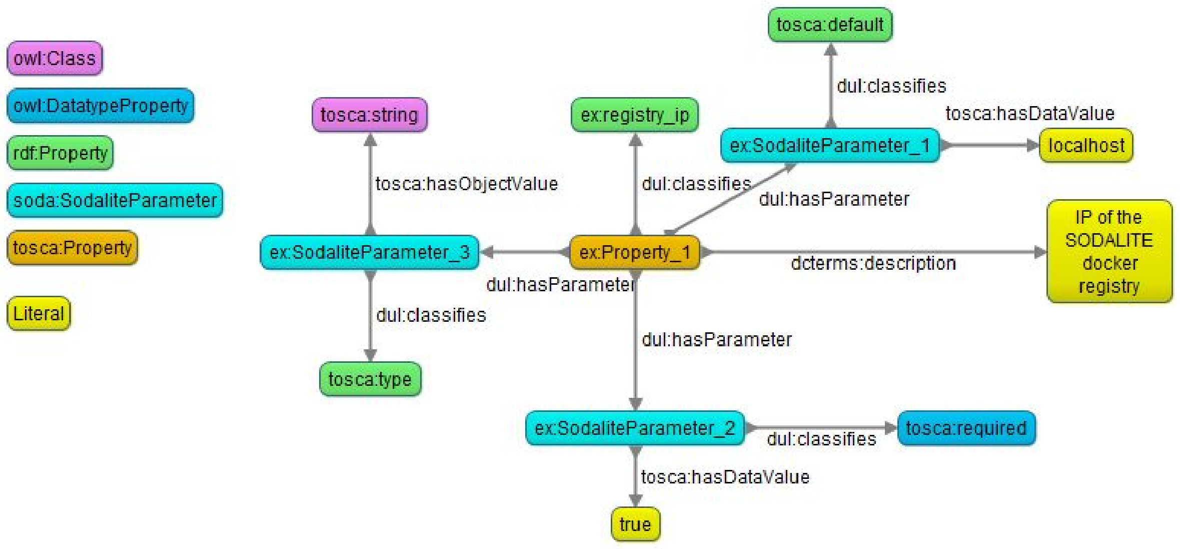


Figure 11. Example property definition

Interfaces: Interface specifications involve more complex structures with nested definitions. To this end, the generated RDF graph that corresponds to the instantiation of the SODALITE ODP becomes larger. However, the main conceptual model remains the same, i.e. definitions of concepts that classify properties and are associated with one or more parameters in order to capture additional descriptive context. Interfaces are defined as instances of the tosca:Interface concept. For readability, Figure 12 presents only a part of the generated RDF graph for the interface.

[26] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

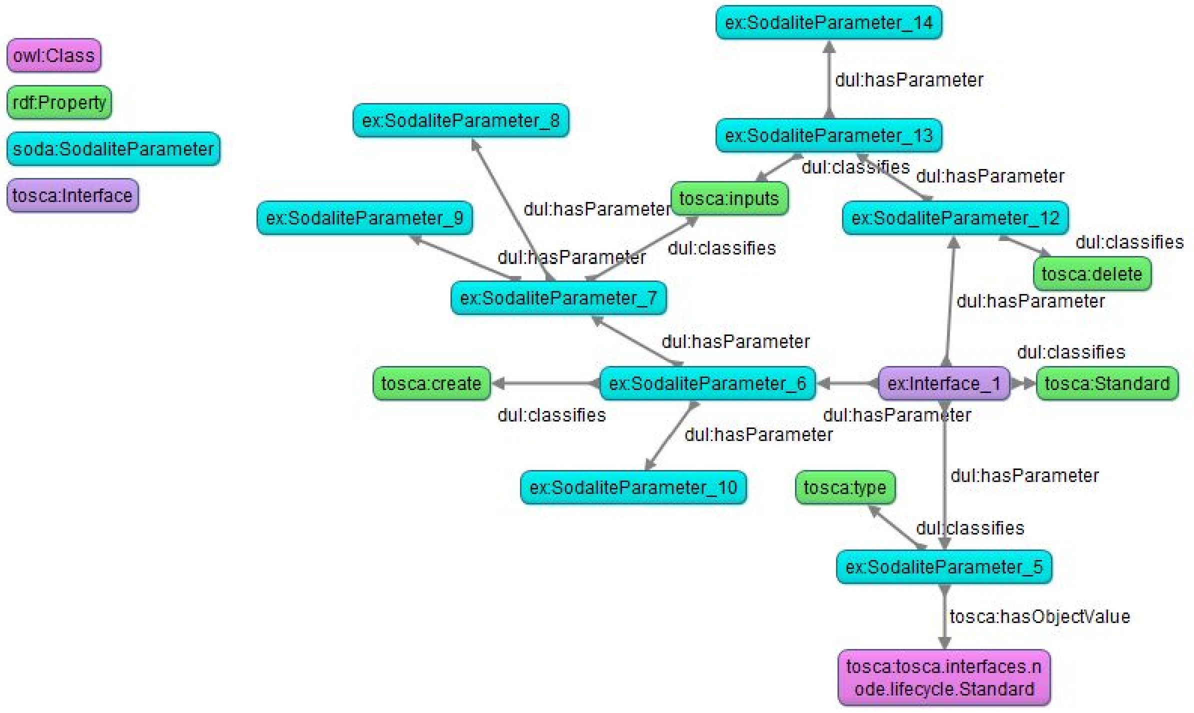


Figure 12. Example interface definition

Capabilities: Finally, the capability is defined as an instance of the tosca:Capability concept. The SODALITE ODP is still valid here (Figure 13).

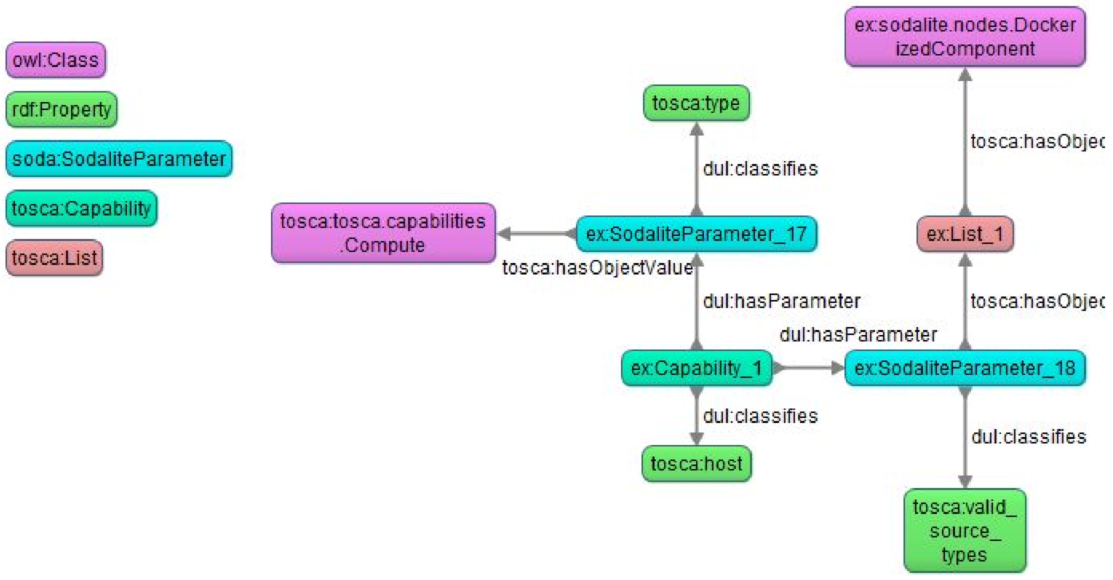


Figure 13. Example of capability definition

Descriptions and situations: So far, we elaborated on the instantiation of the SODALITE ODP to capture the context relevant to a node. As described in the previous sections, the full SODALITE ODP revolves around the notions of situations and description, following the conceptual model of the DnS pattern (in DUL). Therefore, in order to result in a conceptual valid instantiation of the DnS pattern, we also need to associate the aforementioned context (properties, attribute, interface, capability) with a description, which will then be associated with a situation. Figure 14 depicts the resulting RDF graph with the pertinent situation and description instances. It is worth noting the punning capabilities of OWL 2: sodalite.nodes.DockerHost is both a class (rdfs:subClassOf tosca.nodes.SoftwareComponent) and an instance, since it participates in the SODALITE ODP and has a context (soda:hasContext property assertion).

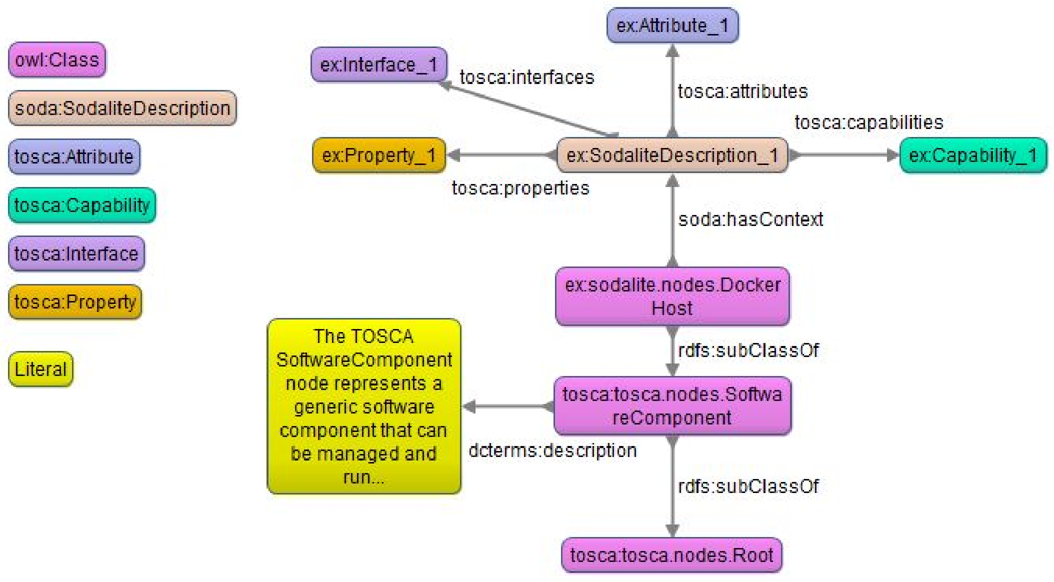
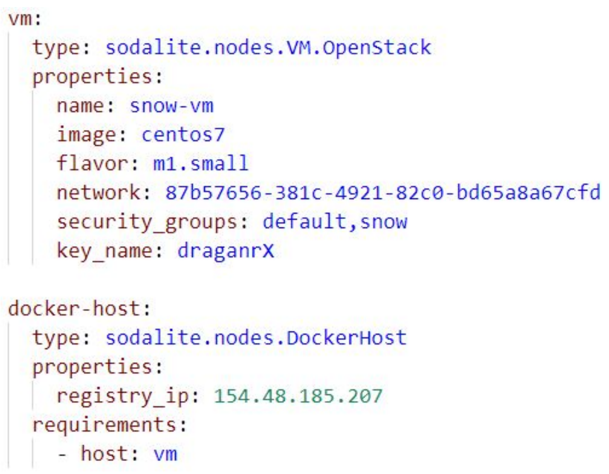


Figure 14. Example of situation and description instances

Example B: Node templates (Tier 2)

We present here an example of using the SODALITE ODP to capture the definition of a node template (Tier 2). The example considered in this section is depicted in Figure 15.



```
vm:
  type: sodalite.nodes.VM.OpenStack
  properties:
    name: snow-vm
    image: centos7
    flavor: m1.small
    network: 87b57656-381c-4921-82c0-bd65a8a67cfd
    security_groups: default,snow
    key_name: draganrX

docker-host:
  type: sodalite.nodes.DockerHost
  properties:
    registry_ip: 154.48.185.207
  requirements:
    - host: vm
```

Figure 15. Example node templates

ODP is used in the same way as we did for node types: each node template is captured as a situation that has a description. Each description describes the properties, attributes, interfaces, requirements, capabilities, etc. of the template. They are captured as concepts that classify certain properties with zero, one or more parameters. Figure 16 depicts the definition of vm and Figure 17 the definition of the docker-host node templates.

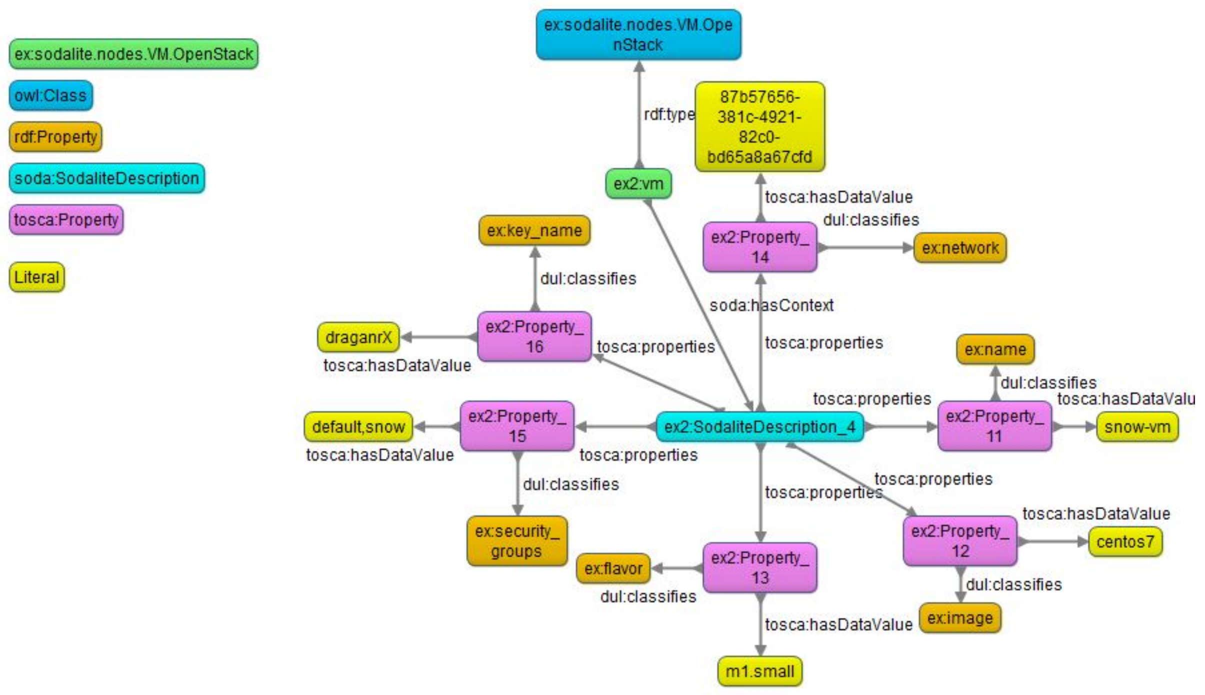


Figure 16. vm node template as instance of the sodalite.nodes.VM.OpenStack resource

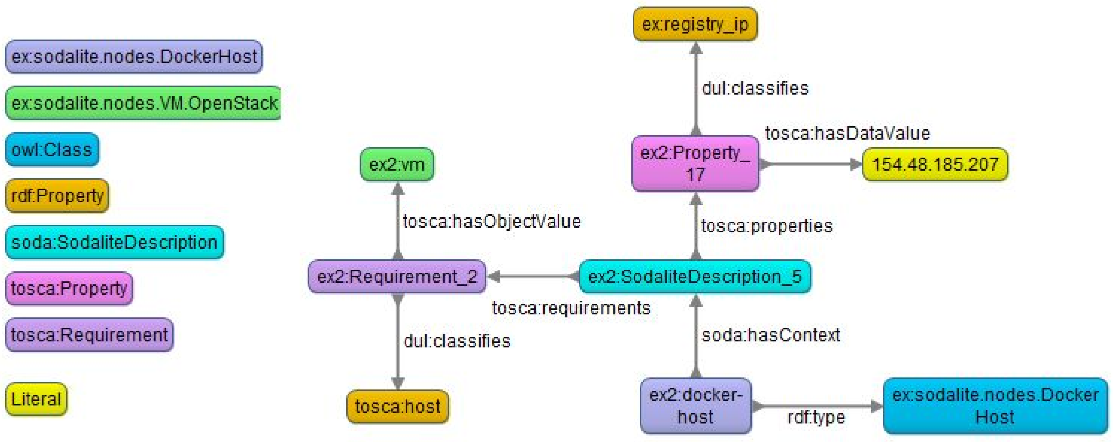


Figure 17. docker-host node template as instance of the sodalite.nodes.DockerHost resource

Topology

The way the RDF graphs of Tier 1 and Tier 2 are connected is described through container instances of the topology conceptual model. Figure 18 depicts an example AADM instance that contains references to the relevant node types and node template instances, along with additional descriptive information, such as the version, date/time of the submission, etc.

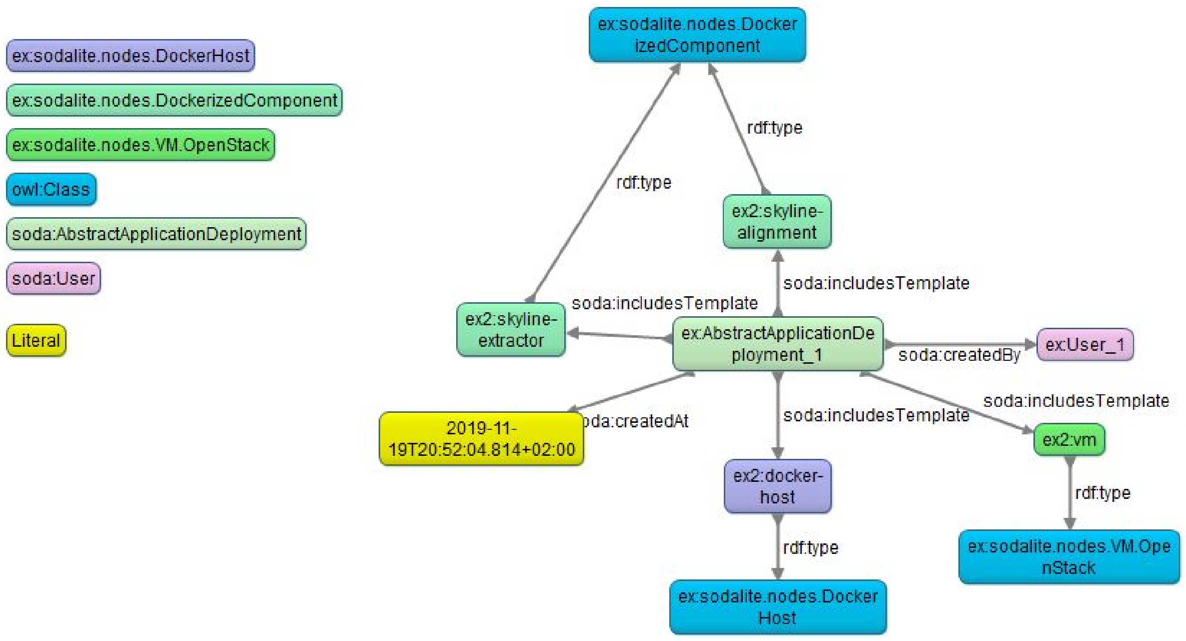

ex:sodalite.nodes.DockerHost
ex:sodalite.nodes.DockerizedComponent
ex:sodalite.nodes.VM.OpenStack
owl:Class
soda:AbstractApplicationDeployment
soda:User
Literal
ex:sodalite.nodes.DockerizedComponent
rdf:type
rdf:type
ex2:skyline-alignment
soda:includesTemplate
soda:includesTemplate
ex2:skyline-extractor
ex:AbstractApplicationDeployment_1
soda:createdBy
ex:User_1
soda:createdAt
2019-11-19T20:52:04.814+02:00
soda:includesTemplate
soda:includesTemplate
ex2:vm
ex2:docker-host
rdf:type
rdf:type
ex:sodalite.nodes.VM.OpenStack
ex:sodalite.nodes.DockerHost

Figure 18. Example topology instance

3.4 Next Steps

In the previous sections, we presented the first version of the SODALITE ODP, which has mainly been used to model node types and node templates relevant to the use case requirements. Several extensions are foreseen for future version of the ontologies, such as:

- Definition of Policies: capturing of performance and non-functional parameters of resources (e.g. QoS). As described in the innovation (Section 2.3.1), SODALITE provides modular and reusable ontological components, following best practices in ontology engineering in terms of a pattern-based design approach. Therefore, the SODALITE meta-model can be easily extended and adapted to support additional modelling requirements, like policies.
- Versioning: ability to store different versions of the same AADM model (e.g. by using named graphs). As described in the innovation (Section 2.3.1), the SODALITE meta-model is defined on top of the OWL 2 meta-modelling capabilities. By treating resources as first-class citizens, SOADALITE allows us to define different interpretations for the same resources, e.g. for the same nodes or properties. This feature can be used in order to keep different versions of the same resources, provisioning inherent support for versioning.
- Adding RDF structures to capture the content of Ansible files relevant to deployment and maintenance interfaces (create, delete, etc.).
- Use of additional W3C standards for modelling, such as the Web Annotation Data Model[27].

[27] https://www.w3.org/TR/annotation-model/

4 Ontology Population and Checking

SODALITE provides the algorithms for generating graphs following the conceptual model described in the previous section. This involves the mapping of DSL (TOSCA models) generated by users (in the IDE) to the SODALITE ontologies and the reasoning infrastructure to build the custom inferencing developed in T4.4 - Analytics and Semantic Decision Support.

As far as the population of the KB is concerned, we have defined an RDF-based exchange model between the IDE and the population engine as a lightweight version of the SODALITE meta-model. The purpose of the exchange model is, on the one hand, to hide the complexity of the conceptual model of the KB, and on the other hand, to serve as a formal, interoperable, machine readable model to exchange descriptions. WP3 is responsible for mapping this exchange model to the SODALITE ontologies, implementing the semantic mapping and structural consistency checking mechanism. In addition, WP3 is responsible for checking the consistency of the KB in terms of the native OWL 2 RL semantics. This is achieved by defining custom consistency checking logic at the level of OWL 2 semantics. It should be noted here that the semantic validation is part of advanced reasoning and decision making (WP4/T4.4). In WP3, consistency checking refers to basic restriction and constraint checking, following the semantics of the meta-model.

4.1 Exchange Model

SODALITE's KB contains both static and dynamic RDF knowledge graphs. The static knowledge graph captures information about the TOSCA specification (TOSCA meta-model / Tier 0). The other two tiers contain knowledge that is dynamically generated, based on the interaction with the IDE users and the results of SODALITE analysis components.

As described in Section 5, users define their models using the SODALITE DSL. The SPE module is responsible for mapping the DSL models to the rich conceptual model of SODALITE. In order to foster interoperability, the mapping services of SPE do not operate directly on top of the DSL, but on top of an RDF-based exchange model defined between the frontend and the backend. In the rest of this section, we briefly describe the specifics of this exchange model.

Intermediate Exchange Model

This intermediate exchange model is a lightweight version of the SODALITE ODP. The main purpose is to hide the conceptual complexity of the SODALITE ODP and to ease the interaction between the IDE and the Semantic Reasoner.

The basic schema is depicted in Figure 19. It involves:

- **Entity hierarchy**: container classes to represent node types, node templates, interfaces, relationships, requirements and capabilities.
- **Parameters**: used to define additional descriptive context for various resources
    - **Attributes**, **Properties**: referenced by entities for defining properties and attributes
    - Each parameter may have additional parameters (nesting)
- **AADM**
    - Root container to provide info about the user, version, etc.

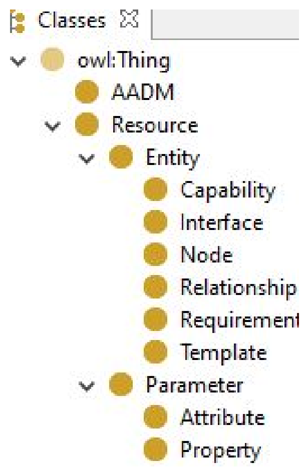


Figure 19. Basic class hierarchy of the exchange model

Apart from classes, the exchange model defines a set of properties (Figure 20). It involves:

- **attributes**, **properties**
  - Domain: Entity.
  - Range: Attribute, Property (respectively).
- **capabilities**, **interfaces**, **requirements**
  - Domain: Entity.
  - Range: Capability, Interface, Requirement (respectively).
- **hasParameter**, **value:** to associate a resource with a parameter; each parameter may have a value.
- **derivesFrom:** to define the super type (string).
- **name**, **description** (strings): for the name and description of a resource.
- **type**: for the type of templates.
- **userId**: properties of the AADM to specify the user id.

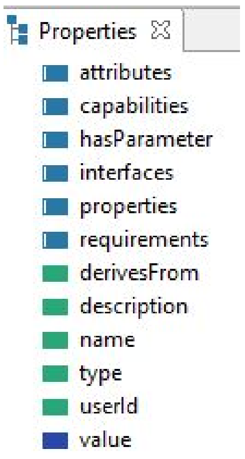


Figure 20. Basic properties provided by the exchange model

This basic schema provides all the necessary structures to translate DSL into the exchange model and to populate the KB. As an example, we present below the DSL for the node template hpc_wm_torque.

```
hpc_wm_torque{
    type: my.nodes.hpc.wm.torque
    attributes{
        public_address: "sodalite-fe.hlrs.de"
        username: "kamil"
        ssh_key: "~/keys/kamil-sodalite-fe"
    }
}
```

The representation in the exchange model is as follows (Turtle syntax[28]):

```
:Template_1
  rdf:type exchange:Template ;
  exchange:name "hpc_wm_torque" ;
  exchange:type "my.nodes.hpc.wm.torque" ;
  exchange:attributes :Attribute_1, :Attribute_2, :Attribute_3 .

:Attribute_1
  rdf:type exchange:Attribute ;
  exchange:name "public_address" ;
  exchange:value "sodalite-fe.hlrs.de" .

:Attribute_2
  rdf:type exchange:Attribute ;
  exchange:name "username" ;
  exchange:value "kamil" .

:Attribute_3
  rdf:type exchange:Attribute ;
  exchange:name "ssh_key" ;
  exchange:value "~/keys/kamil-sodalite-fe" .
```

4.2 Next Steps

The current version of the ontology population provides the necessary features to the IDE in order to assist a user in defining application and resource models. As the various SODALITE components mature and provide advanced functionalities, more complex interactions will be needed to be supported that will be implemented both in terms of the provided API and of backend reasoning services (WP4). In addition, in the first version of the framework, the automated translation of node templates is only supported. For the future, the mapping mechanism will be enriched in order to be able to map also DSL definitions about resources. Moreover, updates on the exchange model might be necessary in order to capture additional knowledge, such as versioning, the content of Ansible files (in interfaces), Docker image files, etc.

[28] Terse RDF Triple Language (Turtle) is a syntax and file format for expressing data in the Resource Description Framework (RDF) data model. Turtle syntax is similar to that of SPARQL, an RDF query language. It is a common data format for storing RDF data, along with N-Triples, JSON-LD and RDF/XML.

5 SODALITE IDE

The SODALITE IDE is an Integrated Development Environment that assists AOEs and REs in the authoring of AADMs and RMs. IDEs also offer complete assistance for the development of the user's application, so that the selection of an IDE for AADM authoring, within the same environment where the application is implemented, is a natural choice. The IDE offers a multi-view textual editor for AADM/RE definition, complemented with edit assisting features such as context-sensitive content-assistance, code completion, syntactic and semantic validation, syntax highlighting, and so on. The IDE also offers views for managing the complete life-cycle of AADM/RE models.

The IDE leverages on the remote SODALITE KB to retrieve resources to be assigned to the AADM node instances, as well as to assess the semantic validity of the model. The IDE also leverages on the IaC layer to deploy the AADM.

AADM and RM are instances of the SODALITE DSL for Abstract Application Deployment and Infrastructure Resources. A domain specific language (DSL) is a computing language designed for a specific modelling purpose within a concrete application domain. In SODALITE, DSLs are specified as meta-models. A meta-models is, in this context, a modelling conceptualization schema (and the rules and constraints that determine how it can be applied) designed to create model instances that are compliant with that schema.

5.1 Domain Specific Language

SODALITE defines two DSLs for the specification of AADMs and RMs. Both DSLs are based on the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA)[29].

The **AADM DSL** captures the essential data included in TOSCA to define abstract application models. This includes the relevant application-owner-centric information required to define application components for deployment, which matches the TOSCA nodes templates and their nested elements: properties, attributes and requirements. AADM DSL references node types (and other TOSCA type specifications) defined within the RM DSL. Moreover, application deployment nodes within the AADM DSL refer to infrastructure resources, available within the KB, which have been defined as node types, using the RM DSL.

The **RM DSL** captures the essential data included in TOSCA to define infrastructure and platform resources for Cloud and HPC; other platforms (e.g. Kubernetes will be supported in future releases of the SODALITE framework). This includes the relevant resource-owner-centric information required to define reusable resources, which matches the TOSCA node type and their nested elements: properties, attributes, interfaces, capabilities and requirements. Other types can be defined, including: data, relationship and policy types.

Both AADM and RM DSL are designed to collect from users the minimum set of deployment topology information needed to synthesize the target TOSCA blueprint, upon its deployment into the IaC layer. Remaining information required to complete the blueprint is obtained from the KB by applying its inference and reasoning capabilities and/or derived by the IaC layer from internal heuristic knowledge. This approach would largely simplify for AOEs and REs the authoring of the abstract application deployment and the resource models, respectively, reducing the cost in fixing potential application deployment modeling errors and in reducing the total application deployment time. In the following paragraphs, we provide additional technical details about the specification and implementation of both DSLs.

5.2 Supported Features

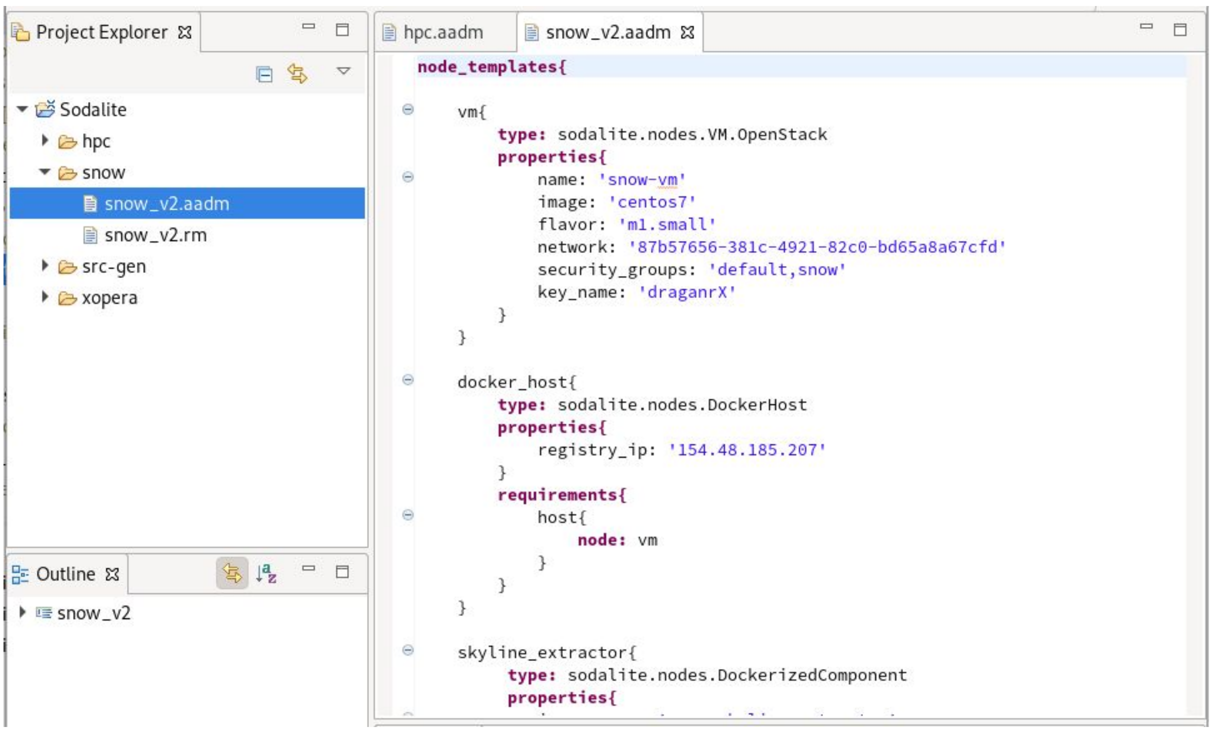
The features that are currently supported are as follows:

[29] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

DSL specification: current version provides a grammar for both the AADM and the RM. They are simplified versions of the TOSCA specification for node templates. They include the required modelling elements to fully specify an AADM for the Snow and HPC use cases.

DSL editor: current version implements the following features (see section above):

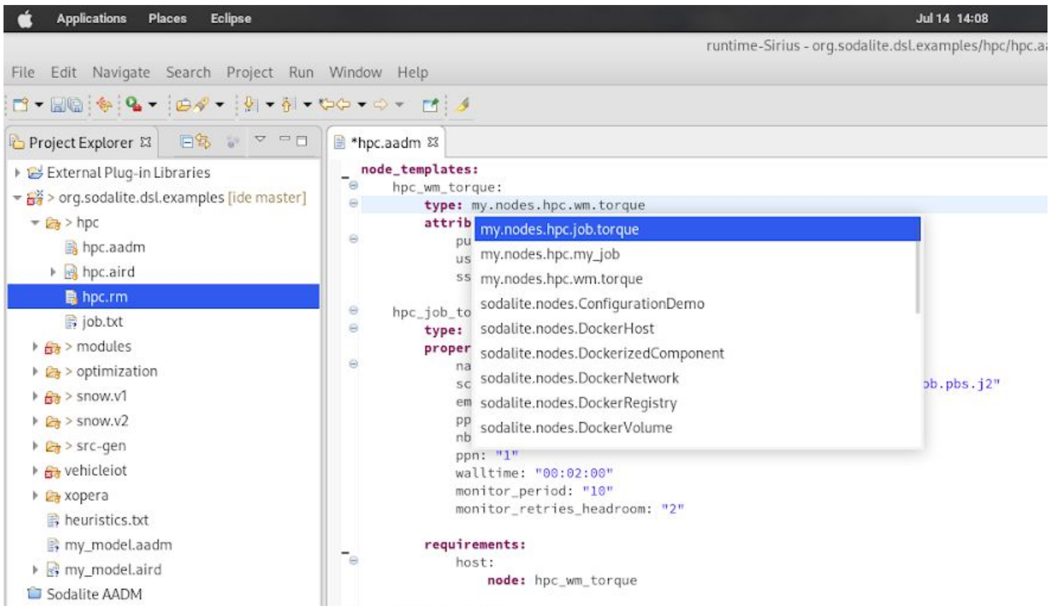
- Modelling support for designing AADMs (see Figure 21): current version supports the modelling of application node instances (e.g. node templates), their type, properties, attributes and requirements.
- Modelling support for designing RMs: current version supports the modelling of data types, artifact types, capability types, interface types, relationship_types, node_types and policy types.



```
node_templates{

    vm{
        type: sodalite.nodes.VM.OpenStack
        properties{
            name: 'snow-vm'
            image: 'centos7'
            flavor: 'm1.small'
            network: '87b57656-381c-4921-82c0-bd65a8a67cfd'
            security_groups: 'default,snow'
            key_name: 'draganrX'
        }
    }

    docker_host{
        type: sodalite.nodes.DockerHost
        properties{
            registry_ip: '154.48.185.207'
        }
        requirements{
            host{
                node: vm
            }
        }
    }

    skyline_extractor{
         type: sodalite.nodes.DockerizedComponent
         properties{
```

Figure 21. Snow UC AADM in SODALITE IDE Editor

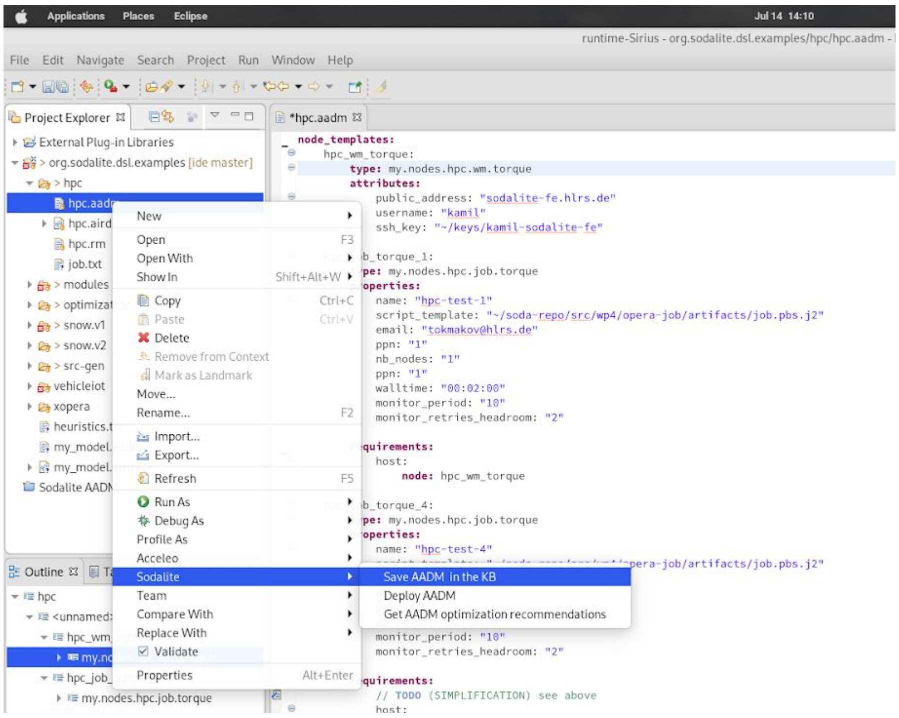
- Textual Editor for AADMs:
    - Creation, edition, and deletion of AADMs.
    - Editing facilities: grammar highlighting, code completion.
    - Context-based content assistance:
        - content assistance based on grammar constraints and relationships: entity cross-resolution.
        - content assistance based on SODALITE KB support (see Figure 22).
- Textual Editor for RMs:
    - Creation, edition, and deletion of RMs,
    - Edition facilities: grammar highlighting, code completion.
    - Context-based content assistance:
        - content assistance based on grammar constraints and relationships: entity cross-resolution.


Applications
Places
Eclipse
Jul 14 14:08
runtime-Sirius - org.sodalite.dsl.examples/hpc/hpc.a
File Edit Navigate Search Project Run Window Help
Project Explorer
External Plug-in Libraries
> org.sodalite.dsl.examples [ide master]
> hpc
hpc.aadm
hpc.aird
hpc.rm
job.txt
> modules
> optimization
> snow.v1
> snow.v2
> src-gen
vehicleiot
xopera
heuristics.txt
my_model.aadm
my_model.aird
Sodalite AADM
*hpc.aadm
my.nodes.hpc.job.torque
my.nodes.hpc.my_job
my.nodes.hpc.wm.torque
sodalite.nodes.ConfigurationDemo
sodalite.nodes.DockerHost
sodalite.nodes.DockerizedComponent
sodalite.nodes.DockerNetwork
sodalite.nodes.DockerRegistry
sodalite.nodes.DockerVolume

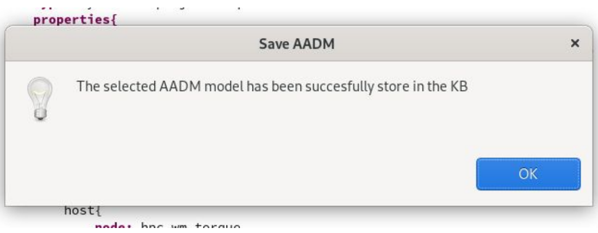
```
node_templates:
    hpc_wm_torque:
        type: my.nodes.hpc.wm.torque
        attrib
            pu
            us
            ss

    hpc_job_to
        type:
        proper
            na
            sc                                                    ob.pbs.j2"
            em
            pp
            nb
            ppn: "1"
            walltime: "00:02:00"
            monitor_period: "10"
            monitor_retries_headroom: "2"

        requirements:
            host:
                node: hpc_wm_torque
```

Figure 22. Context-sensitive content assistance

- Automatic serialization of AADMs into Turtle format for storage into the KB.
- Connectivity between the SODALITE IDE and the KB:
    - for querying that provides support in content-assistance.
    - for storing AADMs into the KB.
- Outline view for both AADMs and RMs.
- Storage of AADMs into the KB (see Figure 23).



```
node_templates:
    hpc_wm_torque:
        type: my.nodes.hpc.wm.torque
        attributes:
            public_address: "sodalite-fe.hlrs.de"
            username: "kamil"
            ssh_key: "~/keys/kamil-sodalite-fe"

    b_torque_1:
        pe: my.nodes.hpc.job.torque
        operties:
            name: "hpc-test-1"
            script_template: "~/soda-repo/src/wp4/opera-job/artifacts/job.pbs.j2"
            email: "tokmakov@hlrs.de"
            ppn: "1"
            nb_nodes: "1"
            ppn: "1"
            walltime: "00:02:00"
            monitor_period: "10"
            monitor_retries_headroom: "2"

        quirements:
            host:
                node: hpc_wm_torque

    b_torque_4:
        pe: my.nodes.hpc.job.torque
        operties:
            name: "hpc-test-4"
                                               pera-job/artifacts/job.pbs.j2"


            monitor_period: "10"
            monitor_retries_headroom: "2"

        quirements:
            // TODO (SIMPLIFICATION) see above
            host:
```



Save AADM

The selected AADM model has been succesfully store in the KB

Figure 23. Storing AADM into the KB

5.3 Next Steps

The next development period will focus on the development of the IDE features that simplifies the authoring and management of AADM and RMs, including improvements in KB enabling assistance, AADM viewpoint visualization, optimization modeling, dashboards for modeling and deployment management, and more:

AADM DSL:

- Extended support for designing AADM, incorporating additional modeling elements for TOSCA instance entities (e.g. templates) not supported in the first version of this DSL.
- Refactoring of the AADM DSL in order to simplify the modeling experience for AOEs.

AADM Editor:

- Support for semantic validation conducted by the Semantic Reasoner and reported back to the IDE:
    - IDE support to request the semantic validation to the Semantic Reasoner.
    - Visualization of the validation issues and hot fixes within the AADM editor.
    - Application of selected hotfixes into the AADM.
- Support for optimization selection.
- Support for multi viewpoint visual representation and modeling

RM DSL:

- Extended support for designing RM, incorporating additional modeling elements for TOSCA type entities not supported in the first version of this DSL.
- Refactoring of RM DSL to simplify to REs the modeling experience.

RM Editor:

- Support for semantic validation conducted by the Semantic Reasoner and reported back to the IDE:
    - IDE support to request the semantic validation to the Semantic Reasoner. Visualization of the validation issues and hot fixes within the RM editor.
    - Application of selected hotfixes into the RM.

Optimization DSL and Editor:

- Definition of Optimization DSL,
- Textual editor for optimization models
- Syntactic and semantic validation of optimization models

IDE:

- Wizards for AADM/RM model creation
- Wizards/Views for AADM/RM store/retrieval to/from KB.
- Wizards for AADM deployment into the IaC layer.
- Governance dashboard for AADM/RM management
- Governance dashboard for AADM deployment management

6 Implementation

This section presents details about the implementation of the Modelling Layer. As briefly described in Section 1.2.1, the main software components are:

The Semantic Knowledge Base (KB) - A semantic repository (RDF triple store) to accommodate SODALITE's knowledge in the domains of applications, infrastructure, performance optimisations, deployment and lifecycle, and more. This knowledge is generated by multiple stakeholders (e.g. Resource Experts) and represented into RDF-based knowledge graphs (ontologies). The KB should be compliant with W3C Standards, especially regarding RDF, OWL 2 and SPARQL recommendations, and provide the core infrastructure for modelling agility, data integration, relationship exploration, data publishing and consumption. In addition, it should be capable of handling powerful semantic queries and of using inference for uncovering new information out of existing relations.

The Semantic Reasoner - A dedicated middleware to support the interaction with the Semantic Knowledge Base through a REST API for importing/retrieving data. It also implements the mapping logic for mapping application and resource models in the abstraction layer of SODALITE (Section 4). To this end, this module reuses the technologies, tools and standards to implement basic SODALITE reasoning services, such as to handle the native OWL 2 semantics and to facilitate the population of the KB.

The SODALITE IDE - A software component to provide complete support for the authoring of abstract application deployment models with the use of the SODALITE DSL (Section 5).

The WP3 internal workflow is depicted in Figure 24. End users use the SODALITE IDE to define the AADM in the SODALITE DSL. As described in section 4, in order to foster interoperability, the DSL is first translated into an RDF-based exchange model (in Turtle syntax) and then pushed to the Semantic Reasoner API in order to be stored in the Semantic Knowledge Base. The Semantic Reasoner API is also used by the IDE in order to fetch data from the knowledge base (in JSON), such as already defined TOSCA entities and properties relevant to specific TOSCA nodes. The Semantic Population Engine is responsible for populating the KB, using the RDF format. Similarly, the Semantic Reasoning Engine communicates with the KB in order to update (after reasoning) or get information. The Semantic Reasoner API is also used by other components of the system, such as components developed within WP4 (e.g. to perform validation).

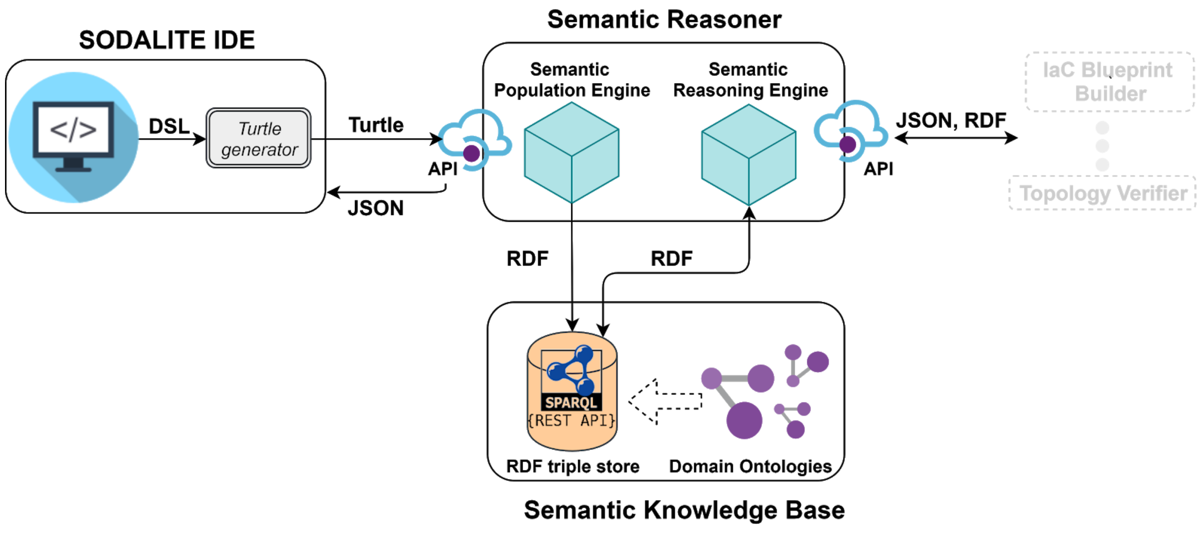


Figure 24. WP3 internal workflow

The rest of this section is structured as follows: in Section 6.1 we present the baseline technologies that have been used or planned to be used to implement the necessary components in the Semantic Modelling layer. We expect to augment these technologies with the necessary features that will be further required to implement the final Modelling Layer. It should be noted that these technologies were selected based on the consortium partners' expertise, as well as the potential to further uptake the work in several tools/technologies that were developed as part of past EU projects and initiatives, in which the consortium partners have been involved. The section continues by describing technical details about the developed WP3 components: the Semantic Knowledge Base (Section 6.2), the Semantic Reasoner (Section 6.3) and the IDE (Section 6.4).

6.1 WP3 baseline technology stack

The following technologies are being considered to be used for the WP3 developments.

6.1.1 TopBraid Composer

Ontology editors are software tools that allow the creation and maintenance of ontologies through a graphical user interface. They provide the interfaces that help end users carry out the main activities of the ontology development process, such as conceptualization, implementation, consistency checking and documentation [3]. A number of ontology editors have been developed, each one having different capabilities and targeting different users, according to their expertise. In SODALITE, we have used the TopBraid Composer[30] as the ontology development environment.

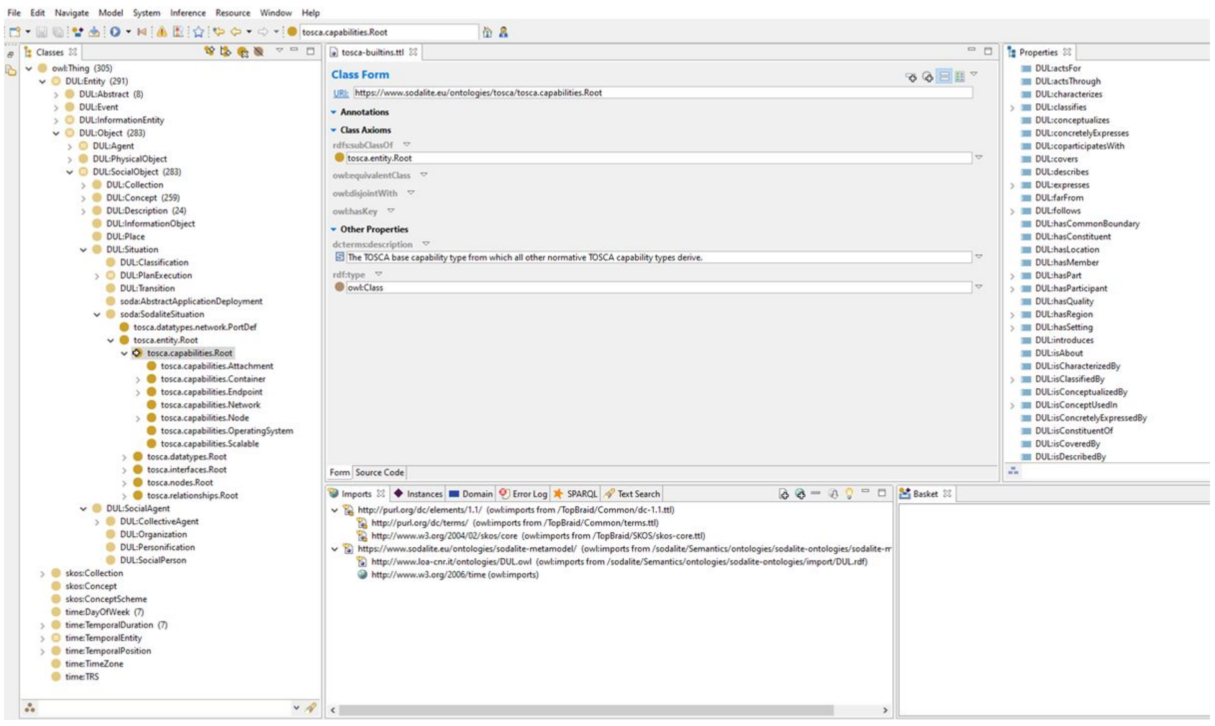
TopBraid Composer is a powerful integrated development environment for implementing Knowledge Graphs and Linked Data services. It is an enterprise-class modelling environment for developing Semantic Web ontologies and building semantic applications. Fully compliant with W3C standards, the suite supports the development of RDF and OWL ontologies, providing at the same time advanced querying and reasoning services. It is implemented as an Eclipse plug-in and it can be used to develop ontology models, convert data and models to and from RDF/OWL, transform and integrate data source integration, and develop Semantic Web services and applications. More specifically, TopBraid Composer lets users:

- Create ontology models.
- Create RDF data.
- Use Forms with drop-downs, autocomplete and wizards, use Visual editors with diagrams for classes and RDF graphs or use syntax-directed text entry.
- Auto-convert from RDFS/OWL to SHACL.
- Auto-generate SHACL from data.
- Work with files or databases.
- Refactor models and data.

It also provides rich capabilities for:

- Inferencing.
- Ontology mapping.
- Auto-generation of SPARQL "by example" from the graph view.
- Query and Rule development with auto-complete and templates.
- Testing – with SPARQL and GraphQL Endpoints running on the localhost server.

[30] https://www.topquadrant.com/products/topbraid-composer/



```
Classes
owl:Thing (305)
  DUL:Entity (291)
    DUL:Abstract (8)
    DUL:Event
    DUL:InformationEntity
    DUL:Object (283)
      DUL:Agent
      DUL:PhysicalObject
      DUL:SocialObject (283)
        DUL:Collection
        DUL:Concept (259)
        DUL:Description (24)
        DUL:InformationObject
        DUL:Place
        DUL:Situation
          DUL:Classification
          DUL:PlanExecution
          DUL:Transition
          soda:AbstractApplicationDeployment
          soda:SodaliteSituation
            tosca.datatypes.network.PortDef
            tosca.entity.Root
              tosca.capabilities.Root
                tosca.capabilities.Attachment
                tosca.capabilities.Container
                tosca.capabilities.Endpoint
                tosca.capabilities.Network
                tosca.capabilities.Node
                tosca.capabilities.OperatingSystem
                tosca.capabilities.Scalable
              tosca.datatypes.Root
              tosca.interfaces.Root
              tosca.nodes.Root
              tosca.relationships.Root
        DUL:SocialAgent
          DUL:CollectiveAgent
          DUL:Organization
          DUL:Personification
          DUL:SocialPerson
skos:Collection
skos:Concept
skos:ConceptScheme
time:DayOfWeek (7)
time:TemporalDuration (7)
time:TemporalEntity
time:TemporalPosition
time:TimeZone
time:TRS
```

```
tosca-builtins.ttl
Class Form
URI: https://www.sodalite.eu/ontologies/tosca/tosca.capabilities.Root
Annotations
Class Axioms
rdfs:subClassOf
  tosca.entity.Root
owl:equivalentClass
owl:disjointWith
owl:hasKey
Other Properties
dcterms:description
  The TOSCA base capability type from which all other normative TOSCA capability types derive.
rdf:type
  owl:Class
```

```
Properties
DUL:actsFor
DUL:actsThrough
DUL:characterizes
DUL:classifies
DUL:conceptualizes
DUL:concretelyExpresses
DUL:coparticipatesWith
DUL:covers
DUL:describes
DUL:expresses
DUL:farFrom
DUL:follows
DUL:hasCommonBoundary
DUL:hasConstituent
DUL:hasLocation
DUL:hasMember
DUL:hasPart
DUL:hasParticipant
DUL:hasQuality
DUL:hasRegion
DUL:hasSetting
DUL:introduces
DUL:isAbout
DUL:isCharacterizedBy
DUL:isClassifiedBy
DUL:isConceptualizedBy
DUL:isConceptUsedIn
DUL:isConcretelyExpressedBy
DUL:isConstituentOf
DUL:isCoveredBy
DUL:isDescribedBy
```

```
Imports
http://purl.org/dc/elements/1.1/  (owl:imports from /TopBraid/Common/dc-1.1.ttl)
  http://purl.org/dc/terms/  (owl:imports from /TopBraid/Common/terms.ttl)
  http://www.w3.org/2004/02/skos/core  (owl:imports from /TopBraid/SKOS/skos-core.ttl)
https://www.sodalite.eu/ontologies/sodalite-metamodel/  (owl:imports from /sodalite/Semantics/ontologies/sodalite-ontologies/sodalite-m
  http://www.loa-cnr.it/ontologies/DUL.owl  (owl:imports from /sodalite/Semantics/ontologies/sodalite-ontologies/import/DUL.rdf)
  http://www.w3.org/2006/time (owl:imports)
```

Figure 25. TopBraid composer Class and Property views

TopBraid Composer comes in three editions: Maestro, Standard and Free. Although the Free edition has some limitations compared to the other two editions, such as direct connectivity with RDF triple stores, it provides a fully-fledged ontology editor (Figure 25) that covers all the ontology development requirements in SODALITE.

Other well-known ontology editors include Protégé[31] and Fluent Editor[32]. **Protégé** is one of the most widely used ontology development tools, which was developed at Stanford University. It is a free, open-source platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. It mainly supports the creation and editing of one or more ontologies in a single workspace via a completely customisable user interface. Visualization tools allow for interactive navigation of ontology relationships. Advanced explanation support aids in tracking down inconsistencies. Refactor operations available including ontology merging, moving axioms between ontologies, rename of multiple entities, and more.

Fluent Editor is a tool for editing, manipulating and querying complex ontologies written in OWL, RDF or SWRL. It is fully compatible with most of the Semantic Web W3C standards (OWL, RDF, and SPARQL) but at the same time has an intuitive user interface that uses the Ontorion Controlled Natural Language (OCNL) that is a human friendly alternative to XML ontology language like OWL or RDF but is completely compatible with OWL2, RDF and SWRL. Furthermore, the OCNL can also be used as a query language compatible with SPARQL.

Although Protégé is considered as the most popular ontology editor [4], TopBraid Composer has been selected as the main ontology editor in SODALITE mainly due to the advanced meta-modelling capabilities it offers. As we explain in Section 3.3, the conceptual model of SODALITE makes extensive use of meta-modelling, where classes are treated as properties as well. TopBraid Composer makes it easier to use the same name for both an instance and a class. However, it should be noted that SODALITE ontologies are not editor-specific. Any general-purpose ontology editor can be used that supports basic ontology development tasks.

[31] https://protege.stanford.edu/

[32] https://www.cognitum.eu/Semantics/FluentEditor/

6.1.2 GraphDB

The RDF triple store is a type of graph database that stores semantic facts. Being a graph database, a triple store handles data as a network of objects with materialized links between them. This makes RDF triple stores the preferred choice for managing highly interconnected data, such as in the SODALITE domain where resource and application models are interconnected through the semantic abstraction layer.

GraphDB[33] is a family of highly efficient, robust and scalable RDF databases. It streamlines the load and use of linked data cloud datasets, as well as a user's own resources. For easy use and compatibility with the industry standards, GraphDB implements the RDF4J framework interface, the W3C SPARQL Protocol specification, and supports all RDF serialization formats. The database is the preferred choice of both small independent developers and big enterprise organizations because of its community and commercial support, as well as excellent enterprise features such as cluster support and integration with external high-performance search applications - Lucene, Solr and ElasticSearch.

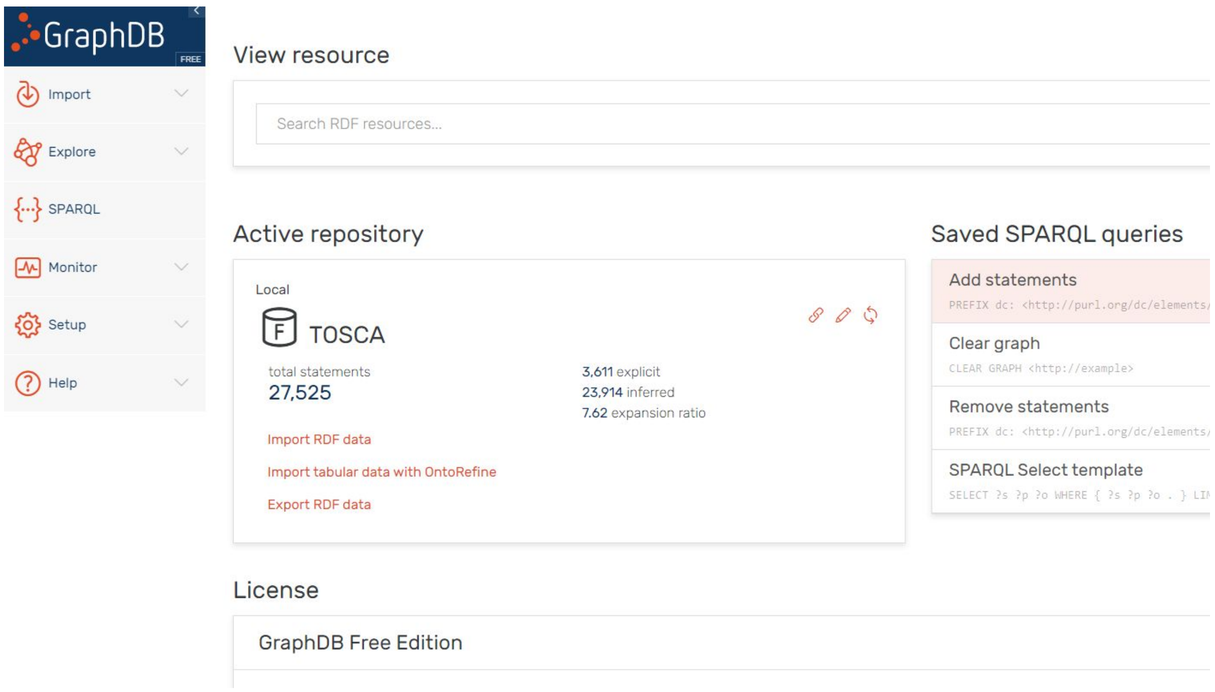
GraphDB is one of the few triple stores that can perform semantic inferencing at scale, allowing users to derive new semantic facts from existing facts. It handles massive loads, queries, and inferencing in real time [5]. GraphDB comes with three editions: Free, Standard and Enterprise.

GraphDB is packaged as a storage and inference layer for RDF4J and makes extensive use of the features and infrastructure of RDF4J, especially the RDF model, RDF parsers, and query engines. Inference is performed by GraphDB's native reasoning engine, where the explicit and inferred statements are stored in highly optimised data structures that are kept in-memory for query evaluation and further inference. The inferred closure is updated through inference at the end of each transaction that modifies the repository.

GraphDB comes with a web-based administration tool (Workbench, Figure 26), which provides a REST API for automating various tasks for managing and administering repositories. The tool can be also used for:

- Managing GraphDB repositories.
- Loading and exporting data.
- Executing SPARQL queries and updates.
- Managing namespaces.
- Managing contexts.
- Viewing/editing RDF resources.
- Monitoring queries.
- Monitoring resources.
- Managing users and permissions.
- Managing connectors.

[33] http://graphdb.ontotext.com/



Local
TOSCA
total statements
27,525
3,611 explicit
23,914 inferred
7.62 expansion ratio
Import RDF data
Import tabular data with OntoRefine
Export RDF data

Add statements
PREFIX dc: <http://purl.org/dc/elements/
Clear graph
CLEAR GRAPH <http://example>
Remove statements
PREFIX dc: <http://purl.org/dc/elements/
SPARQL Select template
SELECT ?s ?p ?o WHERE { ?s ?p ?o . } LI

License
GraphDB Free Edition

Figure 26. Home page of GraphDB Workbench

Other alternative triple stores include AllegroGraph[34], OpenLink Virtuoso[35] and Jena TDB[36]. GraphDB has been selected mainly because it is easy to install and deploy, it provides an easy-to-use administration interface (Workbench) and provides one of the fastest querying and reasoning engines. However, it should be noted that because SODALITE capitalises on existing standards for modelling, reasoning and querying ontologies, any W3C compliant RDF triple store can be used as the underlying RDF triple store in SODALITE.

6.1.3 Eclipse RDF4J

Eclipse RDF4J[37] is a powerful Java framework for processing and handling RDF data. This includes creating, parsing, scalable storage, reasoning and querying with RDF and Linked Data. It offers an easy-to-use API that can be connected to all leading RDF database solutions, including GraphDB. It enables us to connect with SPARQL endpoints and create applications that leverage the power of linked data and Semantic Web.

Programmatically, GraphDB can be used via the RDF4J Java framework of classes and interfaces. RDF4J comprises a large collection of libraries, utilities and APIs. The important components in SODALITE for accessing GraphDB are:

- the RDF4J classes and interfaces (API), which provide a uniform access to the components from multiple vendors/publishers.
- the RDF4J server application.

RDF4J's RDF database API differs from comparable solutions in that it offers a stackable interface through which functionality can be added, and the storage engine is abstracted from the query interface. Many other triple stores can be used through the RDF4J API, including AllegroGraph and OpenLink Virtuoso.

[34] https://franz.com/agraph/allegrograph/
[35] https://virtuoso.openlinksw.com/
[36] https://jena.apache.org/documentation/tdb/
[37] https://rdf4j.org/

6.1.4 SPARQL

SPARQL[38] is the W3C recommendation for querying RDF graphs and its specification defines the syntax and semantics of the SPARQL query language for RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. It also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs.

SPARQL is based on matching graph patterns. The simplest graph pattern is the triple pattern, which is like an RDF triple but with the possibility of a variable instead of an RDF term in the subject, predicate, or object positions. Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed to fulfil a pattern.

However, SPARQL is more than just a query language. It is also an HTTP-based transport protocol, where any SPARQL endpoint can be accessed via a standardised transport layer. RDF results can be returned in several data-interchange formats and RDF entities are identified by Universal Resource Identifiers (URIs).

SPARQL has four types of queries. It can be used to:

- ASK whether there is at least one match of the query pattern in the RDF graph data.
- SELECT all or some of those matches in a tabular form (including aggregation, sampling and pagination through OFFSET and LIMIT).
- CONSTRUCT an RDF graph by substituting the variables in these matches in a set of triple templates.
- DESCRIBE the matches found by constructing a relevant RDF graph.

SPARQL is supported by leading semantic graph databases that also provide intuitive SPARQL editors with autocomplete, explorer and many other features that facilitate building powerful SPARQL queries.

6.1.5 XText

XText[39] is an Eclipse based framework for designing domain specific languages (DSLs) and implementing associated textual editors, which are integrated within the Eclipse workbench. XText leverages the Eclipse Modelling Framework[40] (EMF) to manage the internal model representation of the DSL.

A modeller describes the DSL using the XText grammar. This approach largely simplifies the specification of the DSL, supporting an incremental and iterative DSL authoring. XText DSL grammar definition is modular and composable: DSL entities can refer to other entities defined in other DSLs. From the DSL grammar, Xtext automates the generation of the DSL textual editor, integrated with Eclipse. The editor is enriched with a large number of features that largely improve the user's experience, namely:

- Syntax/semantic colouring: DSL keywords and elements are rendered differently according to a colour schema.
- Validation/Error checking: detection of model entities that are not compliant with the DSL specification (i.e. grammar).
- Content-assistance, auto-completion, cross-references: user is assisted during the authoring of textual DSL models, with suggested/automated completions of DSL snippets, or suggested references to other DSL elements.

[38] https://www.w3.org/TR/sparql11-query/
[39] https://www.eclipse.org/Xtext/
[40] https://www.eclipse.org/modeling/emf/

- Automatic formatting: correct indentation is automatically managed by Xtext.
- Hover information: information about DSL entities are displayed when mouse pointer is hovering over them.
- Quick fix proposals: on spotted validation errors, available quick fixes could be offered to the user and applied into the DSL text, when accepted.
- Outline/Structure view: aside view offers an outline of the DSL elements of the model that is displayed in the DSL editor.
- Automatic conversion into any textual serialization.

Apart from these user's oriented features, Xtext supports the generation of a Web-based textual editor, which can be embedded within some of the most popular Web-based code editors, including Orion[41], Ace[42] or Code Mirror[43]. It can also be integrated with Eclipse based graphical model frameworks, including GEF[44], Sirius[45] or Graphiti[46]. Even more, it supports the Language Server Protocol[47].

6.2 Semantic Knowledge Base

The Semantic Knowledge Base (KB) is the semantic database management system of SODALITE that enables storing, querying and managing structured data. It follows the semantic data schema paradigm, called ontology, which is stored and managed independently from the data. As described in Section 3, SODALITE follows a pattern-based approach to define the application and resource abstraction layer, making extensive use of meta-modelling. As such, both the upper-level patterns and the instantiation of these patterns (resource and application models) are stored and interlinked in the KB, creating the SODALITE Knowledge Graphs that have the following characteristics:

- They have a formal structure (RDF/OWL) that enables their retrieval and reuse in an efficient and unambiguous manner.
- Resource descriptions, such as classes, properties and instances, form a network (graph), where each entity represents part of the description of the entities, related to it.

The KB of SODALITE logically encapsulates two subcomponents:

1. **The RDF triple store:** It is the semantic graph database of SODALITE (NoSQL graph database) that provides advanced integrating capabilities of heterogeneous data, defining links between resources and entities. SODALITE builds its RDF triple store on top of GraphDB, one of the most powerful RDF repositories that offers:
    a. Fast loading and indexing of W3C compliant ontologies (RDF, OWL2).
    b. Full standard-compliant reasoning for RDFS and OWL 2 RL.
    c. Full SPARQL 1.1 support.
    d. Query optimizer allowing effective query execution.
    e. Compatibility with the RDF4J framework.
2. **Domain ontologies:** These ontologies define the conceptual model of SODALITE. Their expressivity is compliant with the OWL 2 RL profile. The current version of the ontologies include modules that provide:
    a. The formal schema, i.e. classes and properties that can be used to capture application and resource models (TOSCA ontology).

[41] http://eclipse.org/orion/
[42] http://ace.c9.io/
[43] http://codemirror.net/
[44] https://www.eclipse.org/gef/
[45] https://www.eclipse.org/sirius/
[46] https://www.eclipse.org/graphiti/
[47] https://langserver.org/

b. The ontology pattern that should be followed (SODALITE meta-model) in order to define modular and reusable knowledge graphs.

In the following we provide technical details about the components.

6.2.1 RDF Triple Store

Software dependencies

- GraphDB 9.0.0.
- Java SE Development Kit 8, 11, or 12.
- RDF4J V3.0.0.
- Windows 10.

Requirements

- To support storing, querying and management of structured data.
- To support existing Semantic Web standards (RDF, OWL 2, SPARQL).
- To allow remote access (HTTP protocol).
- To provide a SPARQL endpoint.
- To support native RDF / OWL 2 RL reasoning.

Composed of

GraphDB (third-party software) is used as the underlying RDF triple store of SODALITE.

Roles that interact with the component (i.e. AOE, RE)

There is no direct interaction of AOEs and REs with the RDF triple store.

Depends on

N/A

Repositories

There is no repository for the triple store. It is a third party, standalone component.

6.2.2 Domain Models

Software dependencies

N/A

Requirements

To provide the necessary knowledge structures and vocabularies to model application/infrastructure models.

Composed of

Serialisation of RDF graphs in different formats (e.g. Turtle).

Roles that interact with the component (i.e. AOE, RE)

N/A

Depends on

N/A

Repositories
https://github.com/SODALITE-EU/semantic-models

6.3 Semantic Reasoner

The Semantic Reasoner can be seen as a service over the Knowledge Base, which acts as the interface to the KB for saving, updating and retrieving information. This interface is used both by components that belong to the Semantic Modelling layer, e.g. the IDE, and by components that belong to different architecture layers, such as in the IaC layer.

The population of the KB, i.e. the instantiation of the respective ontology patterns to capture resources and applications (AADM), is performed by the Semantic Reasoner, which encapsulates the necessary logic to translate the DSL composed in the IDE by the users to the conceptual model of SODALITE. In addition, the Semantic Reasoner provides all the necessary interfaces to retrieve data from the KB, as well as to expose reasoning functionality developed in WP4 with respect to searching and validation services.

More specifically, the Semantic Reasoner logically clusters two modules: the Semantic Population Engine and the Semantic Reasoning Engine.

1. **Semantic Population Engine (SPE):** implements the custom population logic of the KB, i.e. the mapping of TOSCA-related definitions to the abstraction model of SODALITE, instantiating the ontology patterns and applying advanced meta-modelling techniques.
2. **Semantic Reasoning Engine (SRE):** ensures the consistency of the RDF knowledge graphs in terms of native OWL 2 semantics, interfacing with GraphDB's internal OWL 2 RL reasoning engine. It also provides the reasoning infrastructure needed to implement custom reasoning logic in WP4/T4.4 - Analytics and Semantic Decision Support - regarding searching, validation and reuse. Finally, it provides the REST API through which the various components can interact with the KB.

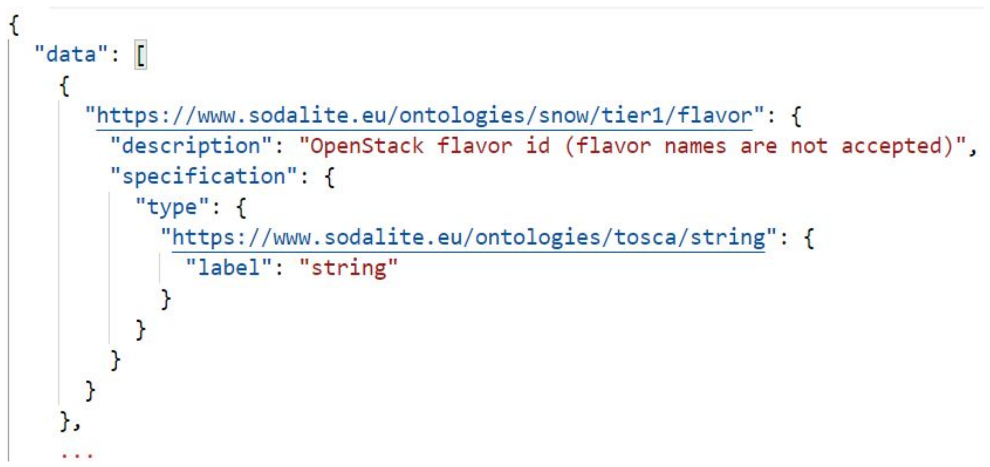
6.3.1 Semantic Reasoning Engine

During the first year of the project, a number of REST API endpoints have been developed in order to assist users in defining models in the IDE. It should be noted that this REST API exposes functionality that has been mainly developed in WP4/T4.4 relevant to searching and validation. More details on the backend implementation of the REST API are provided in D4.1 (Topology Verifier module).

The REST API returns information in JSON. To this end, the Semantic Reasoning Engine performs a transformation of the RDF graphs returned by decision making (WP4) into a lightweight JSON structure in order to ease the processing at the IDE level. In the rest of this section, we present example responses.

GET /properties(resource)

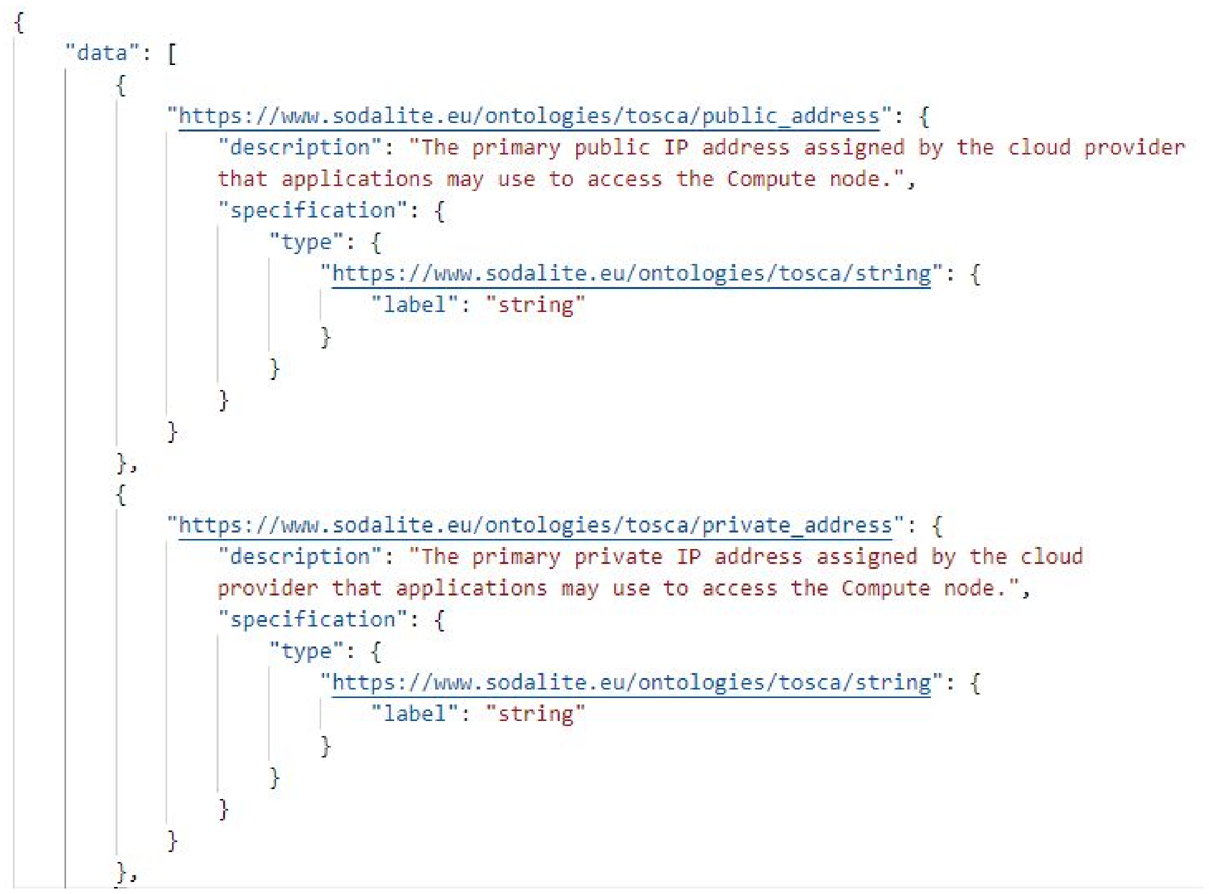
It allows the IDE to retrieve the properties of a node type. This is useful when users are defining a node template of a specific node type and want to be informed about relevant properties. It should be noted that the interface returns all properties relevant to the node, either directly defined in the specification of the node type or inherited from super nodes (derives_from TOSCA relationship). An example JSON output for the properties of sodalite.nodes.VM.OpenStack is presented in Figure 27.



```
{
  "data": [
    {
      "https://www.sodalite.eu/ontologies/snow/tier1/flavor": {
        "description": "OpenStack flavor id (flavor names are not accepted)",
        "specification": {
          "type": {
            "https://www.sodalite.eu/ontologies/tosca/string": {
              "label": "string"
            }
          }
        }
      }
    },
    ...
```

Figure 27. Example response for getting the properties of sodalite.nodes.VM.OpenStack

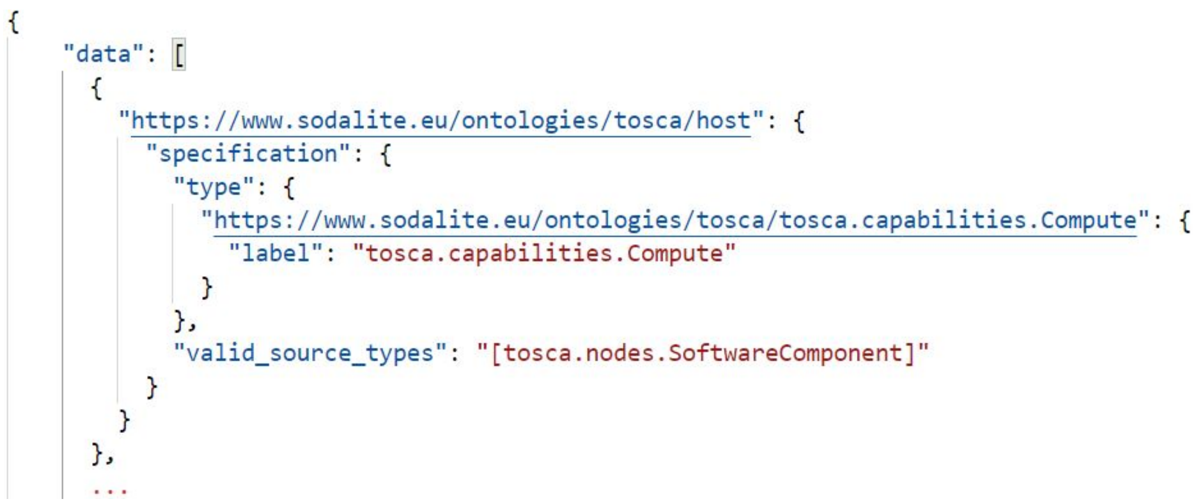
GET /attributes(resource)
This interface is similar to the /properties endpoint, returning relevant attributes.



```
{
    "data": [
        {
            "https://www.sodalite.eu/ontologies/tosca/public_address": {
                "description": "The primary public IP address assigned by the cloud provider
                that applications may use to access the Compute node.",
                "specification": {
                    "type": {
                        "https://www.sodalite.eu/ontologies/tosca/string": {
                            "label": "string"
                        }
                    }
                }
            }
        },
        {
            "https://www.sodalite.eu/ontologies/tosca/private_address": {
                "description": "The primary private IP address assigned by the cloud
                provider that applications may use to access the Compute node.",
                "specification": {
                    "type": {
                        "https://www.sodalite.eu/ontologies/tosca/string": {
                            "label": "string"
                        }
                    }
                }
            }
        },
```

Figure 28. Example response for getting the attributes of sodalite.nodes.VM.OpenStack

GET /capabilities(resource)
Returns the capabilities of a resource, both the ones directly asserted for the resource, and the inherited ones following the hierarchy (Figure 29).

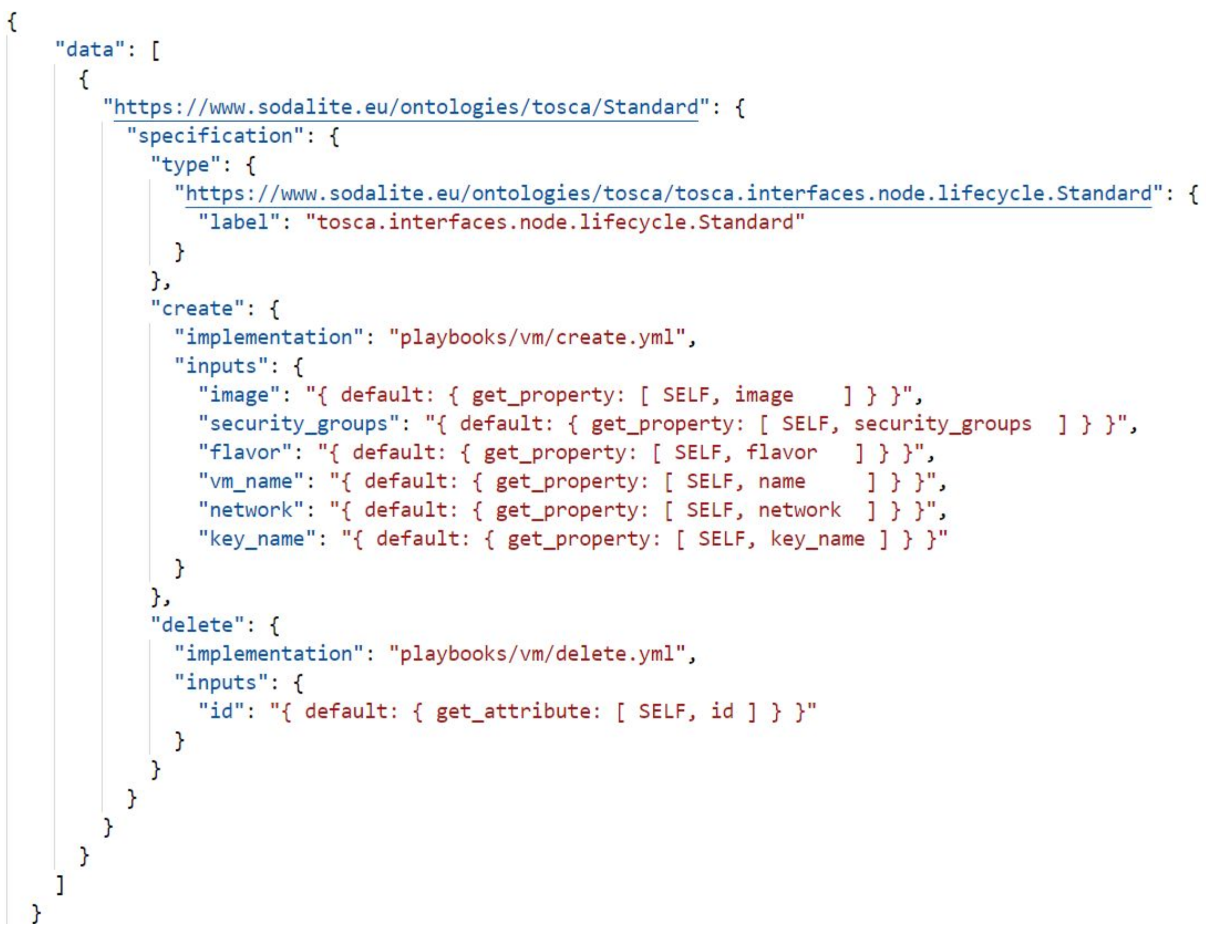


```
{
    "data": [
      {
        "https://www.sodalite.eu/ontologies/tosca/host": {
          "specification": {
            "type": {
              "https://www.sodalite.eu/ontologies/tosca/tosca.capabilities.Compute": {
                "label": "tosca.capabilities.Compute"
              }
            },
            "valid_source_types": "[tosca.nodes.SoftwareComponent]"
          }
        }
      },
      ...
```

Figure 29. Example response for getting the capabilities of sodalite.nodes.VM.OpenStack

GET /interfaces(resource)

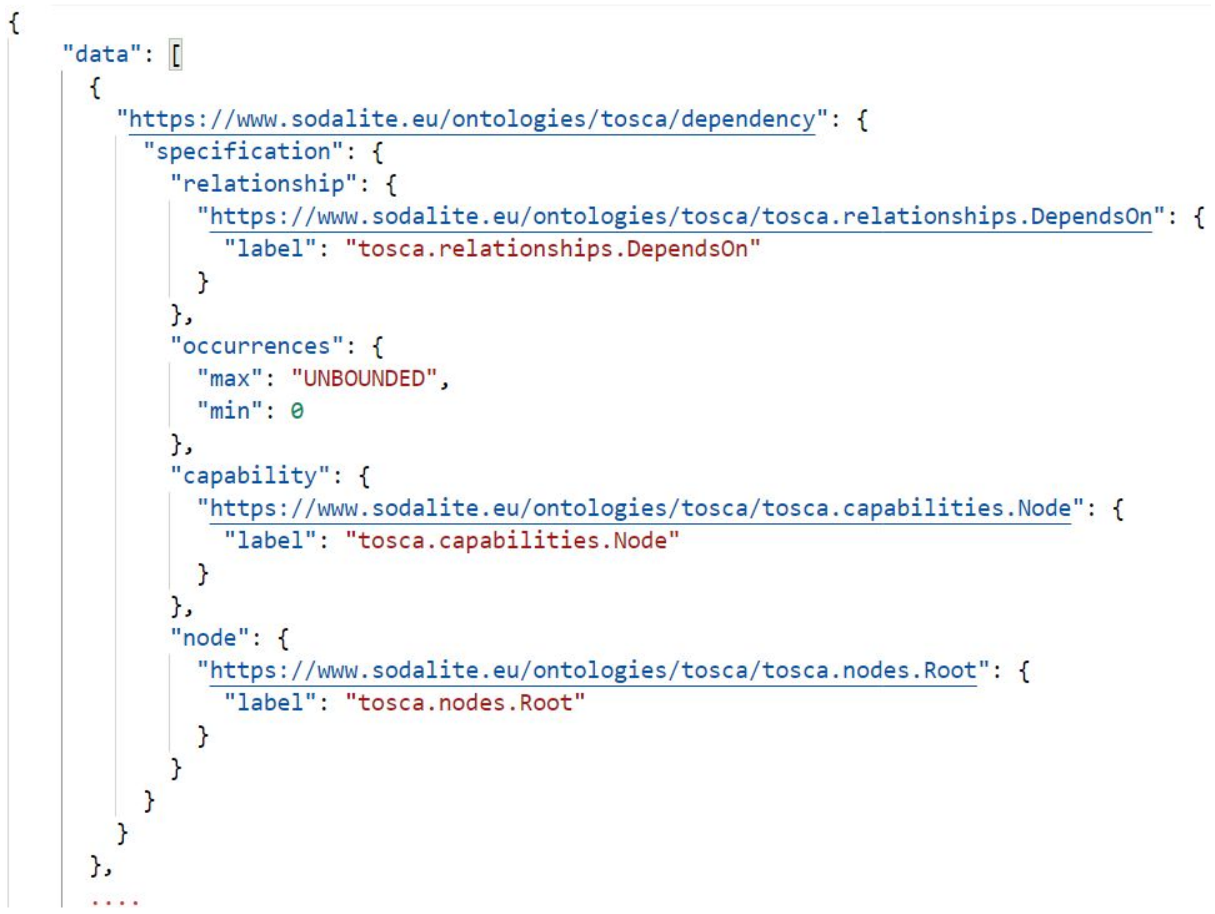
Returns the interfaces of a resource.



```
{
    "data": [
      {
        "https://www.sodalite.eu/ontologies/tosca/Standard": {
          "specification": {
            "type": {
              "https://www.sodalite.eu/ontologies/tosca/tosca.interfaces.node.lifecycle.Standard": {
                "label": "tosca.interfaces.node.lifecycle.Standard"
              }
            },
            "create": {
              "implementation": "playbooks/vm/create.yml",
              "inputs": {
                "image": "{ default: { get_property: [ SELF, image    ] } }",
                "security_groups": "{ default: { get_property: [ SELF, security_groups  ] } }",
                "flavor": "{ default: { get_property: [ SELF, flavor   ] } }",
                "vm_name": "{ default: { get_property: [ SELF, name     ] } }",
                "network": "{ default: { get_property: [ SELF, network  ] } }",
                "key_name": "{ default: { get_property: [ SELF, key_name ] } }"
              }
            },
            "delete": {
              "implementation": "playbooks/vm/delete.yml",
              "inputs": {
                "id": "{ default: { get_attribute: [ SELF, id ] } }"
              }
            }
          }
        }
      }
    ]
}
```

Figure 30. Example response for getting the interfaces of sodalite.nodes.VM.OpenStack

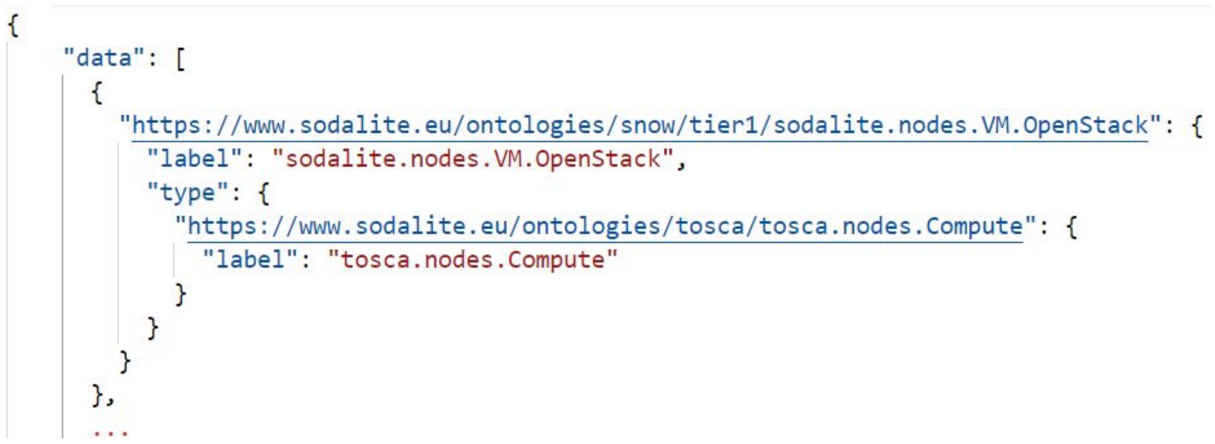
GET /requirements(resource)
Returns the requirements of a resource.



```
{
    "data": [
      {
        "https://www.sodalite.eu/ontologies/tosca/dependency": {
          "specification": {
            "relationship": {
              "https://www.sodalite.eu/ontologies/tosca/tosca.relationships.DependsOn": {
                "label": "tosca.relationships.DependsOn"
              }
            },
            "occurrences": {
              "max": "UNBOUNDED",
              "min": 0
            },
            "capability": {
              "https://www.sodalite.eu/ontologies/tosca/tosca.capabilities.Node": {
                "label": "tosca.capabilities.Node"
              }
            },
            "node": {
              "https://www.sodalite.eu/ontologies/tosca/tosca.nodes.Root": {
                "label": "tosca.nodes.Root"
              }
            }
          }
        }
      },
      ....
```

Figure 31. Example response for getting the interfaces of sodalite.nodes.VM.OpenStack

GET /nodes
Returns all known nodes from the KB.



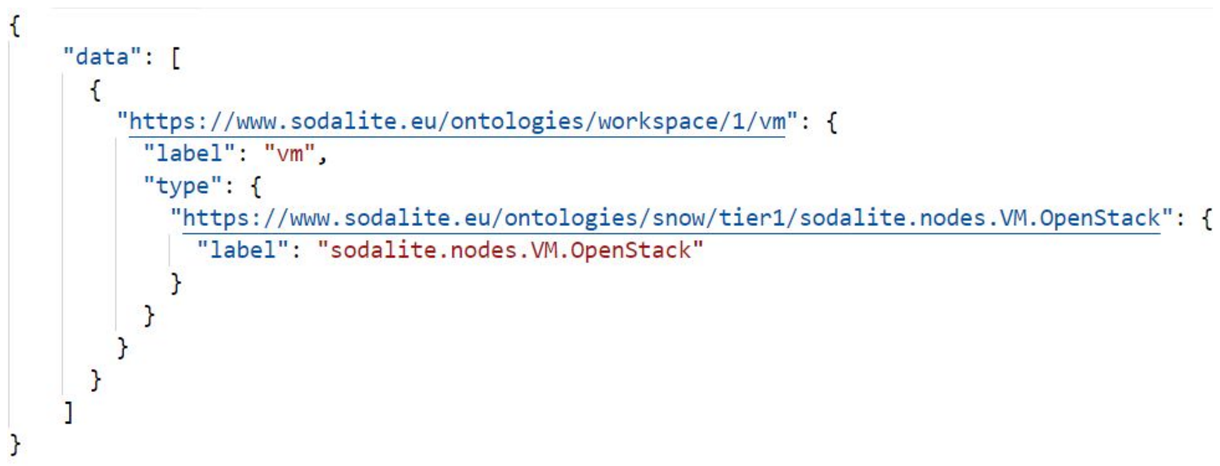
```
{
    "data": [
      {
        "https://www.sodalite.eu/ontologies/snow/tier1/sodalite.nodes.VM.OpenStack": {
          "label": "sodalite.nodes.VM.OpenStack",
          "type": {
            "https://www.sodalite.eu/ontologies/tosca/tosca.nodes.Compute": {
              "label": "tosca.nodes.Compute"
            }
          }
        }
      },
      ...
```

Figure 32. Example response of getting all known nodes

GET /valid-requirement-nodes(requirement, nodeType)
Returns nodes that satisfy a certain requirement, when defining a node template of type nodeType. This interface actually exposes the WP4 decision making functionality relevant to

suggesting to IDE users entities that are semantically valid, checking TOSCA constraints on capabilities and nodes. Figure 33 depicts the response of the API when it is called for requirement=host and nodeType = tosca.nodes.SoftwareComponent. In this example, the vm node template is returned, since it is the only node template in the KB that satisfies the requirement tosca.nodes.SoftwareComponent sets on host, according to TOSCA specification:

```
tosca.nodes.SoftwareComponent:
      [...]
      requirements:
        - host:
          relationship:
            type: tosca.relationships.HostedOn
          capability: tosca.capabilities.Compute
          node: tosca.nodes.Compute
          occurrences: [ 1, 1 ]
```



```
{
    "data": [
      {
        "https://www.sodalite.eu/ontologies/workspace/1/vm": {
          "label": "vm",
          "type": {
            "https://www.sodalite.eu/ontologies/snow/tier1/sodalite.nodes.VM.OpenStack": {
              "label": "sodalite.nodes.VM.OpenStack"
            }
          }
        }
      }
    ]
}
```

Figure 33. example response on calling /valid-requirement-nodes with requirement=host and nodeType = tosca.nodes.SoftwareComponent

POST /saveAADM (ttl, aadm, submissionId)

Stores the submitted AADM in the KB, assigning a unique id (IRI), which is returned back to the client. As described in Section 4.3.2, the IDE sends to the REST API both the DSL definition and an RDF-based translation of DSL into the exchange model that has been defined between the IDE and the Semantic Reasoner. The exchange model actually corresponds to a lightweight version of the SODALITE ODP whose purpose is to define an interoperable way of exchanging information between the frontend and the backend. The role of the Semantic Population Engine is to translate this exchange model into the full conceptual model of SODALITE and populate the KB. Since in the first version of the ontologies and the semantic repository versioning is not supported, there is a need to manually assign an id to the submitted model (submissionId parameter), in order to uniquely identify different versions. In future versions, this parameter will be omitted.

GET /aadm(iri)

Returns the AADM JSON that corresponds to the resource Id (IRI). This is the interface used in WP4 in order to get the AADM definition of a single model. To this end, the Semantic Reasoning Engine performs a translation of the RDF AADM into a JSON. As a parameter, this endpoint requires the id (IRI) returned by calling the saveAADM endpoint.

GET /aadm(iri, format=dsl)
Returns the AADM in DSL textual format that corresponds to the resource Id (IRI).

GET /aadm(user_iri)
Returns the list of AADM models created by a user, given by their id.

Software dependencies

- Java SE Development Kit 8, 11, or 12.
- RDF4J v3.0.0.
- Jersey RESTful Web Services.
- Tomcat 9 (for deploying the REST API).

Requirements

- To support native OWL 2 reasoning.
- To provide the reasoning infrastructure for custom rule-based logic.
- To provide the interface to access the semantic repository.

Composed of
The REST API and the reasoning infrastructure by interfacing with the GraphDB reasoning engine.

Roles that interact with the component (i.e. AOE, RE)
There is no direct interaction of Application Ops and Resource Experts with the Semantic Reasoner.

Depends on
The Semantic Reasoning Engine depends on the Semantic Knowledge Base.

Repositories
https://github.com/SODALITE-EU/semantic-reasoner

6.3.2 Semantic Population Engine

Software dependencies

- Java SE Development Kit 8, 11, or 12.
- RDF4J v3.0.0.

Requirements

- To map DSL definitions to the SODALITE semantic models.
- To ensure the mapping consistency.

Composed of
Mapping services of the RDF-based exchange model on the SODALITE ODP.

Roles that interact with the component (i.e. AOE, RE)
There is no direct interaction of Application Ops and Resource Experts with the Semantic Population Engine.

Depends on

The Semantic Population Engine depends on the Semantic Knowledge Base.

Repositories
https://github.com/SODALITE-EU/semantic-reasoner

6.4 SODALITE IDE

The SODALITE IDE is the visual programming interface between the end users, namely the Application Ops Experts (AOEs) and the Resource Experts (REs) (see D2.1), and the SODALITE Infrastructure as Code (IaC) Layer (see D2.1). The IDE enables (see D2.1 for additional details):

- Application Ops Experts to:
  - Define an Abstract Application Deployment Model (AADM).
  - Select suitable infrastructure/platform resources from the KB that satisfy the requirements of the AADM nodes.
  - Store the AADM into the KB.
  - Initiates the deployment of the AADM within the IAC layer.
- Resource Experts (REs) to:
  - Model infrastructure/platform resources to be stored into the KB.
  - Map resources and optimizations.

The IDE provides complete support for design-time modelling of AADMs by AOEs and resource models (RMs) by REs. Both AADMs and RMs are compliant to the SODALITE DSLs defined (based on TOSCA) to support the deployment of complex systems into distributed infrastructures (e.g. Cloud or HPC).
Both users use the IDE to:

- Browse the AADMs/RMs stored locally (e.g. with the user's local file system) or remotely in the KB. AOEs/REs can retrieve/store models from/to the KB.
- Create/update AADMs/RMs within a textual editor. Future versions of the IDE will consider support for browser textual edition, or standalone IDE graphical AADM authoring. Current prototype consists of a standalone IDE with AADM/RM textual edition.
- Verify the syntactic/semantic validity of AADMs/RMs. Syntactic conformance (to the AADM/RM schema) is managed by the textual editor. Semantic conformance is verified by the Semantic Reasoner. Verification issues are reported to the user in the textual editor, next to the affected element.
  - As part of the verification or the optimization task initiated within the deployment process, verification and/or optimization suggestions will be presented to the user, next to affected AADM/RM elements. Available quick fixes will be provided and executed when selected.
- Request the deployment of the AADM within the SODALITE IaC layer.

AADM and RM authoring in the IDE requires users to provide a minimal amount of information, as it relies on the Semantic KB (interfaced by the Semantic Reasoner) and the IaC to infer missing information that fills the gaps. Therefore, proposed SODALITE DSLs for AADM and RM, despite the fact that they are based on the TOSCA specification, are a subset of them that focus on gathering only essential information.
The IDE relies on the KB (through the Semantic Reasoner) to guide the AOEs on the authoring of an AADM, by providing suggestions to complete the AADM, by detecting and spotting conformance issues, by suggesting optimization patches, and so on.
The IDE serializes the AADMs into a Turtle serialization format that is compatible with the KB; then it sends them to the KB to be stored. In addition, upon user's request, the IDE sends the AADM to the IaC framework in JSON format to be deployed into the target infrastructure.

The IDE consists of the following components:

- DSLs for AADM and RM.
- DSL Editor.

These components are described in the following in more detail.

Software dependencies

The IDE depends on the following software dependencies:

- Eclipse 2019-09 (4.13.0).
- EMF- Eclipse Modelling Framework 2.19.0.
- XText Complete SDK 2.19.0.

Requirements

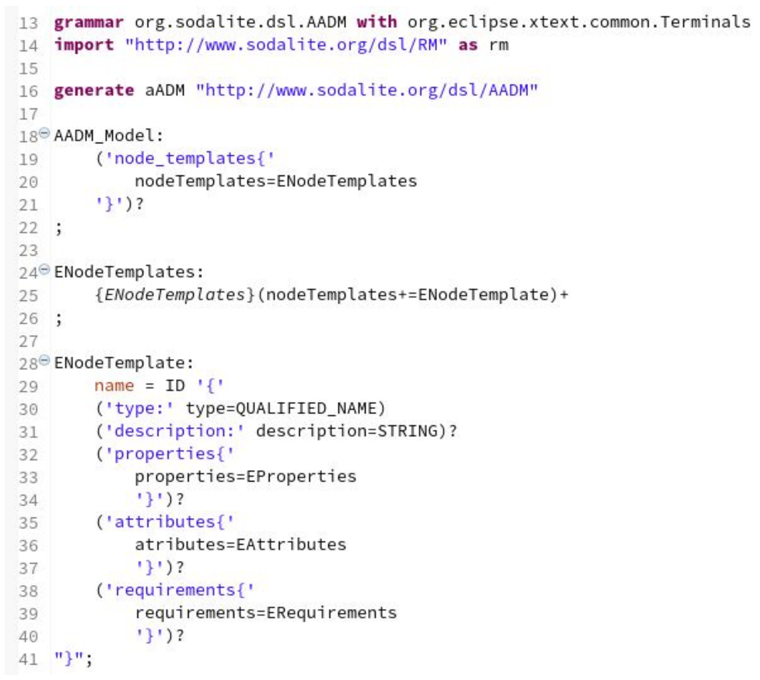
The IDE is requested to fulfil the following requirements (see D2.1):

- Support the specification of AADMs:
    - design the application topology.
    - describe application constraints.
    - design inter-component boundaries.
    - express optimization requirements and constraints.
    - Assign target resources to the AADM.
- Support the specification of RMs.
- Show to the AOE and RE known resources from the KB.
- Conduct a syntactic and semantic validation of AADMs and RMs.
- Show validation inconsistencies and other recommendations.
- Store AADMs and RMs into the KB.
- Request to the IaC layer the validation of the AADM.
- Request to the IaC layer the deployment of the AADM.

Composed of

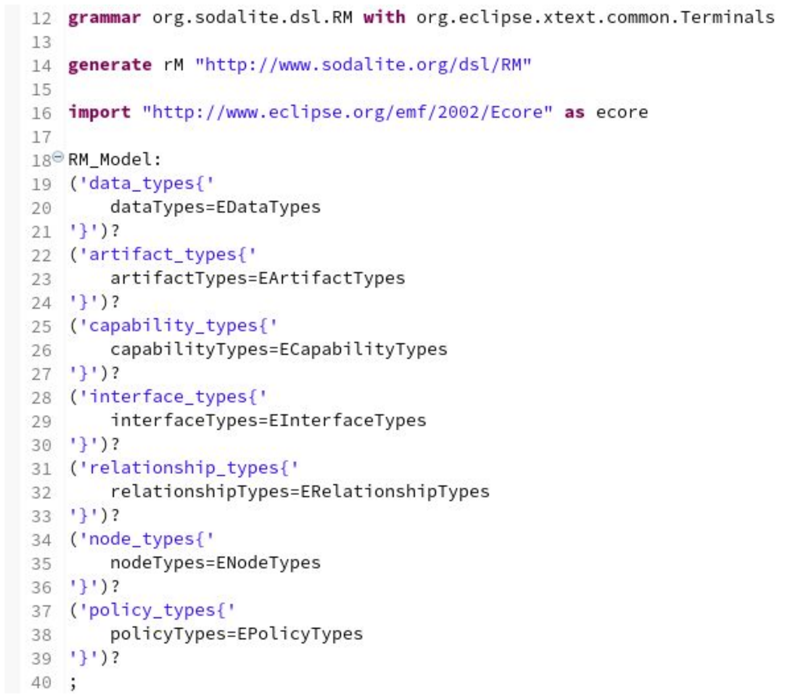
The IDE consists of the following components:

- DSLs for AADM and RM: both DSLs have been implemented as Eclipse EMF Ecore meta-models using the XText framework. For both DSLs, we have defined their corresponding grammar. Figure 34 shows a snippet of the AADM grammar. Figure 35 shows a snippet of the RM grammar.



```
grammar org.sodalite.dsl.AADM with org.eclipse.xtext.common.Terminals
import "http://www.sodalite.org/dsl/RM" as rm

generate aADM "http://www.sodalite.org/dsl/AADM"

AADM_Model:
    ('node_templates{'
        nodeTemplates=ENodeTemplates
    '}')?
;

ENodeTemplates:
    {ENodeTemplates}(nodeTemplates+=ENodeTemplate)+
;

ENodeTemplate:
    name = ID '{'
    ('type:' type=QUALIFIED_NAME)
    ('description:' description=STRING)?
    ('properties{'
        properties=EProperties
        '}')?
    ('attributes{'
        atributes=EAttributes
        '}')?
    ('requirements{'
        requirements=ERequirements
        '}')?
"}";
```

Figure 34. Snippet of AADM grammar



```
grammar org.sodalite.dsl.RM with org.eclipse.xtext.common.Terminals

generate rM "http://www.sodalite.org/dsl/RM"

import "http://www.eclipse.org/emf/2002/Ecore" as ecore

RM_Model:
('data_types{'
    dataTypes=EDataTypes
'}')?
('artifact_types{'
    artifactTypes=EArtifactTypes
'}')?
('capability_types{'
    capabilityTypes=ECapabilityTypes
'}')?
('interface_types{'
    interfaceTypes=EInterfaceTypes
'}')?
('relationship_types{'
    relationshipTypes=ERelationshipTypes
'}')?
('node_types{'
    nodeTypes=ENodeTypes
'}')?
('policy_types{'
    policyTypes=EPolicyTypes
'}')?
;
```

Figure 35. Snippet of RM grammar

- DSL Editor: XText framework automatically generates a rich-featured textual editor for both DSLs out of their grammar.

Roles that interact with the component (i.e. App Expert, ResExpert)

The IDE is the main SODALITE user interface for the following roles:

- Application Ops Expert (AOE): this role uses the IDE to define an abstract application deployment model (see UC1 in D2.1) and to assign AADM nodes to target resources (see UC2 in D2.1).

- Resource Expert (RE): this role uses the IDE to define new resource patterns in a RM and assign them to existing resources (see UC12 in D2.1).

Depends on

The IDE depends on the following SODALITE components:

- KB: this semantic repository contains information about reusable resources and types required in the definition of AADM whose specification lies on RMs.
- Semantic Reasoner API: this is the interface between the IDE and the KB. It provides a predefined set of queries to retrieve information about available resources and types from the KB, which can be reused in the definition of AADM within the IDE editor. It also supports the storage of AADMs into the KB. The IDE also uses this interface to request the semantic validation of both AADMs and RMs. As a result of this interaction, the IDE can be notified about model building inconsistencies and other recommendations. They will be presented to the user next to the model entities affected by the validation error or by the recommendation.
- IaC Verifier (IaCVerificationAPI): before an AADM can be deployed, the IDE will request this component to verify the conformance of the AADM with the deployment process. As a result of this interaction, the IDE can be notified about model building errors, including topology and provisioning workflow verification errors. They will be presented to the AOE next to the AADM entities affected by the verification error.
- IaC Blueprint builder/Abstract Model Parser (DeploymentPreparationAPI): the IDE will request this component of the IaC layer to request the deployment of an AADM.
- Bug Predictor and Fixer (DefectPredictionCorrectionAPI): the IDE will request this component to predict bugs in the AADM. As a result of this interaction, the IDE can be notified about model building errors and bugs. They will be presented to the AOE next to the AADM entities affected by the verification error. Complementing the reported bugs, this component may offer some bug fixes. They will offer to the AOE in the IDE editor, within the bug report. If the AOE accepts to apply any fixes, the IDE will request its application to this component.

Repositories

The IDE component is located in the folder *dsl* of the repository *ide* in the SODALITE GitHub portal: https://github.com/SODALITE-EU/ide

The structure of this repository is as follows:

- org.sodalite.dsl.AADM.parent: this is the parent Maven project for the AADM DSL and Editor component.
    - org.sodalite.dsl.AADM.feature: defines the Eclipse feature for the AADM DSL and Editor component.
    - org.sodalite.dsl.AADM.ide: creates the platform-independent IDE functionality for the AADM editor plugin.
    - org.sodalite.dsl.AADM.repository: provides an Eclipse update-site repository to install the AADM Editor.
    - org.sodalite.dsl.AADM.target: defines the Eclipse target (e.g. dependencies) required by the ADDM editor.
    - org.sodalite.dsl.AADM.tests: defines unit tests for the AADM language.
    - org.sodalite.dsl.AADM.ui.tests: define unit tests for the ADDM editor.
    - org.sodalite.dsl.AADM.ui: provides AADM Editor functionality and other contributions to the Eclipse workbench.
    - org.sodalite.dsl.AADM.web: provides support to embed the AADM editor within a Web browser.

    - org.sodalite.dsl.AADM: defines the AADM grammar.
- org.sodalite.dsl.RM.ide: creates the platform-independent IDE functionality for the RM editor plugin.
- org.sodalite.dsl.RM.ui: provides RM Editor functionality and other contributions to the Eclipse workbench.
- org.sodalite.dsl.RM: defines the RM grammar.
- org.sodalite.dsl.kb_reasoner_client: provides a Java client for the Semantic Reasoner API, which is used by both the AADM and RM IDE editors to send requests to the RB Reasoner.

The entire IDE DSL component, which encompasses the AADM and RM DSL and editors plugins, is managed by Maven, so that the Eclipse update site for the SODALITE IDE could be built within the SODALITE CI/CD pipeline. Through this update site, SODALITE AOEs and RMs could easily install the IDE locally. See README.md file in SODALITE IDE repository for instructions to build the update site and install the IDE within Eclipse.

7 Conclusion

This deliverable presented the current version of the SODALITE Semantic Modelling layer, which is relevant to T3.1 “Application Semantic Modelling” and T3.2 “Infrastructure Semantic Modelling”. More specifically, the key meta-modelling principles and the SODALITE ODP have been presented, which facilitate semantic representation of TOSCA-based cloud applications and cloud infrastructures. In addition, the first version of the Semantic Reasoner has been described for populating the KB and providing a REST API that different modules use to get information from the KB. The module also provides the reasoning infrastructure to support entity search and matchmaking, enabling the implementation of recommendation services and to support semantic validation of the submitted AADLs. Finally, we described the specifics of the SODALITE IDE that allows end users to define AADMs by reusing components and resources from the KB.

Next steps include further enrichments and enhancements of the SODALITE ontology-based framework in three main directions. First, to update the semantic models (and the KB population services) in order to support the representation of additional information based on the use case and component requirements. Second, to extend the provided REST API in order to support additional searching and reuse capabilities. Finally, the IDE will improve support for AADM and RM modeling, including extensions to support additional TOSCA types and templates, simplified modelling for AOEs and REs, support for semantic and optimization validation of AADMs and RMs and hotfix application, wizards for AADM/RM persistences within the KB and AADM deployment within the IaC layer.

8 References

[1] A. Gangemi and V. Presutti, “Ontology Design Patterns,” in *Handbook on Ontologies*, 2009.

[2] R. Fischer and C. Janiesch, “A method to classify standards in emerging technologies: The case of cloud computing,” in *ECIS 2014 Proceedings - 22nd European Conference on Information Systems*, 2014.

[3] H. Brabra, A. Mtibaa, F. Petrillo, P. Merle, L. Sliman, et al., “On semantic detection of cloud API (anti)patterns”. Information and Software Technology, Elsevier, vol. 107, pp.65 - 82, 2019.

[4] T.Rebele, F.Suchanec, J. Hoffart, et al., “YAGO: A Multilingual Knowledge Base from Wikipedia, Wordnet, and Geonames”, *In Proceedings of International Semantic Web Conference*, pp. 177-185, Springer, 2016.

[5] T. R. Gruber, “A translation approach to portable ontology specifications,” *Knowl. Acquis.*, 1993.

[6] R. Studer, V. R. Benjamins, and D. Fensel, “Knowledge engineering: Principles and methods,” *Data Knowl. Eng.*, vol. 25, no. 1–2, pp. 161–197, Mar. 1998.

[7] F. van H. Deborah L. McGuinness, “Owl web ontology language overview,” *W3C Recomm. 10.2004-03*, 2004.

[8] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[9] K. Hammar, *Content Ontology Design Patterns: Qualities, Methods, and Tools*, vol. 1879. Linköping University Electronic Press, 2017.

[10] M. Y. Vardi, “Why is modal logic so robustly decidable?,” in *Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop*, 1996.

[11] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler, “OWL2: The next step for OWL,” *J. Web Semant.*, vol. 6, no. 4, pp. 309–322, 2008.

[12] D. Kourtesis, J. M. Alvarez-Rodríguez, and I. Paraskakis, “Semantic-based QoS management in cloud systems: Current status and future challenges,” *Futur. Gener. Comput. Syst.*, 2014.

[13] F. Chen, X. Bai, and B. Liu, “Efficient service discovery for cloud computing environments,” in *International Conference on Computer Science and Information Engineering*, pp. 443–448, 2011.

[14] W. T. Tsai, X. Sun, and J. Balasooriya, “Service-oriented cloud computing architecture,” in *ITNG2010 - 7th International Conference on Information Technology: New Generations*, 2010.

[15] N. Bassiliades, M. Symeonidis, G. Meditskos, E. Kontopoulos, P. Gouvas, and I. Vlahavas, “A semantic recommendation algorithm for the PaaSport platform-as-a-service marketplace,” *Expert Syst. Appl.*, vol. 67, pp. 203–227, Jan. 2017.

[16] N. Bassiliades, M. Symeonidis, G. Meditskos, E. Kontopoulos, P. Gouvas, and I. Vlahavas, “PaaSport Semantic Model: An Ontology for a Platform-as-a-Service Semantically Interoperable Marketplace”, *in Data and Knowledge Engineering*, vol. 113, pp. 81-115, Elsevier, 2018.

[17] B.Martino, A.Esposito, S. Nacchia, S. Maisto, U. Breitenbücher, “An Ontology for OASIS TOSCA”, *Advances in Intelligent Systems and Computing*, vol. 1150, pp. 709-719, Springer 2020.

[18] A. Willner, M. Giatili, P.Grosso, C.Papagianni, M.Morsey, I.Baldin. “Using semantic web technologies to query and manage information within federated cyber-infrastructures.” *Information*, pp. 1–26, 2017.

[19] S. Challita, F. Paraiso, and P. Merle, “Towards Formal-based Semantic Interoperability in

Multi-Clouds", 10th *IEEE International Conference on Cloud Computing (CLOUD),* pp. 710–713, June 2017.

[20] A. Zhou, K. Ren, X. Li, W. Zhang, X. Ren, "Building Quick Resource Index List Using WordNet and High-Performance Computing Resource Ontology towards Efficient Resource Discovery", *In Proceedings of 21st IEEE International Conference on High Performance Computing and Communications ,17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems,* pp. 885- 892, 2019.

[21] D. Andročec, and N. Vrček, "Methodology for detection of cloud Interoperability problems.", *International Journal of Electrical and Computer Engineering Systems*, vol. 7, pp. 53-59, 2016.

[22] D. Andročec, and N. Vrček, "Ontologies for platform as service APIs interoperability.", *Cybernetics and Information Technologies*, vol. 16, pp. 29-44, 2016

[23] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist*. 2011.

[24] B. Glimm, S. Rudolph, and J. Völker, "Integrated metamodeling and diagnosis in OWL 2," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010.

[25] B. Motik, "On the properties of metamodeling in OWL," *J. Log. Comput.*, 2007.

[26] N. Jekjantuk, G. Gröner, and J. Z. Pan, "Modelling and reasoning in metamodelling enabled ontologies," in *Knowledge Science, Engineering and Management*, Springer, 2010, pp. 51–62.

[27] T. Tudorache, "Ontology Engineering: Current State, Challenges, and Future Directions," 2019.

[28] C. M. Keet, "An introduction to ontology," *Choice Rev. Online*, vol. 51, no. 04, pp. 51-2000-51–2000, 2013.

[29] A. Gangemi, "Ontology design patterns for semantic web content," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2005.

[30] A. Gangemi and P. Mika, "Understanding the Semantic Web through Descriptions and Situations," in *Proceedings of ODBASE03 Conference*, Springer, Berlin, Heidelberg, pp. 689–706, 2003.

[31] N. Guarino, A. Oltramari, L. Schneider, C. Masolo, and A. Gangemi, "Sweetening Ontologies with DOLCE," 2007.

[32] S. Erdweg, T. Van Der Storm, M. Völter, M.Boersma, R.Bosman, W. R. Cook. et al , "The state of the art in language workbenches.", *In International Conference on Software Language Engineering,* pp. 197-217, Springer, 2013.

[33] B.Langlois, C.E.Jitia, and E.Jouenne, "DSL classification". In OOPSLA 7th workshop on domain specific modeling, October 2007.

[34] M. Pfeiffer, and J.Pichler, "A comparison of tool support for textual domain-specific languages.", *In Proceedings of the 8th OOPSLA Workshop on Domain-Specific Modeling*, pp. 1-7, October 2008.

[35] B. Merkle, "Textual modeling tools: overview and comparison of language workbenches", *In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 139-148, October 2010.

[36] M. Eysholdt, and H.Behrens, " Xtext: implement your language faster than the quick and dirty way.", *In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 307-309, October 2010.

[37] F.Jouault, J.Bézivin, and I.Kurtev, "TCS: a DSL for the specification of textual concrete syntaxes in model engineering.", *In Proceedings of the 5th international conference on*

Generative programming and component engineering, pp. 249-254, October 2006.

[38] F. Heidenreich, J. Johannes, S.Karol, M.Seifert, and C.Wende, "Model-based language engineering with EMFText.", *In International Summer School on Generative and Transformational Techniques in Software Engineering*, pp. 322-345, Springer, July 2011.

[39] H. Krahn, B. Rumpe, and S.Völkel, "MontiCore: a framework for compositional development of domain specific languages.", *International journal on software tools for technology transfer*, vol.12, pp.353-372, 2010, Online: http://www.monticore.de/

[40] P. Klint, T. Van Der Storm, J. J. Vinju, "RASCAL: A domain specific language for source code analysis and manipulation.", In SCAM, pp. 168–177, IEEE, 2009.

[41] L.C.L Kats, E. Visser, "The Spoofax language workbench: Rules for declarative specifica-tion of languages and IDEs.", *In OOPSLA*, pp. 444–463, ACM, 2010.

[42] V. Vujović, M. Maksimović, and B. Perišić, "Comparative analysis of DSM graphical editor frameworks: Graphiti vs. Sirius", *In Proceedings of the 23rd International Electrotechnical and Computer Science Conference (ERK'14)* , 2014.

[43] V. Vujović, M. Maksimović, and B. Perišić, "Sirius: A rapid development of a DSM graphical editor.", *In IEEE 18th International Conference on Intelligent Engineering Systems INES*, pp. 233-238, 2014.

[44] D. Moody, "What makes a good diagram? Improving the cognitive effectiveness of diagrams in IS development," *15th international conference of Information Systems Development*, Springer, 2006.

[45] S. Meliá, C. Cachero, J. M. Hermida, and E. Aparicio, "Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study.", *Software Quality Journal*, vol. 24, pp. 709-735, 2016.

[46] "Oasis topology and orchestration specification for cloud applications version 1.0", November 2013, http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

[47] M. Carlson, M. Chapman, A. Heneveld, S. Hinkelman, D. Johnston-Watt, A. Karmarkar, T. Kunze, A. Malhotra, J. Mischkinsky, A.Otto, V.Pandey, G.Pilz, Z.Song,and P. Yendluri, "Cloud Application Management for Platforms," December 2012. Available: https://www.oasis-open.org/committees/download.php/47278/CAMP-v1.0.pdf

[48] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, " Cloudcamp: A model-driven generative approach for automating cloud application deployment and management.", *Tech. Rep. ISIS-17-105, Vanderbilt University, Nashville, TN, USA*, 2017.

[49] N. Ferry, A. Rossini, F. Chauvel, B. Morin, A. Solberg, (2013, June). "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems.", *In IEEE Sixth International Conference on cloud computing*, pp. 887-894, June 2013.

[50] M. Palyart, D. Lugato, I. Ober, and J.M. Bruel, "MDE4HPC: an approach for using model-driven engineering in high-performance computing.", *In International SDL Forum*, pp. 247-261, Springer, July 2011.

[51] M. Śmiałek, K. Rybiński, R. Roszczyk, and K. Marek, " Towards a Unified Requirements Model for Distributed High Performance Computing.", *In Data-Centric Business and Applications,* pp. 1-20, Springer, 2020.

[52] U.Breitenbücher, T. Binz, O. Kopp, F. Leymann, F., and D. Schumm, "Vino4TOSCA: A visual notation for application topologies based on TOSCA. ", *In OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pp. 416-424, Springer, September 2012.

[53] O. Kopp, T. Binz, U. Breitenbücher, and F.Leymann, "Winery–a modeling tool for

TOSCA-based cloud applications. “, *In International Conference on Service-Oriented Computing*, pp. 700-704, Springer, December 2013.

[54] P. Štefanic, M. Cigale, F. Q. Fernandez, D. Rogers, L. Knight, A. C. Jones, and I.Taylor, “TOSCA-based SWITCH Workbench for application composition and infrastructure planning of time-critical applications”, *In conference of the 3rd edition in the series of workshop on Interoperable infrastructures for interdisciplinary big data sciences*, Zenodo, 2018.

[55] T. Tudorache, “Ontology engineering: Current state, challenges, and future directions”, *Semantic Web – Interoperability, Usability, Applicability an IOS Press Journal*, vol. 11, no. 1, pp. 125-138, 2019.

[56] B. Neumayr and M. Schrefl, “Multi-level conceptual modeling and OWL”, *In Proceedings of International Conference on Conceptual Modeling,* vol. 5833, pp. 189-199, 2009.

[57] B. Motik, “On the properties of metamodeling in OWL”, *Journal of Logic and Computation,* vol. 17, no. 4, pp. 617-637, 2007.

9 Appendix

We present in this section example definitions of the snow use case. More specifically, we present the DSL syntax for the node templates, the intermediate exchange model and the final SODALITE meta-model definition. The TOSCA representation of the node templates is given below.

```
topology_template:
  node_templates:
    vm:
      type: sodalite.nodes.VM.OpenStack
      properties:
        name: snow-vm
        image: centos7
        flavor: m1.small
        network: 87b57656-381c-4921-82c0-bd65a8a67cfd
        security_groups: default,snow
        key_name: draganrX

    docker-host:
      type: sodalite.nodes.DockerHost
      properties:
        registry_ip: 154.48.185.207
      requirements:
        - host: vm

    skyline-extractor:
      type: sodalite.nodes.DockerizedComponent
      properties:
        image_name: snow-skyline-extractor
        ports: 8080:8080
        exposed_ports: 8080
      requirements:
        - host: docker-host

    skyline-alignment:
      type: sodalite.nodes.DockerizedComponent
      properties:
        image_name: snow-skyline-alignment
        ports: 8081:8080
        exposed_ports: 8080
      requirements:
        - host: docker-host
```

DSL syntax

```
node_templates{

     vm{
        type: sodalite.nodes.VM.OpenStack
        properties{
            name: 'snow-vm'
            image: 'centos7'
            flavor: 'm1.small'
            network: '87b57656-381c-4921-82c0-bd65a8a67cfd'
            security_groups: 'default,snow'
            key_name: 'draganrX'
        }
    }

    docker_host{
        type: sodalite.nodes.DockerHost
        properties{
            registry_ip: '154.48.185.207'
        }
        requirements{
            host{
                node: vm
            }
        }
    }

    skyline_extractor{
         type: sodalite.nodes.DockerizedComponent
         properties{
            image_name: 'snow-skyline-extractor'
            ports: '8080:8080'
            exposed_ports: '8080'
         }
         requirements{
            host{
                node: docker_host
            }
         }
    }

    skyline_alignment{
        type: sodalite.nodes.DockerizedComponent
        properties{
            image_name: 'snow-skyline-alignment'
            ports: '8081:8080'
            exposed_ports: '8080'
        }
        requirements{
            host{
                node: docker_host
            }
        }
    }
}
```

Exchange Model

```
# baseURI: https://www.sodalite.eu/ontologies/exchange0/
# imports: https://www.sodalite.eu/ontologies/exchange/
@prefix : <https://www.sodalite.eu/ontologies/exchange0/> .
@prefix exchange: <https://www.sodalite.eu/ontologies/exchange/> .
@prefix exchange0: <https://www.sodalite.eu/ontologies/exchange0#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:
  rdf:type owl:Ontology ;
  owl:imports exchange: ;
  owl:versionInfo "Created by the SODALITE IDE" ;
.

:AADM_1
  rdf:type exchange:AADM ;
  exchange:userId "27827d44-0f6c-11ea-8d71-362b9e155667" ;
.

:Property_1
  rdf:type exchange:Property ;
  exchange:name "name" ;
  exchange:value "snow-vm" ;
.
:Property_2
  rdf:type exchange:Property ;
  exchange:name "image" ;
  exchange:value "centos7" ;
.
:Property_3
  rdf:type exchange:Property ;
  exchange:name "flavor" ;
  exchange:value "m1.small" ;
.
:Property_4
  rdf:type exchange:Property ;
  exchange:name "network" ;
  exchange:value "87b57656-381c-4921-82c0-bd65a8a67cfd" ;
.
:Property_5
```

```
  rdf:type exchange:Property ;
  exchange:name "security_groups" ;
  exchange:value "default,snow" ;
.
:Property_6
  rdf:type exchange:Property ;
  exchange:name "key_name" ;
  exchange:value "draganrX" ;
.
:Property_7
  rdf:type exchange:Property ;
  exchange:name "registry_ip" ;
  exchange:value "154.48.185.207" ;
.
:Property_8
  rdf:type exchange:Property ;
  exchange:name "image_name" ;
  exchange:value "snow-skyline-extractor" ;
.
:Property_9
  rdf:type exchange:Property ;
  exchange:name "ports" ;
  exchange:value "8080:8080" ;
.
:Property_10
  rdf:type exchange:Property ;
  exchange:name "exposed_ports" ;
  exchange:value "8080" ;
.
:Property_11
  rdf:type exchange:Property ;
  exchange:name "image_name" ;
  exchange:value "snow-skyline-alignment" ;
.
:Property_12
  rdf:type exchange:Property ;
  exchange:name "ports" ;
  exchange:value "8081:8080" ;
.
:Property_13
  rdf:type exchange:Property ;
  exchange:name "exposed_ports" ;
  exchange:value "8080" ;
.

:Parameter_1
  rdf:type exchange:Parameter ;
  exchange:name "node" ;
  exchange:value "vm" ;
.

:Requirement_1
  rdf:type exchange:Requirement ;
  exchange:name "host" ;
  exchange:hasParameter :Parameter_1 ;
.
:Parameter_2
  rdf:type exchange:Parameter ;
  exchange:name "node" ;
  exchange:value "docker_host" ;
.

:Requirement_2
  rdf:type exchange:Requirement ;
  exchange:name "host" ;
  exchange:hasParameter :Parameter_2 ;
.

:Parameter_3
  rdf:type exchange:Parameter ;
  exchange:name "node" ;
  exchange:value "docker_host" ;
.

:Requirement_3
  rdf:type exchange:Requirement ;
  exchange:name "host" ;
  exchange:hasParameter :Parameter_3 ;
.

:Template_1
  rdf:type exchange:Template ;
  exchange:name "vm" ;
  exchange:type "sodalite.nodes.VM.OpenStack" ;
  exchange:properties :Property_1 ;
  exchange:properties :Property_2 ;
  exchange:properties :Property_3 ;
  exchange:properties :Property_4 ;
  exchange:properties :Property_5 ;
  exchange:properties :Property_6 ;
.
:Template_2
  rdf:type exchange:Template ;
  exchange:name "docker_host" ;
  exchange:type "sodalite.nodes.DockerHost" ;
  exchange:properties :Property_7 ;
  exchange:requirements :Requirement_1 ;
.
:Template_3
  rdf:type exchange:Template ;
  exchange:name "skyline_extractor" ;
  exchange:type "sodalite.nodes.DockerizedComponent" ;
  exchange:properties :Property_8 ;
  exchange:properties :Property_9 ;
  exchange:properties :Property_10 ;
  exchange:requirements :Requirement_2 ;
.
:Template_4
  rdf:type exchange:Template ;
  exchange:name "skyline_alignment" ;
  exchange:type "sodalite.nodes.DockerizedComponent" ;
  exchange:properties :Property_11 ;
  exchange:properties :Property_12 ;
  exchange:properties :Property_13 ;
  exchange:requirements :Requirement_3 ;
.
```

SODALITE Meta-model

```
@prefix dul: <http://www.loa-cnr.it/ontologies/DUL.owl#> .
@prefix soda: <https://www.sodalite.eu/ontologies/sodalite-metamodel/> .
@prefix tosca: <https://www.sodalite.eu/ontologies/tosca/> .
@prefix ws: <https://www.sodalite.eu/ontologies/workspace/1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ws:AADM_a0p0o8iq4enfgvepasnlh0193r a soda:AbstractApplicationDeployment;
  soda:createdAt "2020-01-27T09:42:02.579+02:00";
  soda:createdBy ws:27827d44-0f6c-11ea-8d71-362b9e155667;
          soda:includesTemplate     ws:docker_host,     ws:skyline_alignment,
ws:skyline_extractor,
    ws:vm .

ws:27827d44-0f6c-11ea-8d71-362b9e155667 a soda:User .

ws:vm                                                                      a
<https://www.sodalite.eu/ontologies/snow/tier1/sodalite.nodes.VM.OpenStack>;
  soda:hasContext ws:Desc_6ee6c02gj5kcb3gl3nt6e3hgsq .

ws:Desc_6ee6c02gj5kcb3gl3nt6e3hgsq a soda:SodaliteDescription;
            tosca:properties      ws:PropClassifer_5fbmfmjehp3bfjfcgqk5qjv88o,
ws:PropClassifer_5phkcn69cctahuu3lboh5aut3i,
                                  ws:PropClassifer_emp5em4jdt7eupotnh9gi9pfih,
ws:PropClassifer_hndt50uf4kpiqt0oblk3kgt5jq,
                                  ws:PropClassifer_kafsrlfpo9smdktcp5nbti9kn5,
ws:PropClassifer_sei8blnl0ludk8gsh16u354elr .

ws:PropClassifer_sei8blnl0ludk8gsh16u354elr a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/image>;
  tosca:hasDataValue "centos7" .

ws:PropClassifer_5phkcn69cctahuu3lboh5aut3i a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/flavor>;
  tosca:hasDataValue "m1.small" .

ws:PropClassifer_5fbmfmjehp3bfjfcgqk5qjv88o a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/hpc/tier1/job_name>;
  tosca:hasDataValue "snow-vm" .

ws:PropClassifer_hndt50uf4kpiqt0oblk3kgt5jq a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/key_name>;
  tosca:hasDataValue "draganrX" .

ws:PropClassifer_emp5em4jdt7eupotnh9gi9pfih a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/network>;
  tosca:hasDataValue "87b57656-381c-4921-82c0-bd65a8a67cfd" .

ws:PropClassifer_kafsrlfpo9smdktcp5nbti9kn5 a tosca:Property;
                                                              dul:classifies
<https://www.sodalite.eu/ontologies/snow/tier1/security_groups>;
  tosca:hasDataValue "default,snow" .

ws:docker_host                                                             a
<https://www.sodalite.eu/ontologies/snow/tier1/sodalite.nodes.DockerHost>;
  soda:hasContext ws:Desc_4e5rjkgnb0qbmua96sp16ggfq3 .

ws:Desc_4e5rjkgnb0qbmua96sp16ggfq3 a soda:SodaliteDescription;
  tosca:properties ws:PropClassifer_ba1pu92n6r06uj291g70c3eask;
  tosca:requirements ws:ReqClassifier_m0819bq22rvhvdbtmt3m7bc463 .

ws:PropClassifer_ba1pu92n6r06uj291g70c3eask a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/registry_ip>;
  tosca:hasDataValue "154.48.185.207" .

ws:ReqClassifier_m0819bq22rvhvdbtmt3m7bc463 a tosca:Requirement;
  <dul:hasParameter> ws:ParamClassifier_6rbsqgigpnvm3a4h6r7ee2p7b4;
  dul:classifies tosca:host .

ws:ParamClassifier_6rbsqgigpnvm3a4h6r7ee2p7b4 a soda:SodaliteParameter;
  dul:classifies tosca:node;
  tosca:hasObjectValue ws:vm .
```

```
ws:skyline_extractor                                                     a
<https://www.sodalite.eu/ontologies/snow/tier1/sodalite.nodes.DockerizedCompon
ent>;
  soda:hasContext ws:Desc_voma07qtumg42hve9d76271q95 .

ws:Desc_voma07qtumg42hve9d76271q95 a soda:SodaliteDescription;
            tosca:properties      ws:PropClassifer_9mn53ed1osno6472uhkq5uj65v,
ws:PropClassifer_ee5c4meam7bgg4jib3dp7kpvi4,
    ws:PropClassifer_nu90hqd8b7iqpk72o5cn7ro85;
  tosca:requirements ws:ReqClassifier_an1gc6knsnp8a9lfpjj3rmc30a .

ws:PropClassifer_ee5c4meam7bgg4jib3dp7kpvi4 a tosca:Property;
                                                              dul:classifies
<https://www.sodalite.eu/ontologies/snow/tier1/exposed_ports>;
  tosca:hasDataValue "8080"^^xsd:int .

ws:PropClassifer_9mn53ed1osno6472uhkq5uj65v a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/image_name>;
  tosca:hasDataValue "snow-skyline-extractor" .

ws:PropClassifer_nu90hqd8b7iqpk72o5cn7ro85 a tosca:Property;
                                                              dul:classifies
<https://www.sodalite.eu/ontologies/snow/tier1/exposed_ports>;
  tosca:hasDataValue "8080:8080" .

ws:ReqClassifier_an1gc6knsnp8a9lfpjj3rmc30a a tosca:Requirement;
  <dul:hasParameter> ws:ParamClassifier_45774i7h3qg3r34j4oqt10odi3;
  dul:classifies tosca:host .

ws:ParamClassifier_45774i7h3qg3r34j4oqt10odi3 a soda:SodaliteParameter;
  dul:classifies tosca:node;
  tosca:hasObjectValue ws:docker_host .

ws:skyline_alignment                                                     a
<https://www.sodalite.eu/ontologies/snow/tier1/sodalite.nodes.DockerizedCompon
ent>;
  soda:hasContext ws:Desc_8jsgm0vqebem77jujrl7vb16m2 .

ws:Desc_8jsgm0vqebem77jujrl7vb16m2 a soda:SodaliteDescription;
            tosca:properties      ws:PropClassifer_6l5jhg01kr9va3rk3962jh9rsn,
ws:PropClassifer_sa3gd9pqghp219ecp1nd7ifphc,
    ws:PropClassifer_t37gv7otfqjvlc2pq4fblcooqs;
  tosca:requirements ws:ReqClassifier_11a12mvma6tifioe5hilrsr1hk .

ws:PropClassifer_sa3gd9pqghp219ecp1nd7ifphc a tosca:Property;
                                                              dul:classifies
<https://www.sodalite.eu/ontologies/snow/tier1/exposed_ports>;
  tosca:hasDataValue "8080"^^xsd:int .
```

```
ws:PropClassifer_t37gv7otfqjvlc2pq4fblcooqs a tosca:Property;
                                                              dul:classifies
<https://www.sodalite.eu/ontologies/snow/tier1/exposed_ports>;
  tosca:hasDataValue "8081:8080" .

ws:PropClassifer_6l5jhg01kr9va3rk3962jh9rsn a tosca:Property;
  dul:classifies <https://www.sodalite.eu/ontologies/snow/tier1/image_name>;
  tosca:hasDataValue "snow-skyline-alignment" .

ws:ReqClassifier_11a12mvma6tifioe5hilrsr1hk a tosca:Requirement;
  <dul:hasParameter> ws:ParamClassifier_fkigm15qcibjcpldcdtebd2qn2;
  dul:classifies tosca:host .

ws:ParamClassifier_fkigm15qcibjcpldcdtebd2qn2 a soda:SodaliteParameter;
  dul:classifies tosca:node;
  tosca:hasObjectValue ws:docker_host .
```