Software and Systems Modeling (2026) 25:515–548
https://doi.org/10.1007/s10270-025-01294-1

REGULAR PAPER

A knowledge-based approach for guided development of Infrastructure as Code

Zoe Vasileiou[1] · Indika Kumara[2] · Georgios Meditskos[1] · Kamil Tokmakov[4] · Dragan Radolović[5] · Jesús Gorroñogoitia Cruz[6] · Elisabetta Di Nitto[7] · Damian Andrew Tamburri[3] · Willem-Jan Van Den Heuvel[2] · Stefanos Vrochidis[8]

Received: 24 January 2024 / Revised: 16 April 2025 / Accepted: 21 April 2025 / Published online: 23 June 2025

Abstract

Infrastructure as Code (IaC) uses versionable software code to define, deploy, and configure physical computational resources, software execution platforms, and applications. As a result, IaC enables the scalable management of complex computing environments while preventing environment drift. IaC frameworks typically offer specific languages such as the industrial Terraform, Ansible, Chef, or TOSCA—standing for Topology and Orchestration Specification for Cloud Applications—the OASIS (Organization for the Advancement of Structured Information Standards) open standard approach to IaC. Developing high-quality IaC for deploying and managing applications demands expertise and knowledge in specific IaC languages, infrastructure resources, resource providers, quality issues in IaC scripts, and so on. While several model-driven engineering (MDE) approaches have been proposed to simplify IaC development, they cannot capture and use expert knowledge to assist with modeling tasks and MDE processes by providing interactive recommendations. This paper presents a knowledge-based framework for guiding the model-driven development of IaC. We use TOSCA as the target IaC language as it is an open standard. We enable IaC and resource experts to share their IaC and resource-related knowledge with application operational experts to help simplify the development of application deployment models. We use an ontology to record the relevant deployment knowledge and ontology reasoning to implement modeling guidance capabilities such as TOSCA model auto-completion, code smell and error detection, and model element matchmaking. We show the flexibility of our methodology by applying it to three industrial applications, covering cloud, edge, and HPC (High-Performance Computing) domains. Moreover, we also assess the use acceptance of our approach and framework by conducting controlled experiments with expert and non-expert IaC users. The results indicate that our method can simplify IaC development by providing appropriate recommendations.

Keywords Model-driven engineering · IaC · Recommendation system · TOSCA · Ontology · Knowledge graph · Semantic web

1 Introduction

The DevOps approach—blending equally and at the same level of the organization both development and operations people and assets—is the de-facto way for developing and managing complex software applications [20]. A vital enabler of the successful adoption of DevOps is the automation of application provisioning and management [68]. The software stack enabling such automation reflects *Infrastructure as Code (IaC)*, a paradigm for provisioning and managing a computing environment using the explicit definition of the environment in source code and applying software development principles, practices, and tools [32, 39]. Many IaC tools exist, each with specific capabilities, from building application images to resource provision, application deployment and orchestration, and managing configurations used by applications [32].

In our previous work, we investigated the key challenges that IaC practitioners face through a systematic gray literature study [32] and semi-structured interviews with practitioners [26] to reveal a lack of dedicated IDEs (Integrated

Communicated by Massimo Tisi.

European Commission Grant no. 825480 (H2020), SODALITE.

Extended author information available on the last page of the article

Development Environment) specifically designed to support the development, operation, and maintenance of IaC. Moreover, we reported that coding IaC requires specialized skills and knowledge of infrastructure design, system design, different infrastructure types (e.g., cloud, edge, and HPC (High-Performance Computing)), infrastructure providers, resource selection, good and bad practices, etc. Thus, practitioners need an IDE and interactive guidance throughout the IaC development lifecycle. Moreover, it is essential to promote capturing, sharing, retrieving, and using the relevant knowledge and empower collaboration within a cross-functional team of members such as application operational personnel, resource (cloud/edge/HPC) experts, and security experts [12].

Several recent surveys reviewed the research literature on the modeling and development of IaC [5, 15, 43, 48, 68]. Existing research studies have investigated topics such as domain-specific languages (DSLs), model-driven engineering (MDE) techniques, and quality assurance. While there are several studies on guided MDE (i.e., recommendation systems for MDE) in general [1, 38, 42, 51, 54], to the best of our knowledge, there is no such work for IaC development.

This paper presents an approach that can interactively guide users in developing high-quality IaC for deploying applications in heterogeneous environments. Based on the expertise and responsibilities of the users, we identify two distinct roles: Resource Experts (REs) and Application Operational Experts (*AppOps*). The resources include hardware and software computational resources (i.e., devices and execution environments in UML deployment models). REs know various types of resources, including their properties and usage/configuration constraints and thus can accurately model resources. AppOps know their applications intimately and can define the deployment models of the applications by reusing and instantiating the resources specified by the REs. We aim to simplify and speed up IaC development by enabling knowledge sharing within a cross-functional team and providing interactive coding recommendations. We use TOSCA (Topology and Orchestration Specification for Cloud Applications) [1] as the IaC language. TOSCA is an open standard. Hence, it can help avoid vendor lock-in and improve the portability of application deployments [44]. We use ontologies to capture the relevant domain knowledge, for example, TOSCA semantics, resource configurations, and good and bad practices of coding IaC programs. Decision support services primarily use ontological reasoning (but are not limited to) and offer capabilities such as TOSCA coding recommendation, resource discovery and matchmaking, and deployment model validation. We implemented the TOSCA blueprints for three industrial use cases using our guided IaC development approach and conducted control experiments with expert and non-expert user groups. The experiments with each group showed that users consider our approach very useful and have a high potential for adoption by practitioners.

In summary, we make the following contributions in this paper:

1. We propose a methodology for the guided development of TOSCA blueprints for deploying applications on hybrid infrastructures. REs can model resources, and AppOps can use those resource models to define application deployment models.
2. We support four kinds of guidance to reduce the complexity in authoring resource and application deployment models: resource matchmaking, model validation, code smell detection, and model auto-completion.
3. We validated our approach by applying it to the three industrial use cases.
4. We also assessed the user acceptance of our approach through control experiments with both experienced and novice IaC users.

We created and validated our approach within the SODALITE Horizon Europe project .[2] We published the implementation of the SODALITE framework as a handbook [18]. In addition, several key research outcomes have also been published, for example, runtime environment [33], IaC defect prediction [8, 34], and IaC misconfiguration taxonomy [41]. This paper focuses on the SODALITE methodology that supports the guided development of TOSCA-based deployment models.

The remainder of the paper is organized as follows. Section 2 uses a motivating scenario to illustrate the main requirements of a guided IaC development tool. Section 3 provides an overview of IaC and TOSCA. Section 4 presents our approach, while Sect. 5 provides implementation details. Section 6 presents the evaluation of our approach, including industrial use cases and user studies. Section 7 discusses the threats to validity and the mapping of our guidance types to the common MDE recommendation types. Section 8 reviews the existing research studies on IaC development, and Sect. 9 concludes the paper with an outlook for future work.

2 Motivating Scenario and General Requirements

This section features a case study from the SODALITE H2020 project, namely the *Snow* application, to illustrate

[1] https://www.oasis-open.org/news/announcements/tosca-simple-profile-in-yaml-v1-3-oasis-standard-published/.

[2] https://www.sodalite.eu/.

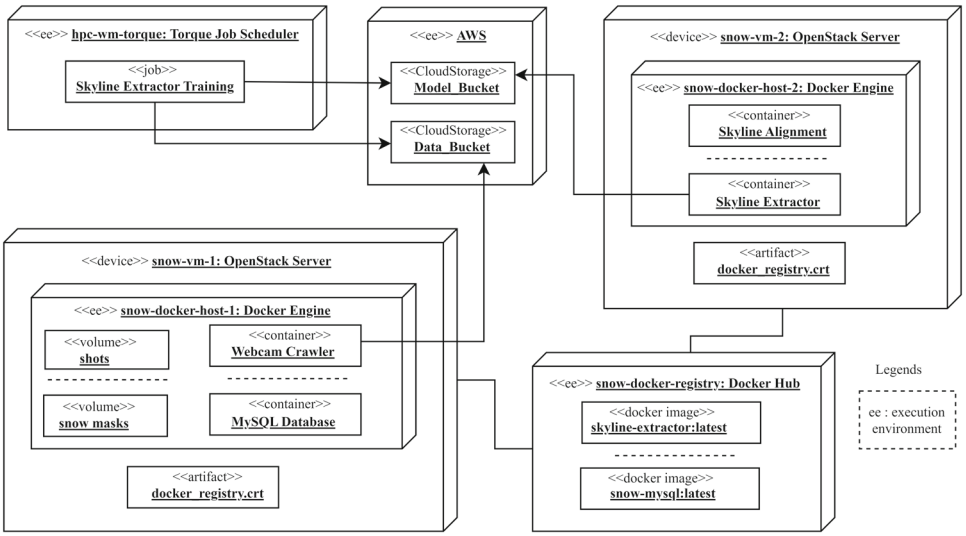


Fig. 1 Simplified UML deployment diagram for the snow use case

the key challenges of developing IaC to implement and manage complex heterogeneous applications. Figure 1 shows the UML deployment diagram of the snow application. The snow application uses crawled images from multiple data sources to predict the presence of snow in the mountains. It is an exemplary use case for a heterogeneous application as it uses diverse resources such as virtual machines (VMs), containers, storage, network, and HPC nodes from different resource providers. The components of the snow image processing data pipeline are containerized using Docker, and the corresponding container images are stored in the Docker Hub image registry. Two VMs from the OpenStack private cloud (a testbed) host the pipeline components. Among the components, the skyline extractor uses a deep learning model to detect snow masks. The corresponding model training pipeline is executed on an HPC cluster managed by the Torque resource manager. The end-users cannot directly provision HPC compute nodes and can only submit the jobs to be executed. The training data and models are in AWS (Amazon Web Services) storage buckets.

In collaboration, AppOps and REs must be able to create IaC scripts to provision resources and deploy the snow application. We envisage the following key requirements for a tool that can empower them in authoring IaC-based deployment models.

Req1: Guide AppOps through the complex task of IaC specification. When deploying the snow application in a given target deployment environment, AppOps need to select the resources from the deployment environment, provision them, deploy the individual components of the application onto the selected resources, and configure both resources and components. For example, for deploying the snow application, they need to create two VMs on the OpenStack testbed, install Docker in each, deploy the webcam crawler and MySQL database components as containers in one VM, and the skyline alignment and extractor components in the other VM. They must also create two storage buckets in AWS and schedule the skyline extractor model training job in the Torque resource manager.

AppOps can create IaC scripts to automate this deployment process. However, coding IaC requires specialized skills and knowledge, including infrastructure design, resource types (e.g., VM and HPC nodes), infrastructure providers (e.g., AWS and OpenStack), IaC languages (e.g., TOSCA and Ansible), good/bad IaC coding practices [32], and potential misconfigurations in IaC scripts [41]. Unfortunately, they may not possess these skills and knowledge [26]. Moreover, a try-and-fix way of developing IaC is not feasible due to the considerable overhead of testing IaC programs, which would require the actual software provisioning/configuration/operation. Hence, AppOps need a development environment that can interactively guide them in developing high-quality IaC programs.

Req2: Allow REs to share their knowledge about the deployment environment. The REs know the deployment environment and how to use it, including resources and their properties, possible configurations, and management policies. For example, an HPC expert knows how to configure an HPC cluster correctly and securely. A cloud expert has the experience and knowledge to select virtual machine types, create secure virtual networks, and define elasticity policies and constraints. Promoting capturing, sharing, retrieving, and using such knowledge is essential to help AppOps deploy and manage an application securely and optimally in complex deployment environments.

Consider our snow use case. The HPC testbed in the SODALITE project is managed by the High-Performance Computing Center (HLRS) at the University of Stuttgart. HLRS developers (i.e., REs) have the expertise to create and schedule HPC jobs securely and effectively in their compute

cluster. However, snow application developers (i.e., AppOps) do not possess this knowledge. To alleviate this knowledge gap, HLRS developers can define the IaC modules to access the HPC testbed securely, create HPC resources, and run HPC jobs, and then share those modules with application developers.

3 Background

This section introduces TOSCA and discusses IaC languages in general, highlighting their commonalities and differences.

3.1 TOSCA

TOSCA [3] is an OASIS (Organization for the Advancement of Structured Information Standards)[3] standard that aims to support the modeling, deployment, and management of cloud applications. OASIS is an international non-profit consortium that supports the collaborative development of open standards in various domains, including web services, cloud computing, and security.

Figure 2 shows a part of the TOSCA meta-model. With TOSCA, the application deployment blueprint is defined as a topology of nodes that host the application components [13, 44]. This topology is represented by the *Topology Template* consisting of *Node Templates* and *Relationship Templates*. The node templates describe software and hardware components, and the relationship templates describe the relationships among those components. The semantics (e.g., properties of the components and the operations that can be executed on the components) of the templates are defined by the *Node Types* and *Relationship Types*. Furthermore, different policies can be defined and attached to entities in a deployment model using *Policy Types*.

Figure 3 shows a snippet of a TOSCA blueprint. The node templates `snow-mysql`, `snow-docker-host`, and `snow-vm` represent a Dockerized MySQL database that stores snow images, a Docker engine that hosts the database container, and an OpenStack VM that hosts the Docker engine. The fourth node template `autoscale` defines the auto-scaling policy for the VM. The right part of the figure shows the corresponding `DockerizedComponent`, `DockerHost`, and `OpenStackVM` (custom) node types, and the `AutoScale` policy type. All node types inherit configurations such as properties, interfaces, and other relevant attributes from the `SoftwareComponent` node type from the TOSCA standard. In addition, the relationship property of `DockerizedComponent` indicates that a Dockerized component is hosted by a Docker engine/host.

[3] https://www.oasis-open.org/.

3.2 Comparing TOSCA with other IaC solutions

Many other IaC Solutions exist, such as Ansible,[4] Terraform,[5] Chef,[6] and Puppet.[7] Moreover, each IaC solution provides a domain-specific language (DSL), which may be tailor-made (e.g., TOSCA, Puppet, Ansible, and Terraform) or use an existing programming language (e.g., Chef). Ansible mainly uses a YAML-based DSL, Terraform provides a custom DSL, namely HCL (HashiCorp Configuration Language), and Chef uses a Ruby-based DSL. There are also IDEs to help IaC developers, either as standalone tools or as plug-ins for existing general-purpose IDEs.

We can categorize the languages used in IaC solutions broadly as declarative and imperative styles. The declarative style enables the developers to define the compute stack's intended state and leaves it up to the IaC framework to determine how to get there. The imperative style allows developers to specify the process that changes the stack's present state into the intended state as an ordered set of steps [32]. An IaC language can also have both declarative and imperative characteristics. TOSCA, Terraform, and Puppet are declarative, and Chef is imperative. Ansible IaC programs can have both imperative and declarative characteristics. For example, Ansible playbooks may consist of tasks that use declarative modules (e.g., service and apt) or imperative modules (e.g., shell and command). The order of the tasks in the playbook can determine the final state of the target environment.

While IaC languages have different notation and syntax, there are considerable similarities in their concepts and purposes [68]. Each IaC tool generally has a library of reusable and parametrized components that implement infrastructure management tasks such as creating a VM and installing a MySQL database. Tool providers and their communities typically develop and manage these components. Some examples are Ansible modules, TOSCA node types, and resource types in Terraform, Chef, and Puppet. When application developers create IaC scripts to deploy their applications, they use these components by defining the configuration parameters of the components. For example, a TOSCA blueprint for creating and configuring a VM is a set of node type instances (defined as configurations). For Terraform, Chef, and Puppet, it is a set of resource type instances; for Ansible, it is a set of module instances.

Within the scope of this paper, our objective is to propose an approach to enable interactive IaC coding/modeling assistance in a TOSCA-based IDE. We selected TOSCA since it is an open standard. However, due to the similarities in most

[4] https://www.ansible.com/.

[5] https://www.terraform.io/.

[6] https://www.chef.io/.

[7] https://www.puppet.com/.

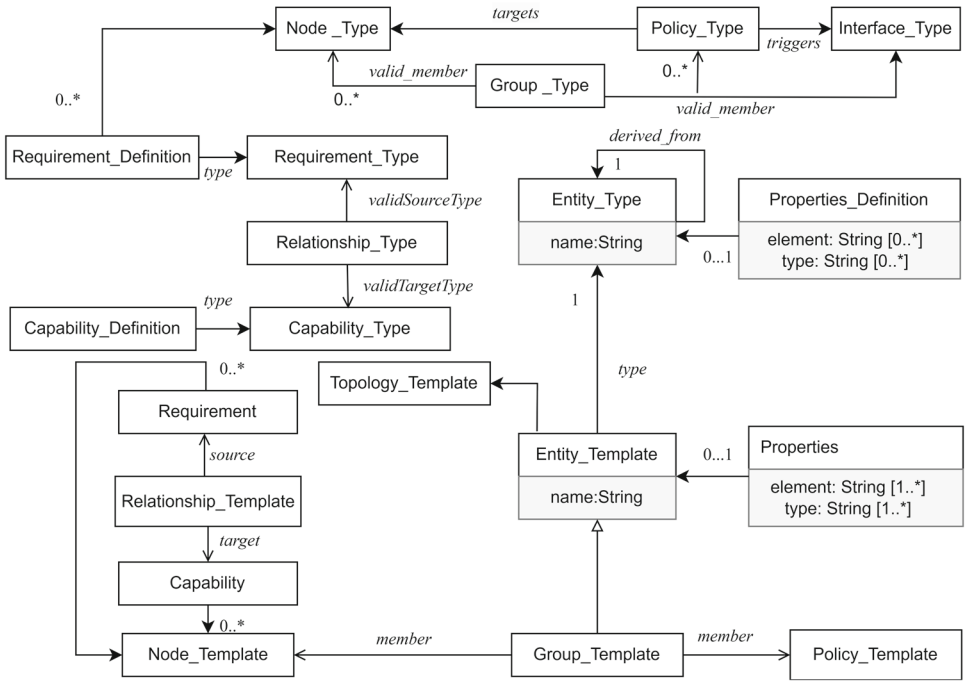


Fig. 2 An excerpt from the TOSCA meta-model, partially based on Challita et al. [13]

Fig. 3 An excerpt of a TOSCA blueprint: node types, node templates, and policy types

```
node_templates:
  snow-mysql:
    type: DockerizedComponent
    properties:
      image_name: "snow-mysql:v2"
    requirements:
        host:
          node: snow-docker-host

  snow-docker-host:
    type: DockerHost
    requirements:
      host:
        node: snow-vm

 snow_vm:
    type: OpenStackVM
    properties:
       name: "HostVM"
       image: "centos7"
       flavor: 'm1.xsmall'
       network: 'provider_64_net'
       key_name: 'my_key'
    requirements:
          host:
             node: snow-docker-host

  autoscale:
    type: AutoScale
    properties:
       min_size: 3
       max_size: 7
    targets: [ snow_vm]
```

```
node_types:
  DockerizedComponent:
    derived_from: tosca.nodes.SoftwareComponent
    requirements:
      host:
        capability: tosca.capabilities.Compute
          node: DockerHost
            relationship: tosca.relationships.HostedOn

 DockerHost:
    derived_from: tosca.nodes.SoftwareComponent
    capabilities:
      host:
        type: tosca.capabilities.Compute

 OpenStackVM:
       derived_from: tosca.nodes.SoftwareComponent
       properties:
          name:
             type: string
          image:
             type: string
          flavor:
             type: string
          network:
             type: string
          key_name:
             type: string


  policy_types:
    AutoScale:
      derived_from: tosca.policies.Scaling
      properties:
         min_size:
            type: integer
            description: 'The minimum no. of instances'
            required: true
         max_size:
            type: integer
            description: 'The maximum no. of instances'
            required: true
```

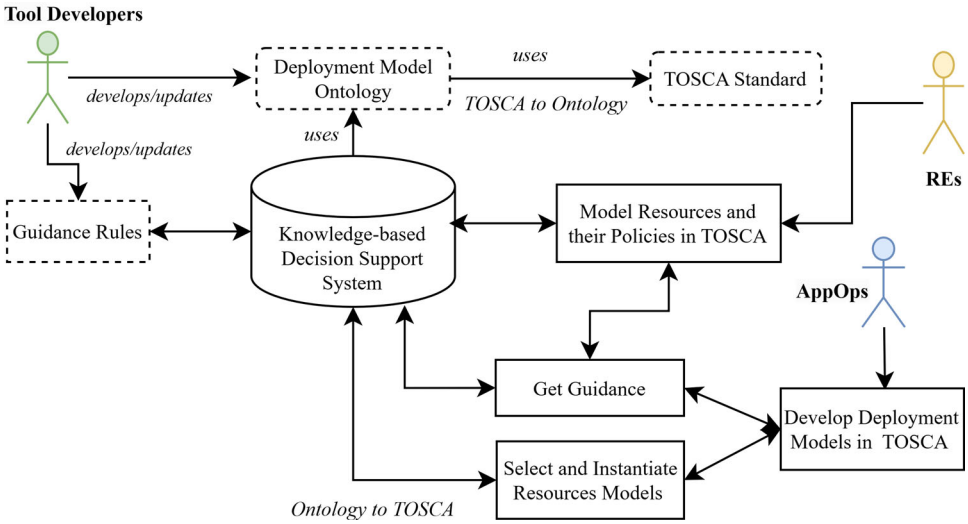


Fig. 4 An overview of our approach

IaC languages, our approach can also potentially be applied to other IaC languages.

4 Guided development of deployment blueprints in TOSCA

This section presents our knowledge-based approach to model TOSCA-based IaC blueprints to deploy complex applications. We developed and validated our approach within the SODALITE Horizon Europe project. Section 4.1 provides an overview of our approach, highlighting the support for the two requirements identified in the previous section. Section 4.2 presents the SODALITE ontology and its support for modeling TOSCA-based deployment blueprints. In Sect. 4.3, we explain how ontological reasoning can be used to implement various types of guidance to simplify the modeling of complex deployment blueprints. Finally, Sect. 4.4 discusses the workflows for different actors in the IaC development environment.

4.1 Approach overview

Figure 4 summarizes our approach, depicting key components, actors, and their responsibilities. The Knowledge-Based Decision Support System (*KB-DSS*) stores the models of resources, platform components, and their instances as interconnected knowledge graphs (KGs) and supports codifying guidance as ontology reasoning logic. In the SODALITE project, we developed an ontology that can represent TOSCA-based deployment models. The developers of the KB-DSS tool (e.g., the SODALITE team) are responsible for creating and managing the ontology and reasoning rules.

Resource Experts (REs) create TOSCA-based models for resources and their access and lifecycle management policies (**Req2**). The typical resources are infrastructure resources (e.g., VMs and HPC resources) and platform/application components (e.g., Docker engine, Dockerized components, and databases). TOSCA-based resource models are automatically converted into knowledge graphs and stored in the *KB-DSS*.

Application Operational Experts (AppOps) are responsible for creating the deployment blueprints for their applications. They can connect to the *KB-DSS*, interactively search and select the models of the resources that their applications use, and instantiate those models to create deployment model instances (i.e., TOSCA topology templates). To reduce the complexity of the modeling efforts of REs and AppOps, *KB-DSS* provides interactive assistance such as auto-completing and validating models (**Req1**). We provide the implementation of the four common types of modeling guidance (covering three industrial use cases of the SODALITE project) and support the creation of custom guidance as ontology reasoning rules. The prospective developers of our tool can update the ontology and the reasoning rules and add new rules to support new modeling assistance types.

It is important to note that REs and AppOps model the deployment models using TOSCA. We provide an IDE that uses a textual and graphical DSL (Domain-Specific Language). The meta-model of the DSL is based on the TOSCA meta-model. The IDE uses the guidance services of the *KB-DSS* to assist the modelers during the modeling process. The TOSCA-based models are automatically and transparently converted to ontology-based models and vice versa. As the primary research contribution of this paper is the ontology-based guided development methodology, we do not discuss the implementation of our IDE in this paper. We refer interested readers to the SODALITE handbook [18]. The rest of

this section discusses our IaC development methodology in detail.

4.2 Capturing deployment modeling knowledge through ontologies

We use ontologies to represent deployment models and the relevant domain knowledge. Ontologies enable interoperability and reuse of knowledge and building guidance services utilizing ontology reasoning techniques [61]. We follow the good practices in ontology engineering to describe deployment-related concepts in an extensible, standardized model. In particular, we use Ontology Design Patterns (ODPs) [7] and exploit the meta-modeling capabilities of the OWL2 Web Ontology Language [40].

4.2.1 SODALITE deployment model ontology and its mapping to TOSCA

Our ontology extends an existing general-purpose ontology, the DOLCE [24], and in particular, the DUL ODP [8] [25], which is part of the DOLCE+DnS Ultralite ontology, a light version of the DOLCE ontology. The ODPs represent modular and reusable ontological components that can be easily leveraged to build complex ontologies. Their usage can also improve comprehension, reasoning, and maintenance of ontologies [7]. We used the DnS pattern because it enables modeling high-level descriptions and situations (domain independent) that can be used in various domains [25].

Figure 5 shows the SODALITE ODP that extends the DnS design pattern. A `SodaliteSituation` can model the entities in an application deployment context or situation. Each `SodaliteSituation` has a descriptive context (`:hasContext`) modeled with `SodaliteDescription`, which specifies (`:specification`) one or more concepts (`SodaliteConcept`) to be used for classifying and interpreting entities within the situation. Each `SodaliteConcept` can have one or more parameters (`SodaliteParameter`). Each `SodaliteConcept` or `SodaliteParameter` might have a property assertion of a literal value (`:hasDataValue`) or an object value (`:hasObjectValue`).

Our ontology is based on the TOSCA standard to foster interoperable modeling and description of IaC programs. Table 1 shows the conceptual mappings between the TOSCA and SODALITE ODP concepts. Figure 6 shows an example of such a mapping corresponding to *snow-vm* and *autoscale* templates in Fig. 3. Node types and templates, relationship types and templates, and policy types and templates (i.e., resources, their relationships, and their policies) are situations (subclass of `SodaliteSituation`). Each situation uses a `SodaliteDescription` to define the concepts for interpreting domain entities in a specific context. Some examples of such concepts are properties, attributes, capabilities, requirements, and policy triggers (modeled as subclasses of `SodaliteConcept`). Each concept may use one or more parameters (modeled as `SodaliteParameter`) to enrich it with additional descriptive context. Consider the *snow-vm* node template in Fig. 3 and its ontological representation in Fig. 6. It is modeled as a `SodaliteSituation` containing a `SodaliteDescription` (*:desc*). The description contains (`:hasContext`) a property and a requirement. Its property `:prop1` is a `SodaliteConcept` that classifies the domain entity *:image* and has a data value *centos*. Its requirement `:req1` (another `SodaliteConcept`) classifies the entity *:host* and includes a `SodaliteParameter` *:param1* that has as an object value (`:hasObjectValue`) of *snow-docker-host*.

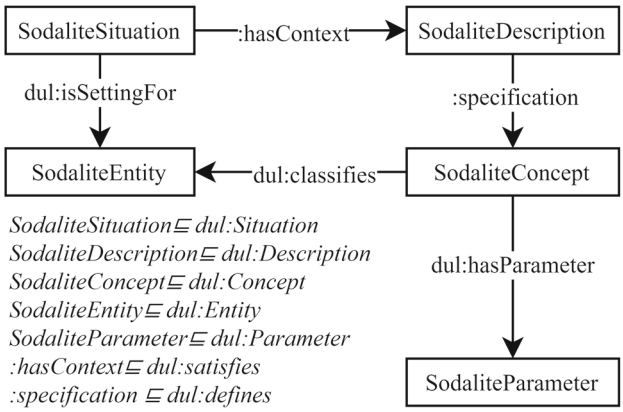


Fig. 5 SODALITE ODP (Ontology Design Pattern)

4.2.2 Layered representation of the SODALITE ontology

We implemented our ontology by utilizing the expressiveness and conceptual modeling capabilities of OWL2, especially meta-modeling (or punning) [40]. OWL2 meta-modeling enables conceiving the complexity of TOSCA by introducing the different modeling layers and the multiple roles per entity. Moreover, it allows the same identifier to be treated as a class and an instance. Thereby, re-usability is promoted as subsumption hierarchies can be captured. For example, a TOSCA node type can be a class and an instance in the ontology. Descriptive contexts, such as node properties and capabilities, can be attached to a node type by treating it as an instance. We have captured the knowledge into four levels through the SODALITE ODP to reduce modeling complexity and separate the modeling responsibilities of AppOps and REs. Figure 7 shows a fragment of the ontology that rep-

[8] http://ontologydesignpatterns.org/wiki/Ontology:DOLCE%2BDnS_Ultralite.

Table 1 Mapping TOSCA concepts to SODALITE ODP concepts

TOSCA concept	Ontology concept
Node Type, Node Template, Relationship Type, Relationship Template, Policy Type, and Policy Template	SodaliteSituation andSodaliteDescription
Policy Target, Property, Attribute, Capability, Requirement, and Trigger	SodaliteConcept andSodaliteParameter

Fig. 6 Ontology OWL elements for TOSCA elements in Fig. 3: **a** snow-vm node template, **b** autoscale policy template, **c** OpenStackVM node type, and **d** AutoScale policy type

resents the deployment model for our motivation use case (presented in Fig. 1)

Tier 0 represents the SODALITE ODP and the TOSCA meta-model. All tiers follow the SODALITE ODP. The meta-model is the static schema of the ontology and consists of the TOSCA vocabulary, mainly various TOSCA normative types such as node, capability, and relationship. In our example, the TOSCA normative resource types `Node`, `SoftwareComponent`, `Compute`, `Capabilities.Compute`, and `Capabilities.Container` are depicted as subclasses of the `SodaliteSituation`. We included the descriptions of the `Compute` and the `SoftwareComponent` nodes through the `hasContext` property assertion, `SoftwareComponent-desc`, and `Compute-desc`, respectively. For example, the `SoftwareComponent-desc` instantiates the `SodaliteDescription` of the ODP layer and has a requirement to be hosted on a `Compute` node. The `node` requirement is an instance of the `SodaliteConcept`.

Tier 1 includes all custom reusable types that the REs can define, for example, OpenStack virtual machines, MySQL databases, Docker engines, and HPC nodes. In our example, `DockerizedComponent`, `DockerHost`, `DockerRegistry`, `OpenstackVM`, `HPCMWTorque`, and `HPCJobTorque` are resource types. A description is included as a context for some resource types through the object property assertion `hasContext`. For example, `HPCWMTorque` uses `HPCWMTorque-desc` to capture its attributes, in this specific case, `username`, an instance of SodaliteConcept. The `username` concept has the `type` parameter, which contains the object value as a string. Torque is a resource manager that can schedule HPC batch jobs over a cluster of distributed compute nodes .[9] The username is the Torque admin user.

Tier 2 represents the deployment model of an application consisting of software and hardware components defined by AppOps. Those components are instances of Tier 0 and Tier 1 types. For example, a Docker engine instance (a specific installation of the Docker engine) and OpenStack VM instances can be modeled as node templates by instantiating the corresponding node types. In our example, the main application instances are the `snow-skyline-extractor` that contains the deep learning model that extracts the skyline of images, and the `snow-skyline-training-job` that trains the model. The rest of the application templates are the dependencies of the two aforementioned application instances, for example, as hosts and registries, which are `docker-host`, `snow-docker-registry`, `snow-vm`, `hpc-wm-torque`, and `modak-job`. Those components are instances of resource types of Tier 1, which are subclasses of `SodaliteSituation`. Each template has its description through the `hasContext` object property assertion. For example, the `hpc-wm-torque` has as a description the `hpc-wm-torque-desc`, which is an instance of the `SodaliteDescription`. This description instantiates the concepts (e.g., the attribute `username`)

[9] https://hpc-wiki.info/hpc/Torque.

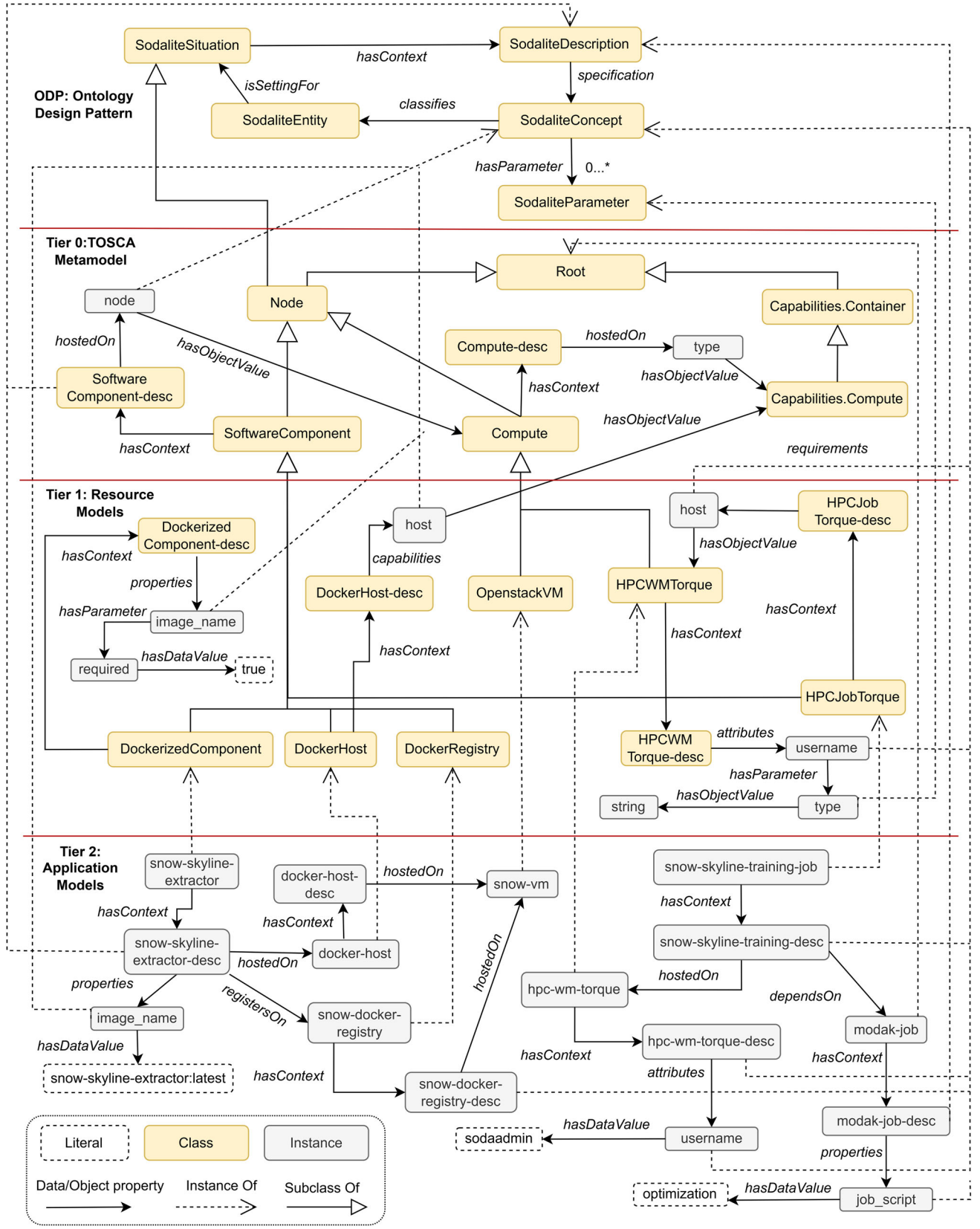


Fig. 7 An excerpt of the ontology from the snow use case

of the corresponding resource type of Tier 1, namely the `HPCWMTorque`.

4.2.3 Ontology-to-TOSCA bidirectional transformation

We utilized the conceptual mappings between TOSCA and our ontology to implement a bidirectional transformation between a TOSCA blueprint and an ontology instance. This bidirectional transformation enhances the framework with additional guidance capabilities. Specifically, REs can import existing TOSCA models into the KB as knowledge graphs to receive guidance on updating them or offering additional resource models that AppOps can utilize when creating application deployment models. The reverse transformation, from OWL to TOSCA, retrieves all the models saved in KB and enables the seamless deployment of application models, which require the models in TOSCA format.

Algorithm 1 shows the pseudo-code for converting a TOSCA blueprint into an ontology representation. The input is a model in TOSCA, and the output is a model represented in OWL language. The following description explains how a TOSCA model can be recursively parsed and transformed into OWL. The function `CovertTOSCA2Ontology` loads the TOSCA model that is written in YAML language to a Map object (`ModelMap`), consisting of key-value pairs, using the `loadYamlToMapObjects` function. Subsequently, the maps of the types are assigned to `typeMaps` through the `getTypeMaps` function. Lines 4–8 of the algorithm iterate over the description of each type, accomplished via key-value pairs. The `ConvertConceptType2OWL` function converts a key-value pair to the corresponding OWL representation. A key denotes the concept name (e.g., a requirement or a property), while the associated value is the nested description.

In line 13, it is conditionally checked if the value is a Map, indicating the presence of a more nested description. In such a case, the `ConvertConceptType2OWL` is called recursively (line 16). The algorithm's output (in OWL) is incrementally built through the `buildOwl` function (line 14), which gets the key and builds the OWL as a string. Through predefined keywords, for example, it maps the standard TOSCA vocabulary with the corresponding OWL representation, such as *requirements host* to *:requirements:Requirement1.:Requirement_1:classifies:host*. As such, the OWL model has been built, and the recursion ends when the value of a key is a Literal (line 17). Finally, in line 9, the OWL model is returned.

Algorithm 2 presents the pseudo-code for transforming the ontology to TOSCA. The input of this algorithm is the identifier of the model (`model_id`) that is already saved in the *KB-DSS* (see Figure 4), and the output is the corresponding TOSCA representation of the model. The `ConvertOntology2TOSCA` function initially queries the

Algorithm 1 Conversion from TOSCA to Ontology

```
   Input: Model m
   Output: owl - model saved in owl in KB
1: function CovertTOSCA2Ontology(m)
2:     ModelMap ← loadYamlToMapsObjects()
3:     typeMaps ← getTypeMaps(ModelMap)
4:     for each t ∈ typeMaps do
5:         for each (key, value) ∈ n do
6:             owl.append(convertConceptTypeElement2OWL
7:             (key, value))
8:         end for
9:     end for
10:    return owl
11: end function
12:
13: function ConvertConceptType2OWL(key, value)        ▷ convert each TOSCA concept (requirements, capabilities, properties, attributes)
14:     if value is Map then
15:         owl.append(buildOwl(key))
16:         (key, value) = getKeyValuePair(value)
17:         ConvertConceptType2OWL(key, value)
18:     else if value is Literal then
19:         owl.append(buildOwl(key))
20:         return owl
21:     end if
22: end function
```

KB (`QueryKBForModel`) through a SPARQL query. This query retrieves the names of the types or templates associated with the resource or application model accordingly. Following this, line 3 involves the assignment of template or type names to the `elements` object through the `getElements` function. Subsequently, `elementObjects` is initialized, and this object will later be appended with TOSCA elements constructed in subsequent lines. Lines 5–8 iterate over the `elements` and query the KB to retrieve detailed descriptions of the elements, which are the underlying concepts. Line 6 calls the `QueryKBForConcepts` to convert the description of each element to its corresponding TOSCA representation. The KB is queried through SPARQL for all kinds of concepts, lines 14–21, and the results are assigned to Java objects (TOSCA object models) in line 23. Finally, in line 9, the Java objects (`elementObjects`) are parsed for mapping them to the corresponding TOSCA elements.

4.3 Capturing guidance through ontology reasoning

We use the semantic models in the knowledge base (KB) to provide interactive modeling assistance to users. In particular, within the scope of the SODALITE project, we developed four guidance types using ontology reasoning: (i) matchmaking, (ii) validation, (iii) smell detection, and (iv) auto-completion. We use SPARQL, the standard ontology query language, to implement them. The SPARQL queries are agnostic to the use cases. The developers of our tool are responsible for creating and modifying these rules.

Algorithm 2 Conversion from Ontology to TOSCA

Input: model_id is the identifier of the model in string format
Output: TOSCA - TOSCA in YAML

function CONVERTONTOLOGY2TOSCA($model\_id$)
  $results \leftarrow QueryKBForModel(model_id)$ ▷ query KB for resource or application model element names (type or template names correspondingly)
  $elements \leftarrow getElements(results)$ ▷ elements can be templates (for application models) or types (for resource models)
  $elementObjects \leftarrow []$
  for each $e \in elements$ **do**
    $eo \leftarrow QueryKBForConcepts(e)$
    $elementObjects.append(eo)$
  end for
  $tosca \leftarrow convertElementObjectsToTOSCA(elementObjects)$ ▷ parse the java objects
  return TOSCA
end function

function QUERYKBFORCONCEPTS(e) ▷ convert each tosca concept (requirements, capabilities, properties, attributes)
  $attr \leftarrow queryKBForAttributes(e)$
  $art \leftarrow queryKBForArtifacts(e))$
  $cap \leftarrow queryKBForCapabilities(e)$
  $inter \leftarrow queryKBForInterfaces(e)$
  $prop \leftarrow queryKBForProperties(e)$
  $req \leftarrow queryKBForRequirements(e)$
  $tar \leftarrow queryKBForTargets(e)$
  $tr \leftarrow queryKBForTriggers(e)$
  $concepts \leftarrow attr, art, cap, inter, prop, req, tar, tr$ ▷ Load concepts to java objects
  $elementObject \leftarrow loadConceptsToObjects(concepts)$
  return elementObject
end function

4.3.1 Matchmaking

Semantic annotation of infrastructure resources and application components (i.e., nodes in a TOSCA-based deployment topology) allows their automated discovery and comparison. We implemented three types of node matchmaking capabilities relevant to TOSCA:

- *Matchmaking based on TOSCA node properties.* Logical expressions on the constraints on node properties can be used to select nodes. These constraints ensure that the nodes satisfy the requirements of specific applications. For example, a VM with an *UbuntuOs* image and a *small* flavor can be discovered by checking the properties of the available compute nodes. The logical expressions can include a wide range of node properties such as resource capabilities, geographical locations of resources, and network capabilities.
- *Matchmaking based on TOSCA node capabilities and requirements.* The nodes can be discovered by matching the requirements of the source node with the capabilities provided by the candidate target nodes. For example, a compute node capable of hosting a MySQL database can be discovered. Moreover, the capabilities of the resources used in the application model can be retrieved, for example, the number of GPUs, the availability of SSDs, and the CPU architecture. This enables AppOps to select the optimal parameters for the application's resources and components (e.g., flags for the compiler that can make the AI training faster on a specific hardware resource).
- *Matchmaking based on TOSCA policies.* We support the policies that are either evaluated to be *true* or *false* based on the values provided by one or more properties of the policy. For example, a node that will be only deployed on the data centers in Germany or the Netherlands can be discovered by checking if the node has a placement policy that enforces the desired deployment constraint.

Listing 1 shows a SPARQL query that implements matchmaking using node properties. The query retrieves all nodes of type `tosca.nodes.Compute` (lines 4–5), whose potential role can be real or virtual machines or servers. Specifically, it narrows down the selection to nodes with the *flavor* property equal to *small* (lines 9–13). Our matchmaking component automatically generated this query based on the information in the node selection expression ?flavor = "m1.small".

Listing 1 Matchmaking SPARQL query for finding compute nodes with small flavor (auto-generated)

```
1  select DISTINCT ?node ?description
   ↪  ?nodeType
2  where {
3
4      ?node rdf:type ?nodeType .
5      ?nodeType rdfs:subClassOf
       ↪  tosca:tosca.nodes.Compute .
6
7      OPTIONAL {?node dcterms:description
       ↪  ?description .}
8
9      ?node
       ↪  soda:hasContext/tosca:properties
       ↪  ?concept .
10     ?concept DUL:classifies snow:flavor .
11     ?concept tosca:hasDataValue ?flavor .
12
13     FILTER ( ?flavor = "m1.small" )
14 }
```

4.3.2 Validation

The validation assistance feature enables users to detect and fix inconsistencies in their deployment models. We consider violations of the constraints defined by the TOSCA standard and those defined by REs as inconsistencies. REs can

define custom constraints for resource models as constraints on the properties of the model elements, such as optional or mandatory properties or value constraints (supported by TOSCA). We use SPARQL queries to implement the validation logic. Zimmermann et al. [73] also used a rule-based approach to ensure the satisfaction of security-related constraints in application deployment models. We developed the rules for validating the inter-node relationships in an application topology. In particular, we support the validation conditions identified by Brogi et al. [10]. Those conditions ensure the validity of sources (i.e., a requirement of a node), instances (i.e., a relationship template), and targets (i.e., a node or a capability of a node) of relationships in a TOSCA-based deployment model.

In addition to the validation of constraints, other types of errors can also be detected. For example, the properties and attributes of the node templates can take their values from the input parameters of the application model. Hence, the application model is deemed inconsistent if an input parameter assigned to a property or attribute is not declared in the model.

Policy violations are also detected by checking whether the application topology violates the constraints defined by a given policy. For example, if a policy has a constraint on the number of CPUs to be 4 at maximum, but a node template has 8 CPUs assigned as its capability, then the policy is violated.

Regarding the validation of the requirements [10], Fig. 8 shows a snippet of a SPARQL-based rule for checking a condition proposed by Brogi et al. [10]. The condition is *When a node is assigned to a requirement of a node template and does not include any capabilities, but instead only the target node template, it should be verified that the target node offers at least one type-compatible capability*. Lines 3–5 retrieve the type of the target node (`type_r_a_node`) for the network (`r_a`) requirement. Lines 10–13 retrieve the capability type (`capabilityType`) from the definition of the target node type. Lines 15–19 verify that the capability type of the target node is a subclass of the capability type (`r_d_capability`) defined in the type definition of the application template.

4.3.3 Smell detection

The IaC programs can have code smells, such as using anti-patterns and violating best coding patterns and practices [32, 49]. Code smells impact the quality and maintainability of the IaC code [55]. The smell detection guidance type supports the users in detecting smells in TOSCA-based deployment blueprints. We use SPARQL queries to codify the detection rules. We support detecting common types of security smells in TOSCA-based deployment models such as unrestricted IP address, empty passwords, invalid port ranges, and admin by default. We refer the interested readers to [34].

An example of a smell related to a security issue is the unrestricted IP address [49] in which a server is bound to 0.0.0.0; therefore, it is exposed through all network interfaces. Listing 2 shows a SPARQL query detecting the unrestricted IP smell. The query retrieves all node templates (line 3) that have property values (lines 4–8) equal to 0.0.0.0 (lines 9–11).

Listing 2 Smell detection SPARQL query detecting invalid ip address binding smell

```
1  select DISTINCT ?node ?description ?nodeType
2  where {
3
4       ?node rdf:type ?nodeType .
5       ?nodeType rdfs:subClassOf
        ↪  tosca:tosca.nodes.Compute .
6
7       OPTIONAL {?node dcterms:description
        ↪  ?description .}
8
9       ?node soda:hasContext/tosca:properties ?concept
        ↪  .
10      ?concept DUL:classifies snow:flavor .
11      ?concept tosca:hasDataValue ?flavor .
12
13      FILTER ( ?flavor = "m1.small" )
14 }
```

4.3.4 Autocompletion

In our modeling environment, we use a TOSCA meta-model aligned with the TOSCA Light [70], where users can use a minimal set of TOSCA constructs to create a deployment model. Users can omit information such as node requirements and relationships from their deployment models, and the *KB-DSS* reasoning capabilities can auto-complete missing content. For example, according to the TOSCA standard, a node requirement is by default required, either when *occurrences* field is omitted or defined explicitly with `min` greater than zero. Thus, the deployment model can be completed automatically even when a user does not include a mandatory requirement in a template. Listings 3 and 4 present an implementation of a model autocompletion use case. The SPARQL query in Listing 3 detects nodes that can fulfill the requirement of serving as a host while satisfying a specific VM placement policy. The SPARQL query in Listing 4 adds the missing required requirement to the `context` of the template by extending its requirements with the `?var_template` detected by executing the previous query.

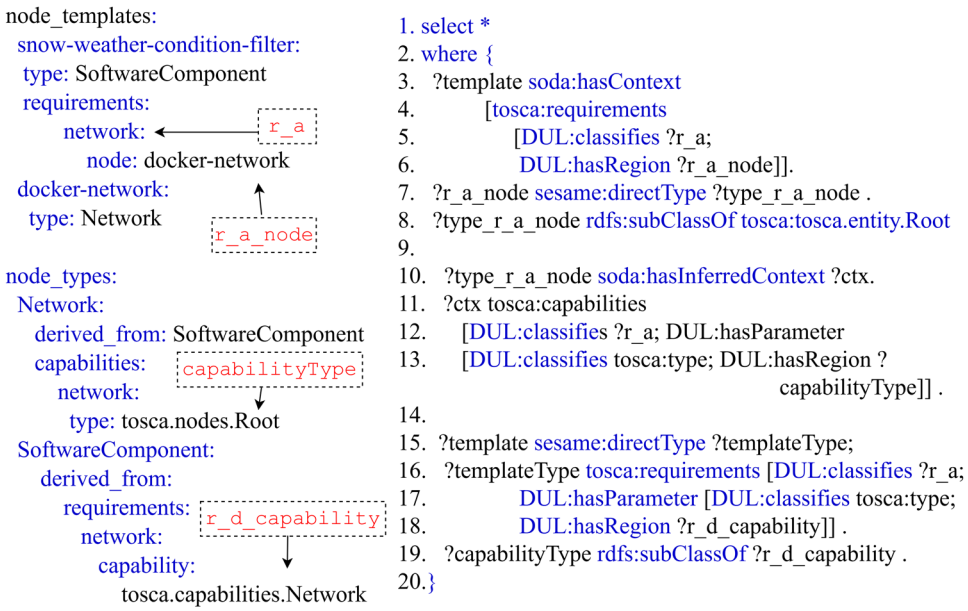


Fig. 8 Validation SPARQL query detecting inconsistencies in the requirements of a node template

Listing 3 Autocompletion SPARQL query for detecting templates being the target of a specific policy

```
select distinct ?template ?context {
    ?template a soda:SodaliteSituation .
    ?template rdf:type ?type .
    ?template soda:hasContext ?context
    ?policytemplate a
    ↪ soda:SodaliteSituation .
    ?policytemplate a
    ↪ tosca:tosca.policies.Placement .
    ?policytemplate soda:targets
    ↪ ?template
}
```

Listing 4 Autocompletion SPARQL query for completing the requirements

```
select distinct ?template ?context {
    ?template a soda:SodaliteSituation .
    ?template rdf:type ?type .
    ?template soda:hasContext ?context
    ?policytemplate a
    ↪ soda:SodaliteSituation .
    ?policytemplate a
    ↪ tosca:tosca.policies.Placement .
    ?policytemplate soda:targets
    ↪ ?template
}
```

4.4 Guided deployment modeling process

We envision an environment in which REs and AppOps collaboratively develop IaC scripts to deploy an application. In particular, REs with intimate knowledge of different types of physical and software computing resources can model resources. AppOps can use these resource models to create application deployment models by mapping application components to resources. Our approach interactively guides both actors when creating their corresponding models. Figure 9 shows the model development workflows for the two actors.

4.4.1 REs workflow

REs, as depicted in Fig. 9a, design the definitions of the resource types (i.e., TOSCA node types) that AppOps will use for the instantiation of the components of their applications. They define the semantics of the resource types, for example, the properties/attributes that can be assigned to the instances of the corresponding resource types. The first step in the workflow is to check if a resource model exists for the target resource type by querying the KB. If there are no models, REs create an empty resource model by providing a unique name. Next, they can complete the newly created model or update/edit an existing model. In either case, REs can request four types of modeling guidance we support. For example, the matchmaking guidance allows experts to discover concepts and resource types that can be reused in their resource models. With the smell detection guidance, they can find the code smells in their models and fix them to improve the quality of the resource types. Finally, REs can save their resource models in the *KB-DSS*. In this step, the models are

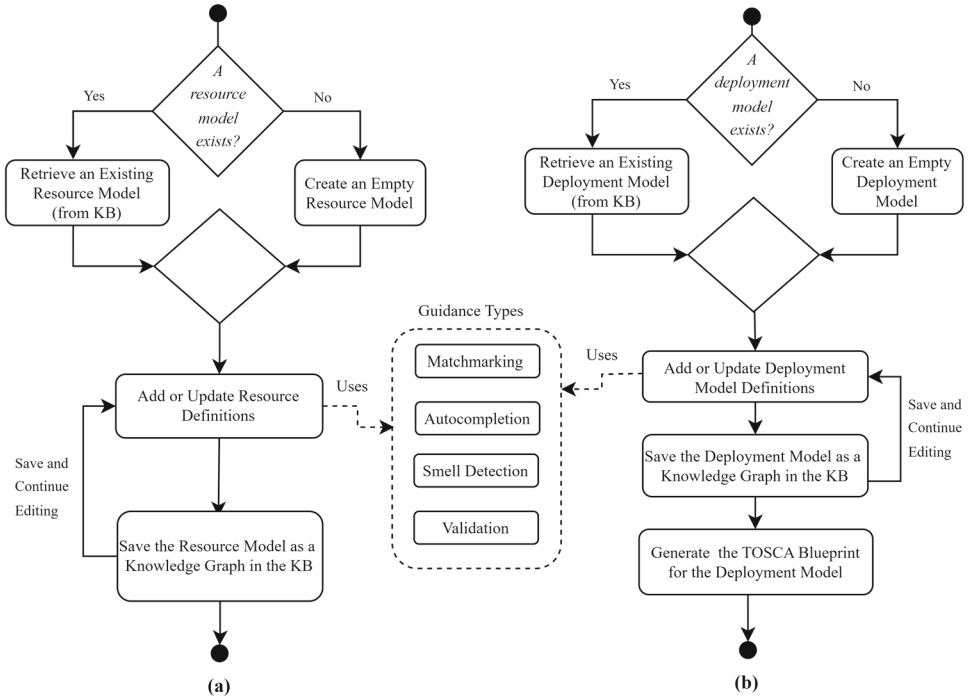


Fig. 9 Guided process for **a** REs and **b** AppOps

checked for completeness, and if there are missing parts (e.g., requirements and dependencies), the autocompletion guidance is automatically applied to produce valid and complete resource models.

4.4.2 AppOps workflow

AppOps, as depicted in Fig. 9b, design a TOSCA-based deployment model for a given application by defining its components and their interconnections in terms of node and relationship templates. They can retrieve existing deployment models from the KB or create an empty model. When updating or defining an existing model, they can interactively request guidance from the *KB-DSS*. All four types of guidance can be used. For example, using matchmaking guidance, AppOps can discover nodes with specific capabilities, find nodes that are subclasses of a specific resource type, or get the properties of a resource type. Using smell detection and validation capabilities, they can detect and fix smells and structural and semantic inconsistencies in their deployment models. Moreover, missing information in the models can be detected and auto-completed using the corresponding guidance feature. A TOSCA-based model created by AppOps may only partially comply with the TOSCA syntax. Hence, AppOps can use the ontology-to-TOSCA transformation capabilities of the *KB-DSS* to generate a valid TOSCA blueprint from the deployment model and use the generated blueprint to deploy the application.

It is important to note that REs and AppOps can reuse existing TOSCA models. They can use the TOSCA-to-ontology capabilities of the *KB-DSS* to import the TOSCA models (YAML files) as ontology models into the KB. Then, they can get guidance from the *KB-DSS* to extend, update, or reuse those TOSCA models.

4.4.3 Algorithm of guided modeling process

Algorithm 3 shows the pseudo-code of our guided modeling process. The input is a resource model (from a RE) or a deployment model/topology (from an AppOps expert). The function AUTHORINGMODEL iteratively requests guidance from *KB-DSS* (lines 3–7). Once the user is satisfied with the result, the model is saved in the KB by the function SAVEMODELTOKG. When creating the model, the user can select four types of guidance: matchmaking, validation, smell detection, and auto-completion.

The GETGUIDANCE procedure integrates the four kinds of guidance, and depending on the guidance type (GTYPE) requested, the corresponding results are returned. For the matchmaking guidance type, the user selects which concepts (SELECTCONCEPTTOMATCH) to be used for matchmaking, such as resources offering specific capabilities or properties and attributes of existing resources, saved in the KB. For the validation guidance type, the user can select the model (SELECTRULESFORVALIDATION) to be validated against a set of specific rules (line 16). If the model is invalid, the errors are returned to the user (line 21). If the guidance type is smell detection, the model is validated against smells, such as security-related issues and violations of good coding practices. The user can optionally address the suggestions

Algorithm 3 Knowledge-based Guidance

Require: Model A, User um
Ensure: The model is valid and the user is satisfied, then the model gets saved to the Knowledge Base (KG)
1: **function** AUTHORINGMODEL(um, A) ▷ The user um requests guidance while authoring a resource model or a deployment model
2:
3: **while** um requests guidance **do**
4: $gType \leftarrow selectGuidanceType()$ ▷ gType = <Matchmaking, Validation,
5: Smell Detection, Autocompletion>
6: $guideResponse \leftarrow getGuidance(A, gType)$
7: **end while**
saveModelToKG(A) ▷ When guidance ends, save it to Knowledge Base (KB)
8: **end function**
9:
10:
11: **function** GETGUIDANCE(A, $gType$)
12: **if** $gType$ is Matchmaking **then**
13: $C \leftarrow selectConceptToMatch()$
14: return matchmake(C)
15: **else if** $gType$ is Validation **then**
16: $R \leftarrow selectRulesForValidation()$
17: $errors \leftarrow []$
18: **for each** $r \in \mathcal{R}$ **do**
19: $errors.append(validateRule(A, r, R))$ ▷ Validate the model against each rule
20: **end for**
21: return errors
22: **else if** $gType$ is Smell Detection **then**
23: return smellsDetect(A)
24: **else if** $gType$ is Autocompletion **then**
25: $R \leftarrow selectRulesForAutocompletion()$
26: return autocompletion(A, R)
27: **end if**
28: **end function**

for fixing the smells. Finally, if the guidance type is autocompletion, the missing parts of the model get auto-filled.

5 Implementation

This section briefly presents the implementation of the two key components in the SODALITE modeling environment: Knowledge-Based Decision Support System (KB-DSS) and SODALITE IDE. The source code of the SODALITE project is available in our GitHub repository .[10] The demos of various SODALITE components, including IDE and KB-DSS, are available on our YouTube channel .[11] The SODALITE handbook [18] describes the implementation of each component in the SODALITE framework in detail.

SODALITE IDE We provide graphical and textual editors to author TOSCA-based deployment models. We implemented our IDE as a set of plug-ins for the Eclipse IDE using the XText [12] and Sirius [13] frameworks. IDE provides a DSL based on the TOSCA meta-model. IDE uses the APIs offered by the KB-DSS to provide the necessary guidance to model developers. Through the IDE, AppOps can also generate TOSCA blueprints from their models (in SODALITE DSL) and deploy the applications through the SODALITE orchestrator [33].

The screenshot of the IDE in Fig. 10 illustrates a scenario in which the user receives the matchmaking guidance about which nodes can secure an application component. Figure 11 shows a validation guide that detects mismatches between a required node and the assigned node. Figure 12 shows the ability to represent the same deployment model in textual and graphical views.

Knowledge-Based Decision Support System Figure 13 shows the components of our decision support system. The knowledge generated by REs and AppOps is accommodated in a Knowledge Base (KB), namely an RDF triple store. The KB complies with W3C standards, i.e., RDF, OWL 2, and SPARQL recommendations. It is hosted on the GraphDB [14] triple store. *Semantic Reasoner* [15] interacts with the KB through a REST API for importing and retrieving data. It comprises two sub-modules, *Semantic Population Engine* and the *Semantic Reasoning Engine*. *Semantic Population Engine* can translate the deployment models in TOSCA and SODALITE DSL to the SODALITE ontology using the mappings presented in Sect. 4.2. *Semantic Population Engine* contains the implementation of the transformation from TOSCA to ontology. *Semantic Reasoner* provides a REST API to execute various types of guidance. It contains the implementation of the transformation from ontology to TOSCA. As discussed in Sect. 4.3, the SPARQL queries are used to implement each guidance. *Semantic Reasoner* executes SPARQL queries against the KB and returns the results to the IDE/user.

Overall, our approach has three languages: OWL, SODALITE DSL, and TOSCA. Our DSL is the external language visible to users (AppOps and REs). TOSCA is the target IaC language. Sirius and XText were used to implement the DSL. IDE transforms DSL models to TOSCA and vice versa. This paper focuses on the bidirectional transformation between ontology and TOSCA, which enables saving DSL and TOSCA models in the KB as knowledge graphs.

This paper focuses on the bidirectional transformation between TOSCA and ontology-based representations utilizing OWL language to enable DSL and TOSCA models to be stored in the knowledge base (KB) as knowledge graphs.

[10] https://github.com/SODALITE-EU.

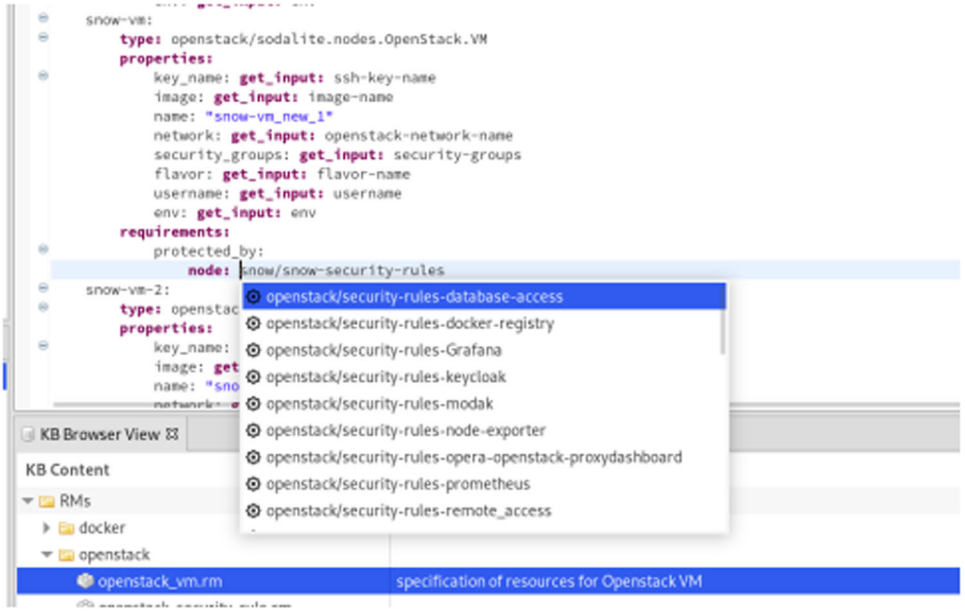
[11] https://www.youtube.com/@SodaliteHresearchproject/videos.

[12] https://www.eclipse.org/Xtext/.

[13] https://www.eclipse.org/sirius/.

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

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



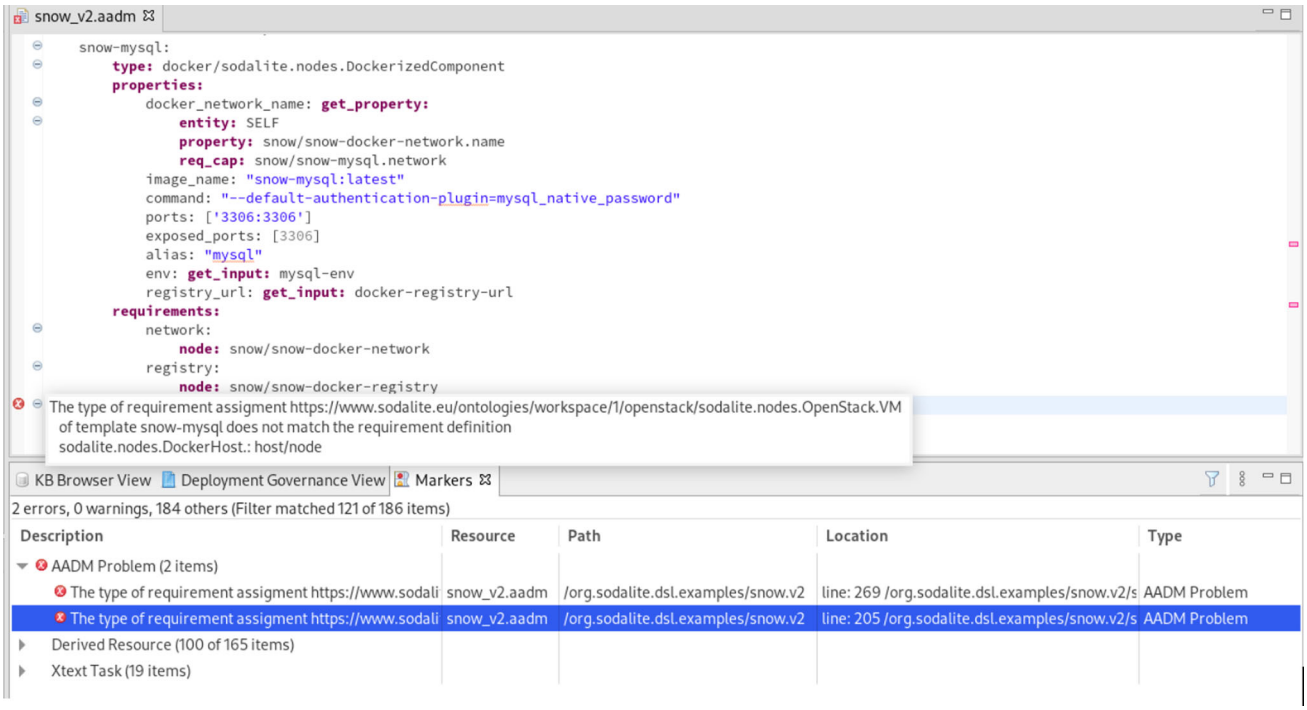


Fig. 10 An excerpt from the IDE showing the matchmaking assistance for resources that can securely protect the snow-vm virtual machine

Fig. 11 An excerpt from the IDE showing the validation assistance showing that a requirement assignment is of the wrong node type according to the DockerizedComponent resource type definition

KB models can also be transformed into DSL and TOSCA models, enabling seamless application deployment through the orchestrator. SODALITE DSL aims to provide users with a lightweight version of TOSCA, eliminating the details that can be automatically filled using our auto-completion feature.

6 Evaluation

To evaluate the proposed approach, we applied our framework to three industrial use cases of the SODALITE Horizon Europe project. We developed the resource and deployment models for each use case using the SODALITE IDE, following our guided development approach. We also performed control experiments with multiple user groups to

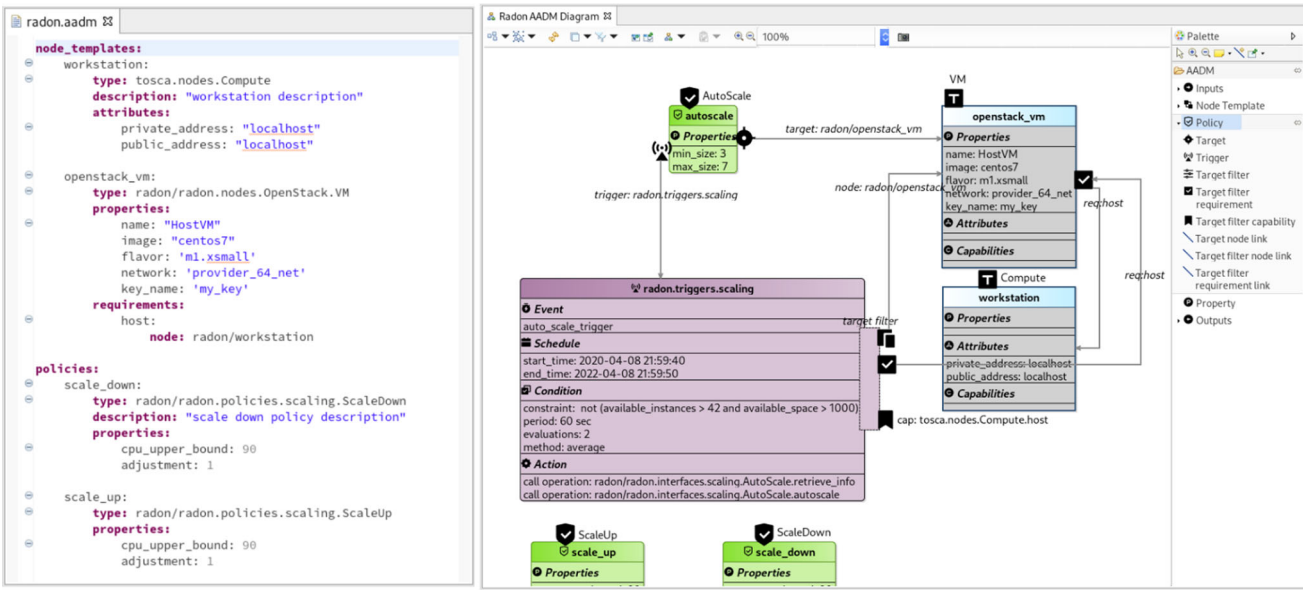


Fig. 12 An excerpt from the IDE showing text and graphical representation of a deployment model

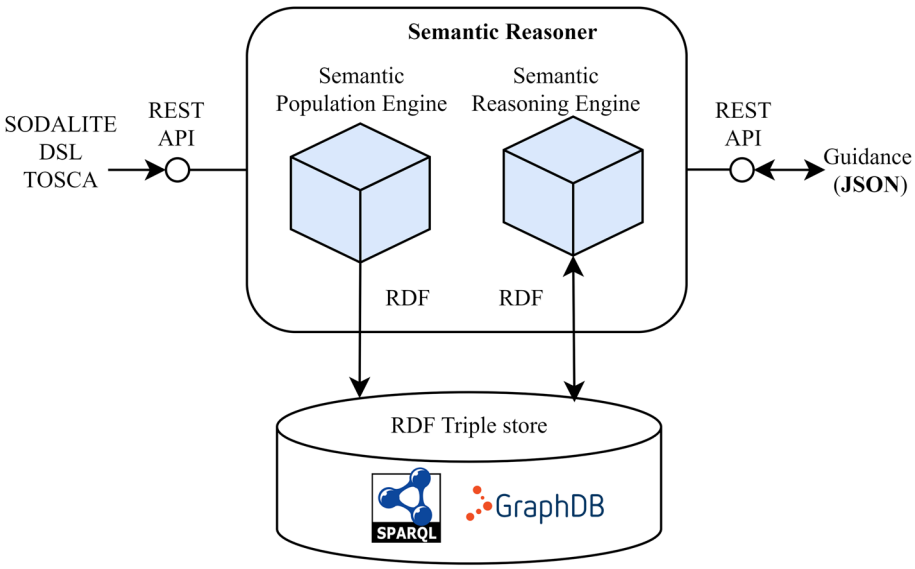


Fig. 13 Knowledge-Based Decision Support System internal architecture

assess the perceived ease of use, usefulness, and intention to adopt our approach and the SODALITE IDE. Additionally, we conducted a workshop at European Grid Infrastructure (EGI) [29], which included a practical hands-on tutorial on the SODALITE framework.

6.1 SODALITE project use cases

Three use cases used to validate our solution: snow index prediction, clinical trial simulation, and vehicle IoT. This section briefly discusses the implementation of each use case, highlighting their complexity, the use of heterogeneous compute resources, and the application of the guidance types.

6.1.1 Snow use case

Polytechnic University of Milan (POLIMI) developed the Snow case ,[16] which uses publicly available images of snow in the mountains to predict water availability. As shown in Fig. 14, it implements a complex data processing pipeline using cloud and HPC resources. The pipeline crawls geo-located images from heterogeneous sources, checks for the presence of mountains in each photograph, extracts a snow mask from the portion of the image denoting mountains, and calculates a water availability indicator, namely the snow index. The resource types modeled in the Snow use case

[16] https://www.sodalite.eu/snow-water.

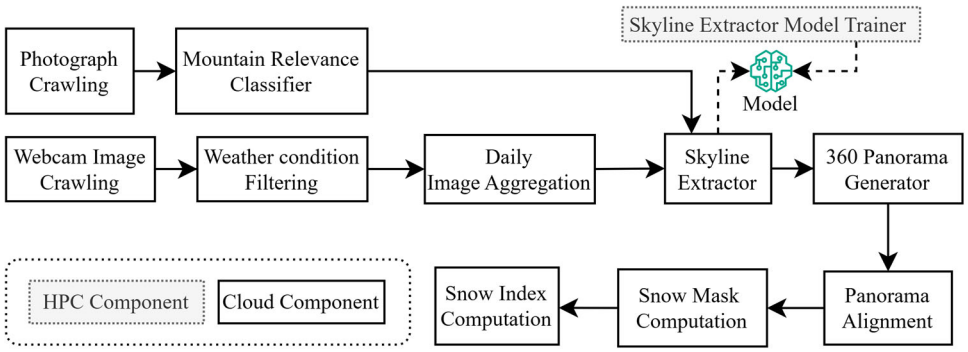


Fig. 14 Pipeline of the snow use case

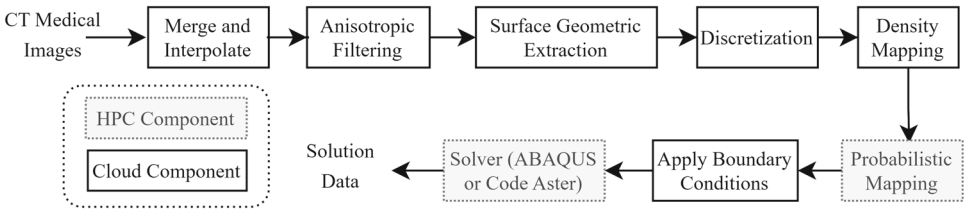


Fig. 15 Pipeline of the clinical trials use case

include dockerized components, docker engines, security rules, VMs, docker volumes, a docker registry, a docker network, (HPC) Torque jobs, (HPC) Torque workload manager, and an autoscale policy for VMs. We used a private OpenStack cloud and a private HPC cluster.

6.1.2 Clinical trials use case

The HPC Center at the University of Stuttgart (HLRS) developed this use case .[17] The clinical trials use case is a simulation process chain supporting in-silico clinical trials of bone-implant systems in Neurosurgery, Orthopedics, and Osteosynthesis. As shown in Fig. 15, the simulation chain is a data processing pipeline with multiple steps executed using HPC and cloud resources. The pipeline first retrieves clinical imaging datasets containing full-body X-ray and magnetic resonance imaging (MRI) data and extracts vertebral bodies. Next, the discretization component generates a volume mesh within the surface geometry, and the density mapping component maps the original image data to the volume mesh by assigning a density value to each element. Then, the probabilistic mapping component transforms the density values in the enhanced meshed geometry into clinical elasticity values. Finally, the solver component computes a solution that describes the structural mechanics of vertebral bodies. The resource types modeled in this use case include Dockerized images, Docker engines, virtual machines, Docker volumes, a Docker registry, a Docker network, (HPC) Torque workload managers, GridFTP data movers and clients, security rules, and (HPC) Torque Jobs.

6.1.3 Vehicle IoT use case

This use case [18] was developed by Adapadant. It focuses on a data management platform for connected vehicles. It implements in-vehicle data processing and intelligence at the network edge to address driver preferences and the need for compliance with GDPR (General Data Protection Regulation). Intelligence capabilities include intrusion and theft detection, drowsiness detection, and license plate detection. They are implemented as machine learning model inference services. As shown in Fig. 16, this use case adopts a microservices architecture, where the inference services are deployed in the edge, and the corresponding model training pipelines are deployed on the cloud. We used a cluster of three edge devices (Raspberry Pi 4, Google Coral AI Dev Board, and NVIDIA Jetson Xavier NX) managed by Kubernetes. The resource types modeled in the Vehicle use case include Dockerized components, Docker engines, VMs, security rules, Docker volumes, a Docker registry, a Docker network, Kubernetes clusters (edge and cloud), and Helm Kubernetes application manager. In [33], we presented SODALITE's support for orchestrating applications on edge-cloud environments in detail.

6.1.4 Coverage of guidance types in use cases

All use cases have been evaluated based on their use of the different types of guidance. Table 2 shows the results. The white rows refer to AppOps, and the gray rows refer to REs. Both AppOps and REs used the matchmaking feature to find the resource models of the compute nodes and software com-

[17] https://www.sodalite.eu/silico-clinical-trials-spinal-operations.

[18] https://www.sodalite.eu/communication-material/use-case-vehice-iot.

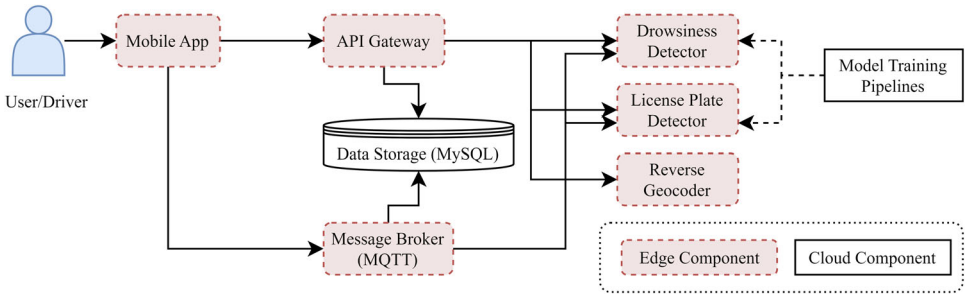


Fig. 16 Vehicle IoT use case architecture

ponents with specific properties and capabilities, such as a Kubernetes node with a GPU or a GridFTP file transfer client. However, AppOps aimed to create resource instances in their application deployment models, and REs focused on updating the properties of resource models and developing new resource models based on existing ones. AppOps used the validation feature to check the constraints on the properties and dependencies (capability and requirement mismatches) of the resource instances in their deployment models. REs used it to check the correctness of their resource models against TOSCA semantics and add validation constraints to the models. Both AppOps and REs employed the smell detection capability to check the presence of various code smells in their models. Finally, AppOps used the auto-completion feature to fill in missing requirements of the node templates (resource instances), and REs used it to add missing relationships among node types.

6.2 Validation with users

We performed controlled experiments with three types of users to assess their perception of the SODALITE modeling framework: non-expert users, TOSCA experts, and SODALITE use case owners. The empirical study measured three widely used perception metrics: perceived ease of use, perceived usefulness, and intention to use [16]. The SODALITE project ran for 36 months. We conducted experiments in the 24th month (M24) and the 37th month (M37). All experiments were conducted remotely. All experiment materials and data are available as a replication package .[19]

6.2.1 Participants

The participants for the M24 experiment included 9 non-experts, 5 TOSCA experts, and 4 use case owners (2 for the Snow use case and 1 per each of the other use cases). Due to the difficulty in recruiting external participants, we opted for *convenience sampling* [4]. The non-experts are students from the MSC in Data Science in Business and Entrepreneurship at the Jheronimus Academy of Data Science (JADS). The participation was completely voluntary. We invited all the students (65), and only 9 accepted. All participants had prior experience in deploying ML and big data applications using Docker, Docker Compose, Kubernetes, and Google Cloud (being taught in the master course). The TOSCA experts are selected from the employees of the industrial partners in the SODALITE project. Their proficiency in TOSCA is 3.80 on average (from 5). Regarding use case owners, none of them had experience in TOSCA. However, the level of their expertise in deploying applications is 3.25 on average (from 5).

The M37 experiment had 11 participants. We recruited them using the internal networks of the partners of the SODALITE project. None of them participated in the SODALTE project. Among the participants, 8 were advanced developers, 1 was an intermediate developer, and 2 were limited developers (able to write small functions). Regarding IaC proficiency, 6 participants had limited knowledge and could reuse existing code, 1 was an advanced IaC developer, and the remaining three were complete novices in the IaC area.

6.2.2 Task description

The main goal of the experiments is to assess the usefulness of the guidance features offered by the SODALITE IDE in different modeling scenarios. The tasks were designed to match the characteristics of various user groups. The experiment used a distributed ML application from the JADS data engineering course. The ML application was selected because non-experts had experience deploying it using Docker, Docker Compose, and Kubernetes. Moreover, the application deployment model needs all key TOSCA constructs supported by the SODALITE IDE. Figure 17 shows the application deployment model. All three services (prediction service, training service, and training db service) are Dockerized components (i.e., Docker containers). They are hosted and managed by a Docker Engine hosted in an OpenStack VM. A security configuration protects the VM. The Docker containers are connected via the two Docker networks. The Docker images are hosted on Docker Hub. These images are pulled and executed to create container instances. Docker volumes store training data and machine learning models.

[19] https://doi.org/10.5281/zenodo.13847546.

Table 2 Evaluation cases per guidance type of the three use cases

UC	Matchmaking	Validation	Smell detection	Autocompletion
	Find compute resources with $num\_gpus = 1$ and $disk\_size$ =1TB and $mem\_size = 50GB$ for hosting the mountain relevance classifier component	Ensure the webcam image crawler (node template) is not connected to an invalid network (not within the valid values set defined in the node type). Ensure the skyline extractor is a a software component resource and is hosted on a VM resource offering specific capabilities. Ensure the webcam image crawler is connected to a MySQL database	All smell types in our taxonomy were tested. An example: A HardCodedSecret smell returned for the username and password of the MySQL database where the images are saved	A missing requirement about a dependence of the weather condition filtering on a database containing the images is auto-completed
Snow	Find existing VM resource models from which an OpenStack VM resource model can be created	Ensure the communication protocol for OpenStack nodes is one of TCP, UDP, and ICMP (set constraints on the property protocol of the SecurityRule resource model)	An InvalidPortRange smell returned for a node type as the port was outside 0–65535 range	A missing relationship is autocompleted so that the OpenStack VM to be secured by a security policy
	Find resource models for software components that support file transfers using GridFTP	Ensure that the boundary condition checking job runs on a batch container runtime	Unrestricted IP address is used in the security rules	Auto-complete the requirements of the boundary condition job that should be hosted on an AWS VM with an autoscaling policy
Clinical	Retrieve all the resource models that have network capabilities (to update their properties)	Ensure batch container nodes always use the singularity as the container engine (adding a constraint on the runtime property of the resource model)	Inconsistent naming convention smell for the kube_config property of a Kubernetes resource model as it uses the snake case style	None
	Find the resource models of VMs that offer a specific number of GPUs (to create node instances)	Ensure a MySQL database containing license plate data is not hosted on a node placed in Germany	An empty password is detected in the MySQL database node template	Autocomplete the host requirements of edge nodes that are governed by anti-affinity policies
Vehicle	Find the resource models of Kubernetes nodes that has GPUs suitable for AI training (to add a node label)	Ensure the Kubernetes node selected includes a GPU (add a constraint on the property gpus)	A Hard-Coded secret was detected in the MySQL database node type	None

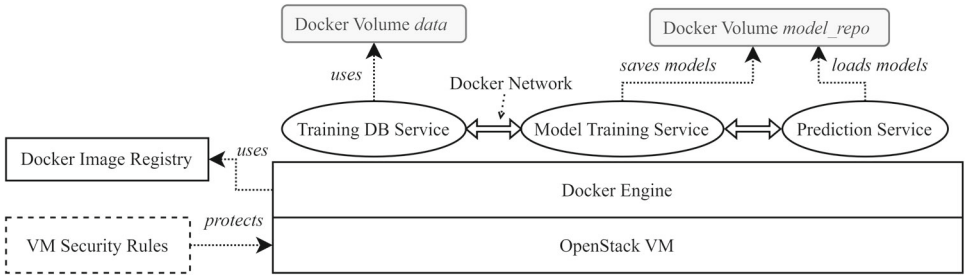


Fig. 17 ML application used for experiments

The M24 experiment consisted of three tasks.

- *Task M24.1* (for non-experts) is to develop a deployment model for the ML application in TOSCA using a plain YAML editor (Exercise A) and the same model in the SODALITE DSL using the SODALITE IDE (Exercise B). We selected a crossover experiment design [62] due to the relatively small sample size and the potential high variability in participants' knowledge and experience of the TOSCA. We randomly assigned the participants into two sub-groups (size of 4 and 5). To control for order effects in the experiment design, we used the ABBA (reverse counterbalancing) method [71]. The first sub-group realized the two exercises in the order, first A and then B, the other sub-group performed the exercises in the opposite order, first B and then A. A potential problem in our crossover design is the carryover effects, where participants' experience and perception gained (or what they have learned) from doing the first exercise can distort their experience and perception of the second exercise, causing bias. A potential solution is to use a sufficiently long washout period, which is the time between two exercises [62]. Unfortunately, due to the concerns, such as participants' preference to do all exercises in the same session and their unavailability for a separate session, we could not use a lengthy washout period (e.g., several weeks or months). Moreover, for software engineering tasks, it may not be practical to make the participants unlearn or forget what they have learned [62].
- *Task M24.2* (for TOSCA experts) performs the same task as non-experts. All TOSCA experts had prior experience using plain YAML editors (as plugins in commercial and open-source IDEs) to develop TOSCA files. Note that the SODALITE IDE can transform a deployment model in SODALITE DSL to the corresponding TOSCA-based model.
- *Task M24.3* (for owners of use cases) is to study the deployment models (in SODALITE DSL) for three SODALITE use cases in the 18th month (M18) and to modify the models by adding new nodes and relationships, and updating and removing some existing nodes and their relationships. The SODALITE project had an intermediate review in M18. Hence, the deployment models created for this milestone were sufficiently complex and already thoroughly tested.

The M37 experiment used the same ML application and had two tasks (for all participants).

- *Task M37.1* focuses on inspection and modification of the deployment model for the ML application. A partially completed deployment model was provided to participants who needed to understand and complete the model using the guidance provided by the IDE.
- *Task M37.2* focuses on resolving deployment issues. The SODALITE framework helps users generate IaC (TOSCA) from the models in the SODALITE DSL. In this step, they will use the offered help dialogs to inspect and solve the problems.

6.2.3 Procedure

Overall, the experiment process consisted of providing tutorials, conducting actual experiments, and collecting the participants' feedback through a questionnaire.

- **Tutorials** We created a tutorial for SODALITE IDE and shared it with the participants one week before the experiment session. The tutorial covered modeling IaC with the IDE and the basis of TOSCA. Moreover, we supplemented the tutorial guides with the videos prepared for the specific experiments.
- *Experimental tasks* Sect. 6.2.2 explains each task in detail. Each experiment session was conducted virtually using Microsoft Teams and Cisco Webex. At least one member of the SODALITE project observed and coordinated each session. In addition, we also provided participants with an experiment guide that explained each step of the experiment, the expected outcome, and the testing of the created artifacts.
- *Post-experiment surveys* After each experiment, an anonymous questionnaire was used to collect the participants' feedback. We used Microsoft Forms to create the questionnaire. Table 3 shows the key questions in the ques-

Table 3 Templates of key questions in the post-experiment surveys

Metric	Questions
Perceived ease of use	(a) How do you judge the perceived ease of use of the <<tool>> (Likert Scale (1–5)) (b) Please justify your answer. (open-ended)
Perceived usefulness	(a) How do you judge the perceived usefulness (e.g., reduce effort, effective in defining the deployment code correctly and intuitively) of the <<tool>>? Likert Scale (1–5) (b) Please justify your answer.(open-ended)
Intention to use	Indicate your agreement with the following statements. Likert Scale (1–3) 1. I see a potential in the adoption of the <<tool>> to automate software deployment. 2.It would be easy for me to become skilled in using the <<tool>>to automate software deployment. 3. I would recommend the <<tool>> to automate software deployment. 4.If I am working at a company in the future, I would like to use the <<tool>> to automate software deployment

Depending on the experiment, <<tool>>is SODALITE IDE or regular YAML editor

tionnaires that aim to measure three perception-based metrics: Perceived Ease of Use, Perceived Usefulness, and Intention to Use. The questionnaire also had questions to obtain some demographic data (non-sensitive) to understand participants' level of expertise, background, and experience with TOSCA tools. The selected perception-based metrics are considered some of the key factors that decide the user acceptance of a particular technology [17] and are recommended to measure the usefulness of a tool [31]. As shown in the table, we used both Likert scale questions and open-ended questions in the surveys. We used a 5-point scale for the two independent variables (perceived ease of use and perceived usefulness) and a 3-point scale for the dependent variable (intention to use). While the 5-point scale was balanced with an equal number of positive and negative response options and a neutral option, the 3-point scale was unbalanced (two positive and one negative option). An unbalanced scale, without a natural option, can be used when the researchers want to draw a line about a particular topic and get a clear opinion [30]. The choices for the scale can also vary depending on the researchers' knowledge of the target audience. We selected a 3-point scale (without the natural option) for the intention to use variables because we wanted to distinguish between the participants who are likely to adopt our tool and those who are not. However, the absence of a neutral response option can limit the response options and distort results. We added this as an internal threat to the validity of our findings.

6.2.4 Results and discussion

We collected and analyzed quantitative data (task completion time and Likert scale answers) and qualitative data (answers to open-ended questions).

Task completion. Task M24.1 (9 non-experts) and Task M24.2 (5 TOSCA experts) were completed by all participants. Regarding Task M24.3, all four use case owners could navigate their case studies' deployment models without specific problems. Three participants were able to extend and modify their deployment models, while one was only partially successful.

Among the 11 participants in the M37 experiment, 10 completed the modeling task successfully (Task M37.1). However, three could not continue with the IaC generation and deployment task (Task M37.2) due to a bug in the IDE on the Windows operating system. The bug was eventually fixed in the next release of the SODALITE IDE. Given the generated code, three participants could deploy and run the application. The others failed because they used an incorrect value of an input parameter for the TOSCA blueprint.

We also measured the average time it took to complete each task. Note that we collected this metric for the M24 experiment. We enforced a time limit for all tasks in M24 (120 min) and urged participants to complete the tasks as early as possible. However, in M37, since we had already measured the task time in M24, we decided to allow participants to complete their tasks freely without time pressure. Figure 18 shows the task completion time for the three M24 tasks. As expected, the TOSCA experts finished the task sooner than the non-expert students. In general, the SODALITE IDE could reduce the time to create a deployment model in TOSCA.

Perceived usefulness. Figures 19 and 20 show the distribution of the perceived usefulness metric. The average and standard deviation (for the M24 and M37 experiments) for the metric (Likert scale 1–5) were 3.82 and 0.72, respectively. Overall, the participants considered the SODALITE IDE helpful in developing TOSCA-based deployment models, and non-experts appreciated the usefulness of the SODALITE IDE more than TOSCA experts. The qualitative analysis of the answers to open-ended questions provided more insight into the participants' perception of the usefulness of the IDE.

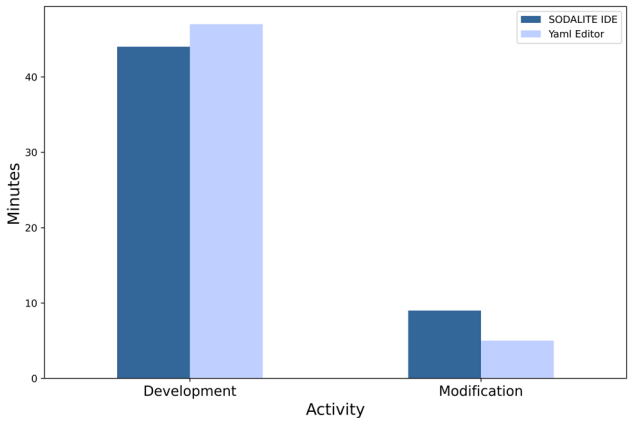


Fig. 18 Task completion time for M24 tasks: M24.1 and M24.2 for development and M24.3 for modification

The non-experts in the M24 experiment considered the SODALITE IDE useful due to its guidance features. They stated that those features can reduce the development effort and help write the IaC code correctly and intuitively. For example, one participant stated: *Especially auto-completion, highlighting, and verification of correctness is super useful. It speeds up development a lot I think and helps avoid annoying and unnecessary errors.* Another participant commented on the usefulness: *I found it very useful because normally YAML text editors (in my experience) don't show mistakes made context-wise (i.e., references to certain variables/parameters). I think this reduces mistakes and makes the overall development go much smoother.*

The TOSCA experts had a similar view on the usefulness of the SODALITE IDE. For example, one participant mentioned: *Autocompletion and suggestions significantly reduce the time for authoring the model. In this way, IDE enables the user to focus on the more complex part of the deployment model.* Another participant said: *When dealing with TOSCA node_types, it often happens that you keep forgetting the node_types definitions and also keep making mistakes which you notice only when deploying. The continuous assistance and model validation offered by SODALITE IDE helps to avoid these.*

The use case owners considered the SODALITE IDE useful, as they can define the whole deployment model with one tool, easily identify errors and warnings in the code, and get meaningful code recommendations. For instance, one owner stated: *I could (with not that much effort) define my deployment model and components relationship with one tool. Having hints and warnings when I'm doing something wrong is always a plus to prevent me from losing time (for syntactic errors, for example).*

The participants in the M37 experiments also commented on the usefulness of the guidance support in the IDE and its effect on reducing development time. For example, one participant said: *The suggestions are very helpful and insightful. Especially when combined with documentation, it makes it*

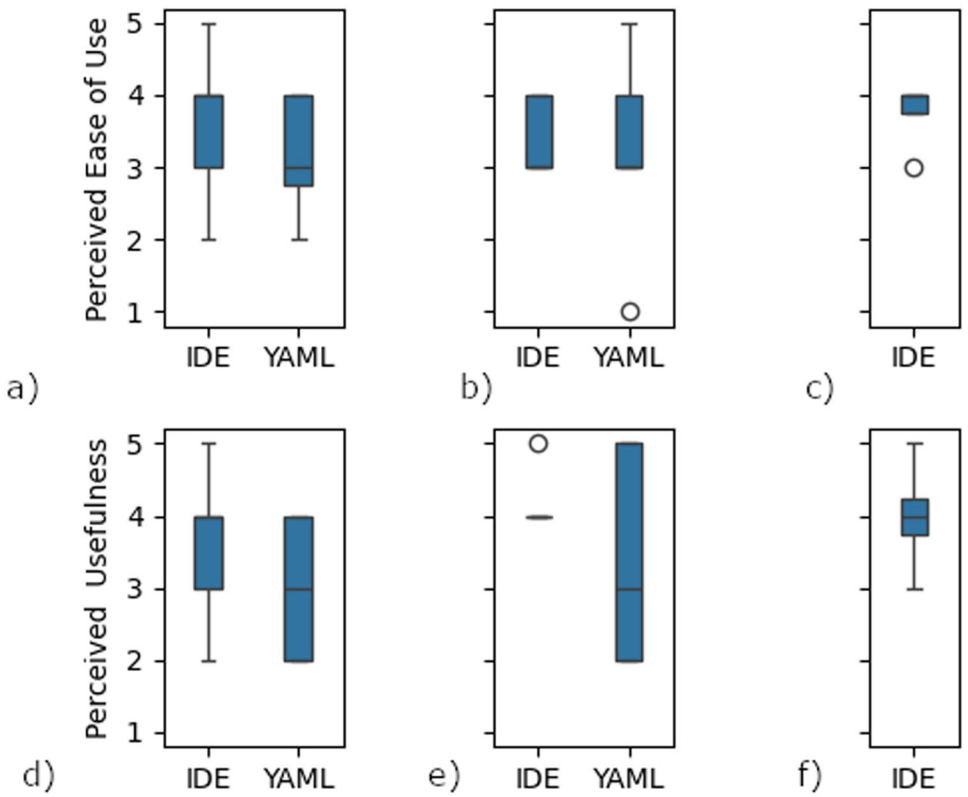


Fig. 19 The metrics (M24 experiment) for the perceived ease of use are depicted for: **a** non-experts, **b** experts, **c** use case owners. The metrics (M24 experiment) for the perceived usefulness are depicted for: **d** non-experts, **e** experts, **f** use case owners. IDE refers to SODALITE IDE, and YAML refers to a textual YAML editor

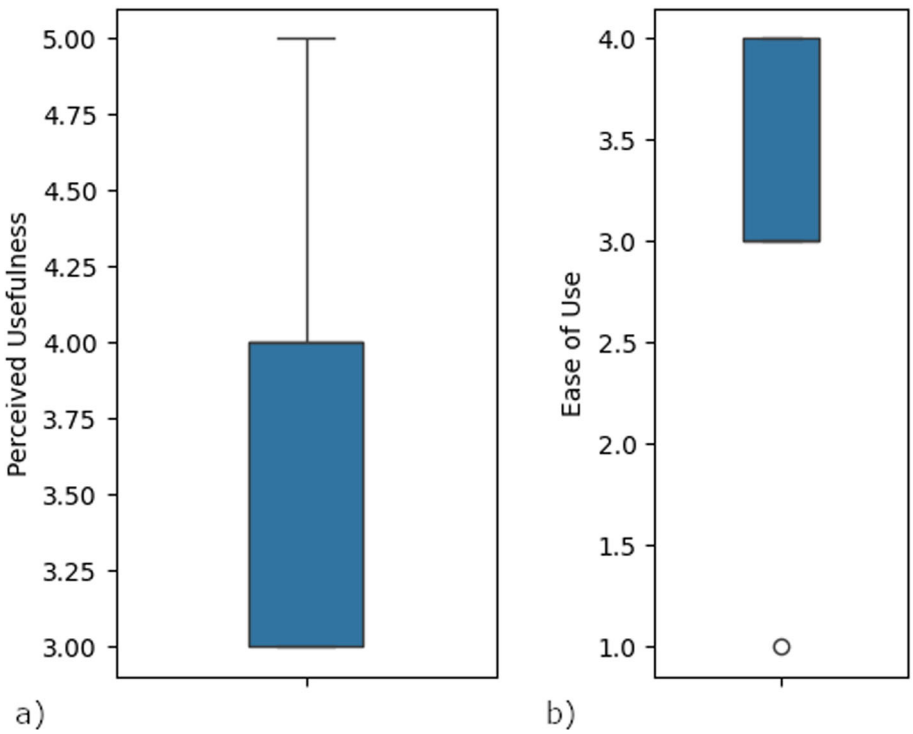


Fig. 20 **a** The metric perceived usefulness from M37 experiment. **b** The metric ease of use from M37 experiment for SODALITE IDE

easier to navigate through the different features of what you want to deploy. Another participant stated: *This [guidance support] reduces configuration time by a lot.*

The participants in M24 and M37 also made several suggestions to make the SODALITE IDE more useful: simplifying the installation process, making the tool available as a plug-in for the most popular general-purpose IDEs, improving the tool documentation, and fixing bugs in the tool. For example, a non-expert user (M24) stated: *I would rate it [SODALITE IDE] somewhere between neutral and useful, with the drawback of having to use Eclipse and the confusing installation.* Another non-expert said: *I thought setting this all up was difficult. And I would use it if it was an extension in VSCode, but not as a separate IDE.* A user case owner mentioned: *I believe it [SODALITE IDE] will become more useful as the bugs are ironed out and the IDE usage becomes more intuitive.*

Perceived ease of use. As shown in Figs. 19a, b, c and 20b, the participants generally considered the SODALITE IDE easier to use than a plain YAML editor for writing IaC. The average and standard deviation (across M24 and M37 experiments) for the metric (Likert scale 1–5) were 3.44 and 0.86, respectively. We also analyzed the answers to the open-ended question about perceived ease of use.

The non-experts in the M24 experiment found that the installation and configuration of the IDE were difficult. However, they observed that the IDE is easy and intuitive to use. For example, a participant stated: *I think the most difficult part was setting everything up and adjusting to the new system. After that, it was quite intuitive.* Another participant mentioned: *The user interface is very intuitive and easy to follow. Very little learning curve.* As noted by another two non-experts, the tutorials and documentation provided a head start to familiarize with the IDE: *By referring to the tutorials which were provided to us in advance, understanding the functioning and logic behind the SODALITE IDE was fairly straightforward.* Finally, according to the participants, the assistance support and the graphical user interface were the key factors that contributed to the usability of the SODALITE easier to use. A participant stated, *It was also nice to see your written model visualized and notified immediately of any contextual mistakes rather than just typos. This helped me not to get into any mistakes on the way and made it, therefore, easier to use.*

The experts and use case owners in the M24 experiment also expressed the difficulty of setting up the IDE. They also identified content assistance and graphical modeling as the facts that make the IDE easy to use. Furthermore, they suggested adding more assistance cases and improving documentation. For example, a TOSCA expert stated: *Code assist is helpful.... Have to be aware of tabs mixed with spaces. Configuring an unknown editor (window sizes, layout, etc.) can be annoying.* A user case owner commented: *Even though it takes some time to look through the different tags and get used to the usage of Eclipse, the perceived usage is, in the end, the one of a graph editor, which makes the IDE in my opinion quite intuitive and easy to understand.*

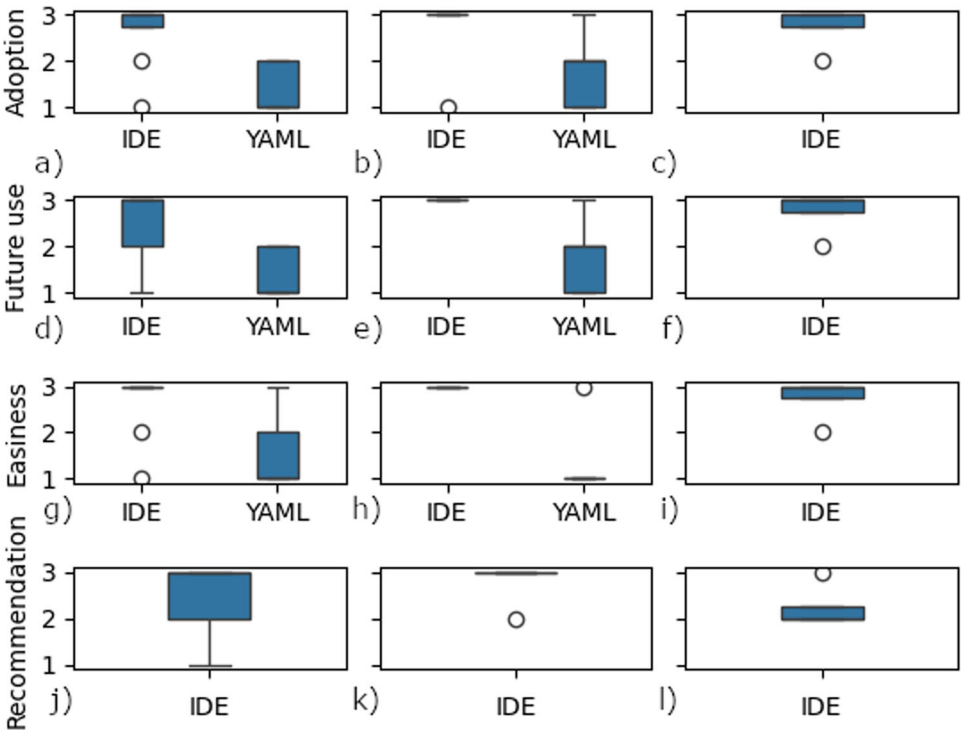


Fig. 21 Intention to use metric from M24 experiments: the potential adoption of the tool by non-experts (**a**), experts (**b**), use case owners (**c**); the willingness to adopt the use of the tool at a company in the future by non-experts (**d**), experts (**e**), use case owners (**f**); the easiness in becoming skilled in using the tool by non-experts (**g**), experts (**h**), use case owners (**i**); and the recommendation of the IDE to others by non-experts (**j**), experts (**k**), use case owners (**l**). IDE: SODALITE IDE and Yaml: a textual YAML editor

The participants in the M37 experiment had a similar view, especially related to the code assistance and the ease of setting up the IDE. For instance, one participant said: ...*However, the little I saw was very interesting and didn't seem too difficult to understand. Probably the hardest task is just to get used to the syntax, but the helper (ctrl + space bar) does a great job in supporting the user.* Another participant stated: *I can see that this would make a very complex problem easier.*

Intention to use Figures 21 and 22 show the distribution of the four variables of the intention to use metric. The average and standard deviation (across M24 and M37 experiments) for the metric (Likert scale 1–3) were 2.5 and 0.65, respectively. The non-experts expressed a solid intention to use SODALITE (88.9% agreement (i.e., answers as 'definitely true') for adopting the SODALITE approach in general, 66.7% for using the IDE for creating deployment code and using it in a company, 77.8% for becoming skilled in using it). The TOSCA experts and use case owners had a similar opinion, for example, 75% and 60% for adopting the SODALITE approach in general and no disagreements (i.e., answers as 'definitely false') on any of the four statements. However, the participants' opinions in the M7 experiments were less assertive but largely positive. For example, 45.5% of 'definitely true' and 45.5% of 'More true than false' for the statement about adopting the SODALITE approach, and there were only 5 'definitely false' answers (from 44) overall.

7 Discussion

7.1 Threats to validity

This section presents the potential threats to internal and external validity [65] that may apply to our study.

7.1.1 Internal validity

The biases in selecting participants for experiments can affect the results of the experiment. For example, TOSCA experts may consider the guidance features offered by our tool helpful since they are already knowledgeable about authoring TOSCA models. To mitigate this threat, we used both TOSCA experts and non-expert users and performed two iterations of experiments. However, the changes in the experiment setup (e.g., use case and participants) between iterations may produce differences in experiment outcomes. We partially mitigated this threat using the same case study and experiment procedures. However, we could not recruit the same participants for both iterations of the experiment, as some participants were unavailable. Another potential threat is using a 3-point scale to measure the intention to use variables in the post-experiment survey. A 3-point scale may be less reliable than a 5-point scale, and two different scales can also affect the internal consistency of the questionnaire.

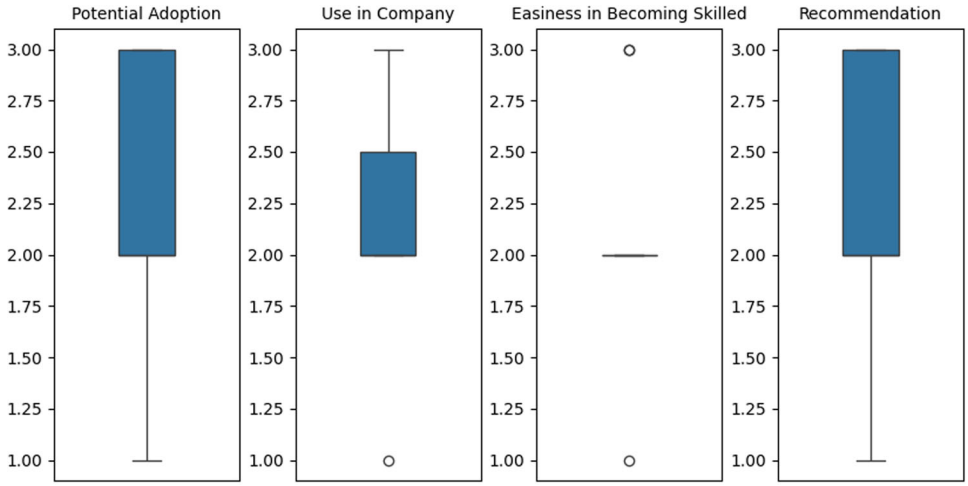


Fig. 22 Intention to use metric from M37 experiments: **a** potential adoption of the SODALITE IDE, **b** using the IDE at a company, **c** easiness in becoming skilled in the IDE, **d** recommendation of the IDE

7.1.2 External validity

The conclusions derived only from a set of small user groups may not be generalized. We tried to mitigate this threat by using different types of subjects, e.g., students, researchers, professionals, IaC experts, and IaC non-experts. Moreover, we developed three industrial applications using the SODALITE framework. Those use cases represent Cloud, HPC, and Edge applications and demonstrate diverse usage scenarios. Finally, our framework is based on the TOSCA IaC language, and the results may not generalize to other IaC languages. Applying our guided MDE approach to other IaC languages is part of our research agenda. As discussed in Sect. 3.2, the conceptual similarities between many IaC languages can make our approach applicable to other IaC languages. In fact, we applied our approach to writing Ansible task descriptions in which users are guided to select Ansible modules and their parameters .[20]

7.2 Mapping to MDE recommendation types

Almonte et al. [1] surveyed the recommender systems in model-driven engineering (MDE) and identified the main types of guidance/recommendation. Table 4 shows the mappings of the guidance types supported by our SODALITE framework to the common MDE recommendation types. The *Create* type makes recommendations for enabling the creation of a model, the *Find* type suggests relevant artifacts, and the *Reuse* recommends parts of existing models. The *Complete* type's purpose is to complete a partial model to make it correct. The *Matchmaking* guidance type corresponds to the *Create*, *Find*, and *Reuse* types as it is related to the suggestions during the creation of models, the querying of the KB for suggesting relevant artifacts, and the reuse of existing artifacts (node types and templates) in the KB correspondingly. The *Validation* guidance type is related to the *Reuse* and *Repair* types, as existing artifacts are reused to suggest correcting an erroneous model. The *Smell Detection* guidance type is related to the *Repair* type, as it enables the users to identify and fix quality issues in the models. The *Auto-completion* guidance type corresponds to the *Complete* and *Reuse* types as it supports auto-completing the models by filling in their missing information through reusing the existing knowledge in the KB.

Table 4 Mapping from SODALITE guidance types to common MDE recommendation types [1]

Our guidance type	Common MDE guidance types
Matchmaking	Create, Find, and Reuse
Validation	Reuse and Repair
Smell Detection	Repair
Auto-completion	Complete and Reuse

8 Related work

This section compares our approach with the relevant literature on IaC development. As necessary, we base our analysis on recent related surveys [1, 5, 15, 43, 68].

CloudCamp [6] is an MDE framework that can generate TOSCA and Ansible IaC scripts from a high-level model of the deployment topology of an application. To automate the generation of executable IaC from the technology-agnostic specifications of application components, CloudCamp uses a knowledge base, which is a relational database storing the software dependencies used by a particular application component type. Cloud DSL [56] and TOSCA Light [69] aim to simplify the development of TOSCA models. Both

[20] https://github.com/SODALITE-EU/ide/tree/master/dsl/documentation/ansible.

offer DSLs for modeling cloud application deployment models; Cloud DSL uses a custom meta-model, and TOSCA Light uses a subset of TOSCA. EDMM(Essential Deployment Meta-model) [66, 67] is another MDE toolchain that includes a DSL for technology-agnostic modeling of deployments and a rule-based transformation engine for generating technology-specific deployment artifacts. EDMM DSL is primarily inspired by TOSCA. Chiari et al. [14] developed another IaC modeling language, DevOps Modelling Language (DOML), which adopts and extends the EDMM approach to support infrastructure provisioning and runtime orchestration.

There exist several frameworks that support the deployment, monitoring, and management of multi-cloud applications using IaC, for example, SeaClouds [11], MODAClouds [19], CloudMF [22], Argon [52, 53], and OCCIware [72]. SeaClouds provides a graphical modeling tool for designing TOSCA-based deployment topologies. MODAClouds offers system developers and operators an IDE for authoring their deployment models in a DSL language, a decision support system for selecting cloud providers, and a runtime environment for monitoring and adapting the deployed applications. The CloudMF framework uses model-driven techniques to support provider-independent modeling of multi-cloud applications. Similarly, Argon also supports defining cloud infrastructure resources through a DSL language and generating provider-specific resource models. Piedade et al. [47] proposed a visual notation for modeling composite dockerized applications that can be deployed using the *docker compose* tool. Finally, OCCIware is a model-driven tool chain for OCCI (Open Cloud Computing Interface), offering support to design, validate, generate, and manage OCCI artifacts.

Several works have used MDE and IaC for deploying and managing applications on IoT, edge, and HPC environments [21, 23, 33, 58, 59]. Song et al. [58] extended CloudMF to support deploying software components across a fleet of edge and IoT devices. Their focus is on modeling device topologies and deployment plans. In [23], they extended their framework to support orchestrating serverless functions on cloud, edge, and IoT resources. Spataru et al. [59] proposed a TOSCA language extension that can model hybrid applications deployed on HPC and VM compute nodes consisting of hardware accelerators. eFlows4HPC [21] extended TOSCA to support the deployment and execution of complex data analytics workflows on HPC and cloud infrastructures. In [33], we presented our SODALITE middleware runtime that uses TOSCA and Ansible to orchestrate and manage applications on edge-cloud infrastructures.

Studies exist on discovering IaC scripts from the source code repositories [64], discovering resources in a given infrastructure, and generating IaC models for the discovered resources [9, 37, 50]. Wettinger et al. [64] developed a framework to crawl public repositories, locate Chef and Juju IaC scripts, and generate TOSCA-based models for the discovered IaC scripts. DrACO [9] can crawl the web to retrieve cloud offerings' information and generate the corresponding TOSCA representations. Rodríguez-García et al. [50] present a framework that can find the natural language descriptions of cloud services and generate the ontology-based descriptions for services. The generated models enable users to find and access cloud services that meet their requirements. Beniamino et al. [37] also employ ontology-based models of cloud services and design patterns to support the discovery and use of cloud services in creating cloud applications. Tortoise [63] is an interactive program repair tool for the Puppet IaC language. It enables system administrators to automatically propagate the manual changes made to the infrastructure back to the Puppet IaC scripts. Finally, several recent studies have used large language models (LLMs) for IaC. The main focus is generating and repairing IaC scripts [36, 60]. However, no studies provide an LLM-based approach that supports the interactive, guided, and collaborative development and maintenance of IaC scripts.

In a recent study [1], Almonte et al. systematically reviewed the literature to help modelers by providing (interactive) suggestions for various MDE activities. The types of MDE artifacts include meta-models, models, transformations, and code generators. The scope of the assistance encompasses the creation, completion, discovery, recommendation, reuse, and repair of artifacts. The recommendation methods include content-based, collaborative filtering, knowledge-based, or hybrid approaches. None of the selected studies in the review by Almonte et al. consider the recommender systems for IaC languages.

Table 5 summarizes the comparison of our approach with the related work. Guidance features include matchmaking and auto-completion, while validation features include model validation and smell detection. In the DSL column of the table, there are two possible values: TOSCA-based and TOSCA-inspired. The TOSCA-based refers to frameworks that use/support the TOSCA language. The TOSCA-inspired refers to frameworks that use a language influenced by TOSCA and its concepts. IaC frameworks may also support the collaborative development of IaC scripts.

Similarly to the other studies, we propose an MDE approach for IaC. We provide a DSL to model the deployment topology of an application in a technology-agnostic manner. The meta-model of our DSL is based on a subset of TOSCA. Instead of providing another modeling language for IaC, we aim to simplify the use of an existing modeling language by utilizing modeling assistance capabilities. In particular, our framework assists the modelers in developing the deployment models in TOSCA via interactive guidance. We primarily use a knowledge-based recommendation approach, where the domain knowledge about resources and applica-

Table 5 Comparative analysis of related work

Framework / Criteria	MDA	Open Standard	DSL	Validation	Guidance	EnableCollaboration
TOSCA Light	+	TOSCA	TOSCA-based	∼	–	–
ModaClouds	+	TOSCA	Custom	–	∼	–
CloudMF	+	TOSCA	Custom	–	–	–
Argon	+	TOSCA	Custom	–	–	–
OCCIware	+	OCCI	Custom	+	–	–
Seacloud	+	TOSCA	Custom	–	–	∼
EDDM	+	TOSCA	TOSCA-Inspired	∼	–	–
Our Approach	+	TOSCA	TOSCA-Based	+	+	+

Legend: + (available), ∼(partially available), – (not available)

tion components is captured and queried using semantic web technologies. Our framework is one of the first approaches to integrate MDE and ontologies in the IaC domain. Furthermore, the framework fosters the collaboration between REs (e.g., cloud experts and HPC experts) and AppOps. Moreover, most of the related work focuses only on cloud applications. In contrast, we consider cloud, edge, HPC, and hybrid applications.

The experiments showed that our approach could make IaC modeling effective and efficient as it enables users to employ interactive assistance to detect missing parts in the model, complete those parts semi-automatically, and validate models. Our findings also agree with the existing literature on guided model-driven engineering. For example, from an interview-based study, Savary-Leblanc et al. [54] found that the completion and validation of the models are the most critical features for a modeling assistant. Mora Segura et al. [38] found that modeling assistants that reuse domain knowledge can help users create more accurate models efficiently.

9 Conclusion

We presented a knowledge-based framework that can provide guidance in authoring deployment models in TOSCA for complex hybrid applications. We enable a cross-functional team in collaboration to develop deployment blueprints in TOSCA. Resource experts (REs) can share knowledge about resources, and Application operational experts (AppOps) can use this knowledge to select resources for application components, design the deployment topology, and ensure the quality of the created TOSCA blueprints. We use ontologies to represent relevant knowledge formally and use ontology reasoning to implement four types of guidance: matchmaking, validation, smell detection, and auto-completion. We showed the feasibility and usefulness of your approach through three industrial case studies and control experiments with TOSCA experts and non-experts.

We foresee several research directions.

Firstly, our framework can be extended in various ways. For example, new types of modeling guidance can be developed, such as recommending deployment alternatives [35] and suggesting fixes to model quality issues. Secondly, within the SODALITE project, we developed an experimental tool that can assist in completing a subset of configurations in Ansible IaC scripts (playbooks). This tool can be enhanced by incorporating ontological modeling of Ansible-related expert knowledge. Another potential extension is the pattern-based search and matching of IaC model elements [2, 27].

Secondly, future studies can investigate the potential of our approach in heterogeneous and decentralized IaC environments. For example, most IaC projects generally use multiple languages as there is no one-size-fits-all language [43]. An ontology-based approach can potentially support the integration and interoperability of different IaC languages in such projects. Moreover, several research studies exist on IaC approaches where cross-DevOps teams provision infrastructures, deploy, and configure applications in a decentralized manner [28, 46, 57]. The application of our approach to decentralized IaC solutions can also be a potential research topic.

Finally, LLMs (Large Language Models) align well with the ultimate objective of our research project, which is to build an intelligent IaC coding assistant. While some studies have already started exploring the potential of LLMs for IaC, none provide interactive guidance for developing and maintaining IaC scripts. In addition, knowledge graphs/ontologies can enhance the precision and interpretability of LLMs [45]. We plan to explore this synergy between a knowledge-based approach and LLMs in the context of IaC development and maintenance.

Acknowledgements This work is supported by the European Commission grant no. 825480 (H2020), SODALITE. We thank all members of the SODALITE consortium for their input and feedback on the development of this paper.

Funding Open access funding provided by HEAL-Link Greece.

References

1. Almonte, L., Guerra, E., Cantador, I., de Lara, J.: Recommender systems in model-driven engineering. Softw. Syst. Model. **21**(1), 249–280 (2022). https://doi.org/10.1007/s10270-021-00905-x
2. Antequera, R.B., Calyam, P., Chandrashekara, A.A., Mitra, R.: Recommending heterogeneous resources for science gateway applications based on custom templates composition. Futur. Gener. Comput. Syst. **100**, 281–297 (2019). https://doi.org/10.1016/j.future.2019.04.049
3. Atrey, A., Moens, H., Van Seghbroeck, G., Volckaert, B., De Turck, F.: An overview of the OASIS TOSCA standard: topology and orchestration specification for cloud applications (2015)
4. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. Empir. Softw. Eng. **27**(4), 94 (2022). https://doi.org/10.1007/s10664-021-10072-8
5. Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., Leymann, F.: A systematic review of cloud modeling languages. ACM Comput. Surv. **51**(1), 1–38 (2018). https://doi.org/10.1145/3150227
6. Bhattacharjee, A., Barve, Y., Gokhale, A., Kuroda, T.: (WIP) CloudCAMP: automating the deployment and management of cloud services. In: 2018 IEEE International Conference on Services Computing (SCC), pp. 237–240 (2018). https://doi.org/10.1109/SCC.2018.00038
7. Blomqvist, E., Hitzler, P., Janowicz, K., Krisnadhi, A., Narock, T., Solanki, M.: Considerations regarding ontology design patterns. Semant. Web **7**, 1–7 (2015). https://doi.org/10.3233/SW-150202
8. Borovits, N., Kumara, I., Di Nucci, D., Krishnan, P., Palma, S.D., Palomba, F., Tamburri, D.A., Heuvel, W.J.: FindICI: using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code. Empir. Softw. Eng. **27**(7), 178 (2022). https://doi.org/10.1007/s10664-022-10215-5
9. Brogi, A., Cifariello, P., Soldani, J.: DrACO: discovering available cloud offerings. Comput. Sci. Res. Dev. **32**(3), 269–279 (2017). https://doi.org/10.1007/s00450-016-0332-5
10. Brogi, A., Di Tommaso, A., Soldani, J.: Sommelier: a tool for validating TOSCA application topologies, pp. 1–22 (2018). https://doi.org/10.1007/978-3-319-94764-8_1
11. Brogi, A., Fazzolari, M., Ibrahim, A., Soldani, J., Wang, P., Carrasco, J., Cubo, J., Durán, F., Pimentel, E., Di Nitto, E., D'Andria, F.: Adaptive management of applications across multiple clouds: the seaclouds approach. CLEI Electron. J. **18**, 2–2 (2015). https://doi.org/10.19153/cleiej.18.1.1
12. Burgueño, L., Clarisó, R., Gérard, S., Li, S., Cabot, J.: An NLP-based architecture for the autocompletion of partial domain models. In: Advanced Information Systems Engineering, pp. 91–106 (2021)
13. Challita, S., Korte, F., Erbel, J., Zalila, F., Grabowski, J., Merle, P.: Model-based cloud resource management with TOSCA and OCCI. Softw. Syst. Model. **20**(5), 1609–1631 (2021). https://doi.org/10.1007/s10270-021-00869-y
14. Chiari, M., Xiang, B., Nedeltcheva, G.N., Di Nitto, E., Blasi, L., Benedetto, D., Niculut, L.: DOML: a new modelling approach to infrastructure-as-code. In: Indulska, M., Reinhartz-Berger, I., Cetina, C., Pastor, O. (eds.) Advanced Information Systems Engineering, pp. 297–313. Springer Nature Switzerland, Cham (2023)
15. Conto, J., Zúñiga Prieto, M., Solano-Quinde, L.: A Systematic Mapping Study of Specification Languages in Cloud Services Development: 4th International Conference, CITT 2018, Babahoyo, Ecuador, August 29–31, 2018, Revised Selected Papers, pp. 72–88 (2019). https://doi.org/10.1007/978-3-030-05532-5_6
16. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. **13**(3), 319–340 (1989)
17. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. **13**(3), 319–340 (1989)
18. Di Nitto, E., Cruz, J.G., Kumara, I., Radolović, D., Tokmakov, K., Vasileiou, Z.: Deployment and Operation of Complex Software in Heterogeneous Execution Environments: The SODALITE Approach. Springer Nature, Berlin (2022)
19. Di Nitto, E., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.S., Petcu, D., Gericke, A., Sheridan, C.: MODAClouds: a model-driven approach for the design and execution of applications on multiple clouds. ICSE Workshop on Modeling in Software Engineering (MISE 2012), pp. 50–56 (2012). https://doi.org/10.1109/MISE.2012.6226014
20. Díaz, J., López-Fernández, D., Pérez, J., González-Prieto, Á.: Why are many businesses instilling a DevOps culture into their organization? Empir. Softw. Eng. **26**(2), 1–50 (2021)
21. Ejarque, J., Badia, R.M., Albertin, L., Aloisio, G., Baglione, E., Becerra, Y., Boschert, S., Berlin, J.R., D'Anca, A., Elia, D., Exertier, F., Fiore, S., Flich, J., Folch, A., Gibbons, S.J., Koldunov, N., Lordan, F., Lorito, S., Løvholt, F., Macías, J., Marozzo, F., Michelini, A., Monterrubio-Velasco, M., Pienkowska, M., de la Puente, J., Queralt, A., Quintana-Ortí, E.S., Rodríquez, J.E., Romano, F., Rossi, R., Rybicki, J., Kupczyk, M., Selva, J., Talia, D., Tonini, R., Trunfio, P., Volpe, M.: Enabling dynamic and intelligent workflows for hpc, data analytics, and AI convergence. Fut. Gener. Comput. Syst. **134**, 414–429 (2022). https://doi.org/10.1016/j.future.2022.04.014
22. Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., Solberg, A.: CloudMF: model-driven management of multi-cloud applications. ACM Trans. Internet Technol. **18**(2), 1–24 (2018). https://doi.org/10.1145/3125621
23. Ferry, N., Dautov, R., Song, H.: Towards a model-based serverless platform for the cloud-edge-IoT continuum. In: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 851–858 (2022). https://doi.org/10.1109/CCGrid54584.2022.00101
24. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening ontologies with DOLCE. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, pp. 166–181. Springer-Verlag, Berlin, Heidelberg (2002)
25. Gangemi, A., Mika, P.: Understanding the semantic web through descriptions and situations. In: CoopIS/DOA/ODBASE (2003). https://doi.org/10.1007/978-3-540-39964-3_44
26. Guerriero, M., Garriga, M., Tamburri, D.A., Palomba, F.: Adoption, support, and challenges of infrastructure-as-code: insights from industry. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 580–589. IEEE (2019)
27. Harzenetter, L., Breitenbücher, U., Falkenthal, M., Guth, J., Krieger, C., Leymann, F.: Pattern-based deployment models and

their automatic execution. In: 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), pp. 41–52 (2018).https://doi.org/10.1109/UCC.2018.00013
28. Imam, F.T.: Application of ontologies in cloud computing: the state-of-the-art. ArXiv:abs/1610.02333 (2016). https://api.semanticscholar.org/CorpusID:9940694
29. Indika, K., , Radolović, D., Cruz, J.G., Vasileiou, Z., Tokmakov, K.: Orchestrating data-intensive applications on federated hybrid infrastructures with the sodalite framework. https://indico.egi.eu/event/5464/contributions/15656/contribution.pdf (2021)
30. Johns, R.: Likert items and scales. Survey Question Bank: Methods Fact Sheet **1**(1), 11–28 (2010)
31. Ko, A.J., LaToza, T.D., Burnett, M.M.: A practical guide to controlled experiments of software engineering tools with human participants. Empir. Softw. Eng. **20**(1), 110–141 (2015). https://doi.org/10.1007/s10664-013-9279-3
32. Kumara, I., Garriga, M., Romeu, A.U., Di Nucci, D., Palomba, F., Tamburri, D.A., van den Heuvel, W.J.: The do's and don'ts of infrastructure code: a systematic gray literature review. Inf. Softw. Technol. **137**, 106593 (2021)
33. Kumara, I., Mundt, P., Tokmakov, K., Radolović, D., Maslennikov, A., González, R., Fernández Fabeiro, J., Quattrocchi, G., Meth, K., Nitto, E., Tamburri, D., Heuvel, W.J., Meditskos, G.: Sodalite@rt: orchestrating applications on cloud-edge infrastructures. J. Grid Comput. **19**, 29 (2021)
34. Kumara, I., Vasileiou, Z., Meditskos, G., Tamburri, D.A., Heuvel, W.V.D., Karakostas, A., Vrochidis, S., Kompatsiaris, I.: Towards semantic detection of smells in cloud infrastructure code. In: WIMS 2020: Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics pp. 63–67 (2020)
35. Kumara, I.P., Ariz, M., Baruwal Chhetri, M., Mohammadi, M., Heuvel, W.J.V.D., Tamburri, D.A.A.: FOCloud: feature model guided performance prediction and explanation for deployment configurable cloud applications. IEEE Trans. Services Comput. (2022). https://doi.org/10.1109/TSC.2022.3142853
36. Low, E., Cheh, C., Chen, B.: Repairing infrastructure-as-code using large language models. In: 2024 IEEE Secure Development Conference (SecDev), pp. 20–27 (2024). https://doi.org/10.1109/SecDev61143.2024.00008
37. Martino, B.D., Esposito, A., Cretella, G.: Semantic representation of cloud patterns and services with automated reasoning to support cloud application portability. IEEE Trans. Cloud Comput. **5**(4), 765–779 (2017). https://doi.org/10.1109/TCC.2015.2433259
38. MoraSegura, Á., de Lara, J., Wimmer, M.: Modelling assistants based on information reuse: a user evaluation for language engineering. Softw. Syst. Model. (2023). https://doi.org/10.1007/s10270-023-01094-5
39. Morris, K.: Infrastructure as Code: Managing Servers in the Cloud. O'Reilly Media Inc, Sebastopol (2016)
40. Motik, B.: On the properties of metamodeling in owl. J. Log. Comput. **17**(4), 617–637 (2007). https://doi.org/10.1093/logcom/exm027
41. Nasiri, R., Kumara, I., Tamburri, D.A., van den Heuvel, W.J.: Towards a taxonomy of infrastructure as code misconfigurations: an ansible study. In: Aiello, M., Barzen, J., Dustdar, S., Leymann, F. (eds.) Service-Oriented Computing, pp. 83–103. Springer Nature Switzerland, Cham (2025)
42. Nieke, M., Sampaio, G., Thüm, T., Seidl, C., Teixeira, L., Schaefer, I.: Guiding the evolution of product-line configurations. Softw. Syst. Model. **21**(1), 225–247 (2022). https://doi.org/10.1007/s10270-021-00906-w
43. Novakova Nedeltcheva, G., De La Fuente Ruiz, A., Orue-Echevarria Arrieta, L., Bat, N., Blasi, L.: Towards supporting the generation of infrastructure as code through modelling approaches - systematic literature review. In: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), pp. 210–217 (2022). https://doi.org/10.1109/ICSA-C54293.2022.00048
44. OASIS: Tosca primer (2013). http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/toscaprimer-v1.0-cnd01.pdf
45. Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., Wu, X.: Unifying large language models and knowledge graphs: a roadmap. IEEE Trans. Knowl. Data Eng. **36**(7), 3580–3599 (2024). https://doi.org/10.1109/TKDE.2024.3352100
46. Philippe, J., Omond, A., Coullon, H., Prud'Homme, C., Raïs, I.: Fast choreography of cross-DevOps reconfiguration with ballet: a multi-site openstack case study. In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 1–11 (2024). https://doi.org/10.1109/SANER60148.2024.00007
47. Piedade, B., Dias, J.P., Correia, F.F.: Visual notations in container orchestrations: an empirical study with docker compose. Softw. Syst. Model. **21**(5), 1983–2005 (2022). https://doi.org/10.1007/s10270-022-01027-8
48. Rahman, A., Mahdavi-Hezaveh, R., Williams, L.: A systematic mapping study of infrastructure as code research. Inf. Softw. Technol. **108**, 65–77 (2019). https://doi.org/10.1016/j.infsof.2018.12.004
49. Rahman, A., Parnin, C., Williams, L.: The seven sins: security smells in infrastructure as code scripts. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 164–175 (2019). https://doi.org/10.1109/ICSE.2019.00033
50. Ángel Rodríguez-García, M., Valencia-García, R., García-Sánchez, F., Samper-Zapater, J.J.: Ontology-based annotation and retrieval of services in the cloud. Knowl. Based Syst. **56**, 15–25 (2014). https://doi.org/10.1016/j.knosys.2013.10.006
51. Saini, R., Mussbacher, G., Guo, J.L.C., Kienzle, J.: Automated, interactive, and traceable domain modelling empowered by artificial intelligence. Softw. Syst. Model. **21**(3), 1015–1045 (2022). https://doi.org/10.1007/s10270-021-00942-6
52. Sandobalin, J., Insfran, E., Abrahão, S.: Argon: a model-driven infrastructure provisioning tool. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 738–742 (2019). https://doi.org/10.1109/MODELS-C.2019.00114
53. Sandobalín, J., Insfran, E., Abrahão, S.: On the effectiveness of tools to support infrastructure as code: model-driven versus code-centric. IEEE Access **8**, 17734–17761 (2020). https://doi.org/10.1109/ACCESS.2020.2966597
54. Savary-Leblanc, M., Le Pallec, X., Gérard, S.: Understanding the need for assistance in software modeling: interviews with experts. Softw. Syst. Model. (2023). https://doi.org/10.1007/s10270-023-01104-6
55. Sharma, T., Spinellis, D.: A survey on software smells. J. Syst. Softw. **138**, 158–173 (2017). https://doi.org/10.1016/j.jss.2017.12.034
56. Silva, G., Rose, L., Calinescu, R.: Cloud DSL: A language for supporting cloud portability by describing cloud entities. CEUR Workshop Proc. **1242**, 36–45 (2014)
57. Sokolowski, D., Weisenburger, P., Salvaneschi, G.: Automating serverless deployments for DevOps organizations. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, pp. 57–69. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3468264.3468575
58. Song, H., Dautov, R., Ferry, N., Solberg, A., Fleurey, F.: Model-based fleet deployment in the IoT-edge-cloud continuum. Softw. Syst. Model. (2022). https://doi.org/10.1007/s10270-022-01006-z
59. Spătaru, A., Iuhasz, G., Panica, S.: Tufa: A TOSCA extension for the specification of accelerator-aware applications in the cloud con-

tinuum. In: 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 1178–1183 (2022). https://doi.org/10.1109/COMPSAC54236.2022.00185
60. Srivatsa, K.G., Mukhopadhyay, S., Katrapati, G., Shrivastava, M.: A survey of using large language models for generating infrastructure as code. In: J. D. Pawar, S. Lalitha Devi (eds.) Proceedings of the 20th International Conference on Natural Language Processing (ICON), pp. 523–533. NLP Association of India (NLPAI), Goa University, Goa, India (2023). https://aclanthology.org/2023.icon-1.48
61. Studer, R., Staab, S.: Handbook on Ontologies, vol. 2. Springer, Berlin (2004)
62. Vegas, S., Apa, C., Juristo, N.: Crossover designs in software engineering experiments: benefits and perils. IEEE Trans. Softw. Eng. **42**(2), 120–135 (2016). https://doi.org/10.1109/TSE.2015.2467378
63. Weiss, A., Guha, A., Brun, Y.: Tortoise: Interactive system configuration repair. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 625–636 (2017). https://doi.org/10.1109/ASE.2017.8115673
64. Wettinger, J., Breitenbücher, U., Kopp, O., Leymann, F.: Streamlining DevOps automation for cloud applications using TOSCA as standardized metamodel. Fut. Gener. Comput. Syst. **56**, 317–332 (2016). https://doi.org/10.1016/j.future.2015.07.017
65. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media, Berlin (2012)
66. Wurster, M., Breitenbücher, U., Brogi, A., Falazi, G., Harzenetter, L., Leymann, F., Soldani, J., Yussupov, V.: The EDMM modeling and transformation system. In: Yangui, S., Bouguettaya, A., Xue, X., Faci, N., Gaaloul, W., Yu, Q., Zhou, Z., Hernandez, N., Nakagawa, E.Y. (eds.) Service-Oriented Computing—ICSOC 2019 Workshops, pp. 294–298. Springer International Publishing, Cham (2020)
67. Wurster, M., Breitenbücher, U., Brogi, A., Harzenetter, L., Leymann, F., Soldani, J.: Technology-agnostic declarative deployment automation of cloud applications. In: Brogi, A., Zimmermann, W., Kritikos, K. (eds.) Service-Oriented and Cloud Computing, pp. 97–112. Springer International Publishing, Cham (2020)
68. Wurster, M., Breitenbucher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., Soldani, J.: The essential deployment metamodel: a systematic review of deployment automation technologies. SICS Softw.-Intensive Cyber-Phys. Syst. **35**(1), 63–75 (2020)
69. Wurster, M., Breitenbücher, U., Harzenetter, L., Leymann, F., Soldani, J.: Tosca lightning: an integrated toolchain for transforming TOSCA light into production-ready deployment technologies. In: Herbaut, N., La Rosa, M. (eds.) Advanced Information Systems Engineering, pp. 138–146. Springer International Publishing, Cham (2020)
70. Wurster, M., Breitenbucher, U., Harzenetter, L., Leymann, F., Soldani, J., Yussupov, V.: TOSCA light: bridging the gap between the TOSCA specification and production-ready deployment technologies. In: CLOSER (2020)
71. Yanju, R.: Abba counterbalancing. In: The ECPH Encyclopedia of Psychology, pp. 1–1. Springer (2025)
72. Zalila, F., Challita, S., Merle, P.: Model-driven cloud resource management with OCCIware. Fut. Gener. Comput. Syst. **99**, 260–277 (2019). https://doi.org/10.1016/j.future.2019.04.015
73. Zimmermann, M., Breitenbücher, U., Krieger, C., Leymann, F.: Deployment enforcement rules for TOSCA-based applications. In: Proceedings of The Twelfth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2018), pp. 114–121. Xpert Publishing Services (2018)

Zoe Vasileiou holds a Master's degree in Computer and Communication Engineering from the University of Thessaly (UTH), Greece. She has gained experience in industry and research environments, contributing to various R&D projects. Recently, she has been involved in the development of Explainable Artificial Intelligence (XAI) models and Decision Support Systems with a focus on Semantic technologies. Her primary interests include neurosymbolic AI, Semantic Web technologies, and Machine Learning.

Indika Kumara is an assistant professor (UD1) at the Jheronimus Academy of Data Science and Tilburg University, the Netherlands. He received his PhD degree from Swinburne University of Technology, Australia. His research interests include data mesh, data and knowledge engineering, machine learning operations, service-oriented computing, cloud computing, infrastructure as code, and quantum computing.

Georgios Meditskos received in 2009 the PhD degree in Informatics from Aristotle University of Thessaloniki in Greece for his dissertation on "Semantic Web Service Discovery and Ontology Reasoning using Entailment Rules". He also holds an MSc and a BSc degree from the same department. In 2012, he joined the Information Technologies Institute (ITI) of the Center for Research and Technology Hellas (CERTH) as a postdoctoral researcher. Since July 2021, he has been an Assistant Professor in the School of Informatics at the Aristotle University of Thessaloniki, Greece. He is the author of more than 100 publications in refereed journals and international conferences. He has actively participated in more than 10 European and National projects relevant to Cultural Heritage, Health, Cybersecurity, Disaster Management, Virtual Environments, and Agents. His research interests revolve around Symbolic AI, and more specifically, on knowledge representation and reasoning in the Semantic Web (RDF/OWL, rule-based ontology reasoning, combination of rules and ontologies), Knowledge Graphs, and context-based multi-sensor reasoning and fusion in Pervasive Environments.

Kamil Tokmakov is a research engineer at the High Performance Computing Center in Stuttgart (HLRS). He received a master's degree in communications technology from Ulm University, Germany. His main research interests include the integration of cloud technologies and services with HPC.

Dragan Radolović is a project manager at XLAB Research. He received a bachelor's degree in computer science and information technology from Ljubljana University. His background includes work as a software architect and project manager in the financial field (banks, financial institutions) and other software industries. His main areas of interest are cloud architectures.

Jesús Gorroñogoitia Cruz BS in Theoretical Physics from the Universidad Complutense de Madrid (UCM), with a Master in Condensed Matter and Statistics Physics by UNED. At Atos Research & Innovation (ARI), he is a senior researcher and software architect in the Advanced Parallel Computing Lab, specialized in HPC orchestration. Before, he was the ARI Research Line Expert on Software Engineering, working on Service Oriented Computing (SOC), Model Driven Development (MDD), Autonomous Computing, or Semantics.

Elisabetta Di Nitto is a full professor at Politecnico di Milano. She received a PhD in computer science and automation from Politecnico di Milano. Her expertise lies in the area of software engineering and, in particular, of large-scale, open, service-oriented systems with a special attention to the techniques to make these systems self-adaptable to the changes in the environment and in the performances of its distributed components, and to enable them to identify and incorporate new components at runtime. Recently, she has been focusing on Cloud Computing and, in particular, on how to design applications that can run on multiple clouds in order to limit vendor lock-in and to increase the availability and reliability of such applications.

Damian Andrew Tamburri Damian is an Associate Professor at the University of Sannio in Benevento and principal investigator for JADS/NXP Semiconductors in Eindhoven (NL). His research interests lie mainly in Data-Intensive Software Architectures (with a focus on Big Data & Data-Intensive Architectures, AI for Cloud & Microservices as well as Machine-Learning & Computational Intelligence Architectures) Operations, with a focus on Architecture Properties (with a focus on Privacy & Security as well as Social Sustainability), and Empirical Software Engineering (with a focus on Organisational, Social, and Societal aspects). What is more, ever since his engagement with the OASIS TOSCA Standardisation Technical Committee, Damian has picked up a considerable interest in Site Reliability and Infrastructure Code maintenance and evolution. Damian has published over 200+ papers in either top Journals such as the Transactions on Software Engineering (TSE) Journal, The ACM Computing Surveys (CSUR) Journal, or top software, service, and data engineering conferences (such as ICSE or FSE, IEEE SERVICES, IEEE BigData). Also, Damian has been an active contributor and led research in many EU FP6, FP7, H2020, and Horizon Europe projects. Damian is an ACM TOSEM editorial board member and online presence director, secretary of the OASIS TOSCA Standardisation TC as well as secretary of the IFIP TC2, TC6, and TC8 WG on "Service-Oriented Computing". Damian is very sociable, so ask any question you wish of him and he'll do his best to answer.

Willem-Jan Van Den Heuvel is a full professor in Information Systems, Data Governance, and Data Engineering at the Jheronimus Academy of Data Science and Tilburg University, the Netherlands. He is the managing director of the European Research Institute of Services Science (ERISS). His research interests are at the cross-junction of software service systems and business process management with an emphasis on (global) networked enterprises.

Stefanos Vrochidis received the Diploma degree in Electrical Engineering from Aristotle University of Thessaloniki, Greece, the MSc degree in Radio Frequency Communication Systems from the University of Southampton and the PhD degree in Electronic Engineering from Queen Mary University of London. Currently, he is a Principal Researcher with the Information Technologies Institute of the Centre for Research and Technology Hellas (ITI-CERTH), the Head of the Multimodal Data Fusion and Analytics (M4D) Group, and co-founder of Infalia and CDXi Solutions. His research interests include multimedia understanding and retrieval, multimodal fusion, computer vision, multimodal analytics, artificial intelligence, as well as health, industrial, media & arts, environmental and security applications. Dr. Vrochidis has participated in more than 100 European and National projects (in more than 20 as project coordinator or Scientific/Technical Manager) and has been a member of the organization team of several conferences and workshops. He has edited 3 books and authored more than 400 related scientific journal, conference, and book chapter publications. He has served as a reviewer in several international Journals and as part of the Organisational and Technical program committee in well-reputed conferences and workshops.

Authors and Affiliations

Zoe Vasileiou[1] · Indika Kumara[2] · Georgios Meditskos[1] · Kamil Tokmakov[4] · Dragan Radolović[5] · Jesús Gorroñogoitia Cruz[6] · Elisabetta Di Nitto[7] · Damian Andrew Tamburri[3] · Willem-Jan Van Den Heuvel[2] · Stefanos Vrochidis[8]

✉ Zoe Vasileiou
zvasileiou@csd.auth.gr, zvasilei@iti.gr

✉ Indika Kumara
i.p.k.weerasinghadewage@tilburguniversity.edu

Georgios Meditskos
gmeditsk@csd.auth.gr

Kamil Tokmakov
kamil.tokmakov@hlrs.de

Dragan Radolović
dragan.radolovic@xlab.si

Jesús Gorroñogoitia Cruz
jesus.gorronogoitia@atos.net

Elisabetta Di Nitto
elisabetta.dinitto@polimi.it

Damian Andrew Tamburri
d.a.tamburri@tue.nl

Willem-Jan Van Den Heuvel
w.j.a.m.vdnheuvel@tilburguniversity.edu

Stefanos Vrochidis
stefanos@iti.gr

[1] Information Technologies Institute, Centre for Research and Technology (CERTH) and School of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

[2] Jheronimus Academy of Data Science (JADS) and Tilburg University, Tilburg, The Netherlands

[3] JADS and Eindhoven University of Technology, Eindhoven, The Netherlands

[4] High Performance Computing Center (HLRS), University of Stuttgart, Stuttgart, Germany

[5] XLAB Research, Ljubljana, Slovenia

[6] ATOS, Madrid, Spain

[7] Politecnico di Milano, Milan, Italy

[8] CERTH, Marousi, Greece