Sodalite

SOftware Defined AppLication Infrastructures managemenT and Engineering

SODALITE Framework - First Version

D6.5

IBM

31.7.2020

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

Deliverable data			
Deliverable	D6.5 SODALITE framework - First version		
Authors	Kalman Meth (IBM) Indika Kumara (JADS) Anastasios Karakostas, Stefanos Vrochidis, Ioannis Kompatsiaris (CERTH) Yosu Gorroñogoitia (ATOS) Román Sosa González (ATOS) Giovanni Quattrocchi (PMI) Dragan Radolović (XLAB) Nejc Bat (XLAB)		
Reviewers	Piero Fraternali (POLIMI) Daniel Vladušič (XLAB)		
Dissemination level	Public, DEM		
History of changes	Name	Change	Date
	Kalman Meth (IBM)	Outline created	17.11.2019
	All	Components added	07.01.2020
	Piero Fraternali, Daniel Vladušič	First quality check & review	09.01.2020
	Kalman Meth (IBM)	Addressed issues from review	22.01.2020
	Kalman Meth (IBM)	Added section 1.2 "SODALITE Context and Goals" to address comments from reviewers. In introduction, added a statement on what is and what is not accomplished in this version, and what we plan for CI/CD. In section 1.3, added an explanation on the innovations of each layer after each layer figure.	19.07.2020

Acknowledgement

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

Table of Contents

List Of Images

Executive Summary

The SODALITE Framework is the software system that includes all SODALITE stable components. This deliverable includes the description of the software that makes up the SODALITE stack, while the actual software is available through GitHub SODALITE repository (https://github.com/SODALITE-EU). The document thus serves as an accompanying textual document, describing the components delivered by the SODALITE consortium at the end of the first year of the project. This comprises what has been achieved for the first SODALITE prototype embodied in Milestone 4 (MS4) of the project which was defined as:

- Laboratory prototype that is “up-and-running”. Use-Cases are defined and can be partially executed on the prototype. Public release.

The SODALITE architecture is divided into 3 main layers: Modelling layer, Infrastructure as Code layer, and Runtime layer. This document summarizes the available stable components in each of these layers, and points to instructions on how these components can be accessed and built.

This document represents the status at the end of Year 1 of the project and will be updated with future releases of the framework at the end of Year 2 (D6.6) and Year 3 (D6.7).

Glossary

Acronym	Explanation
AADM	Abstract Application Deployment Model
ALDE	Application Lifecycle Deploy Engine
AOE	Application Ops Experts
API	Application Program Interface
DSL	Domain Specific Language
IaC	Infrastructure as Code
IDE	Integrated Development Environment
KB	Knowledge Base
KPI	Key Performance Indicator
RDF	Resource Description Framework
REST	Representational State Transfer
TOSCA	Topology and Orchestration Specification for Cloud Applications

1 Introduction

The SODALITE Framework is the software system that includes all SODALITE stable components. **This deliverable presents the status and location of the software that make up the SODALITE stack, describing the components delivered by the SODALITE consortium at the end of the first year of the project.** This is mainly a software deliverable and comprises what has been achieved for the first SODALITE prototype embodied in Milestone 4 (MS4) of the project which was defined as:

- Laboratory prototype that is "up-and-running". Use-Cases are defined and can be partially executed on the prototype. Public release.

Although several components of this First Prototype already use the CI/CD pipeline to automatically build their artifacts, at this stage (M12) many of the components are still built, tested and integrated manually on the local environments and deployments. Instructions are provided individually for each component on how to compile and use the component. Some of the components still lack unit tests. For the next release of the Sodalite Framework (scheduled at M18 of the project) we intend to provide an integrated end-to-end Sodalite stack, including tests, built and deployed using CI/CD tools.

Specific build instructions, with the accompanying examples, are provided on the component's GitHub page, as we treat them as a living document. Furthermore, the M18 release, as the first release of the platform, will be accompanied with the build blueprint, allowing to set-up the complete SODALITE platform in a straightforward manner.

Instructions on how to contribute to the SODALITE stack are provided in deliverable D2.4, *Guidelines for contributors to the SODALITE framework*, where additional software quality checks are also described (e.g. CI/CD).

The SODALITE architecture is divided into 3 main layers: Modelling layer, Infrastructure as Code layer, and Runtime layer. A laboratory prototype of the SODALITE Framework is running on the SODALITE testbed, and the SODALITE Use-Cases can be partially executed on the prototype. This document summarizes the available stable components in each of the layers, and points to instructions on how they can be accessed and built.

Video demonstrations of many of these components are presented on the Project's youtube channel[1].

Additional details of various components can be found in other Year 1 deliverables.

- D2.1 *Requirements, KPIs, evaluation plan and architecture - First version*
  - This deliverable includes the definition of SODALITE requirements and KPIs, the plan for assessing the fulfillment of requirements and KPIs in the context of the SODALITE case studies, and the initial outline of the SODALITE architecture.
- D2.4 *Guidelines for contributors to the SODALITE framework*
  - This document defines the guidelines that external contributors may follow in order to submit their extensions and fixes to the main project baseline. It also contains a brief overview of the repositories whose status are reported here-in.

[1] https://www.youtube.com/watch?v=8YC11JFSWC4

- D3.1 *First version of ontologies and semantic repository*
  - This deliverable describes the first iteration of the semantic models defined by SODALITE, and describes the first version of the Semantic Reasoner module that populates the SODALITE Knowledge Base with the resource models and Abstract Application Deployment Models composed through the textual editor (Application Developer IDE).
- D4.1 *IaC Management - initial version*
  - This deliverable describes the first iteration of development of the Infrastructure as Code (IaC) layer within the SODALITE platform, and describes the deployment preparation process and performance optimisation tasks of infrastructure-as-code produced before deployment.
- D5.1 *Application deployment and dynamic runtime - Initial version*
  - This deliverable describes the first iteration of the SODALITE Runtime Layer, which is responsible for orchestrating the deployment of applications on heterogeneous infrastructures, collecting runtime monitoring information and enabling adaptation of the application to improve its performance.
- D6.2 *Initial implementation and evaluation of the SODALITE platform and use cases*
  - This document reports on the initial implementation of the fundamental components that make up the SODALITE platform, reviews the first prototypes developed for each of the project's three Demonstrating Use Cases, as well as their evaluation and validation.

Throughout the document, we are using the terms Application Ops Experts (AOE), Resource Experts (RE) and Quality Experts (QE). The following table provides a mapping between these roles and the processes defined in the ISO/IEC/IEEE standard 12207 Systems and software engineering — Software life cycle processes:

SODALITE Roles	ISO/IEC/IEEE standard 12207 processes
Application Ops Experts (AOE)	Operation processes and maintenance processes
Resource Experts (RE)	Infrastructure management and Configuration management processes
Quality Experts (QE)	Quality Management and Quality assurance processes

This document represents the status at the end of Year 1 of the project and will be updated with future releases of the framework at the end of Year 2 (D6.6) and middle of Year 3 (D6.7).

1.1 Structure of the Document

The next subsections reproduce (mainly from D2.1) an overview of the SODALITE Context and Goals followed by the SODALITE architecture and components that make up the SODALITE Framework. The following sections of the document list the components (grouped according to the architecture structure) that make up the SODALITE Platform. For each component, we provide a short description, status, and pointers to source code repository and instructions on how to build the code.

1.2 SODALITE Context and Goals

The SODALITE vision is to *support Digital Transformation of European Industry through (1) increasing design and runtime effectiveness of software-defined infrastructures, to ensure*

high-performance execution over dynamic heterogeneous execution environments; (2) increasing simplicity of modelling applications and infrastructures, to improve manageability, collaboration, and time to market.

Within this vision, SODALITE *provides application developers and infrastructure operators with tools that (a) abstract their application and infrastructure requirements to (b) enable simpler and faster development, deployment, operation, and execution of heterogeneous applications reflecting diverse circumstances over (c) heterogeneous, software-defined, high-performance, cloud infrastructures, with a particular focus on performance, quality, manageability, and reliability*.

In particular, SODALITE is focusing on supporting the entire life cycle of the so-called Infrastructure as Code (IaC). IaC means limiting the need to manually provision resources, configuring them and deploying an application by offering to DevOps teams the possibility to code such tasks into proper scripts that are then executed by proper orchestrators, thus introducing significant automation in the application life cycle.

The following innovations are envisioned:

1. Application Deployment Modeling and Infrastructure as Code Modeling - Build deployment patterns based on preexisting models.
2. Ease of Deployment, Orchestration, and Provisioning.
3. Verification and Bug Prediction - Support for a subset of smells, anti-patterns, and bugs in IaC scripts (TOSCA and Ansible).
4. Monitoring and Reconfiguration - Monitoring of metrics relevant for Cloud, HPC, and Edge computing environments and applications; Basic event-driven deployment refactoring decision making; Support for detecting and fixing performance anti-patterns.
5. Performance Optimization - Static application optimisation for Cloud, HPC and Edge; Support for modelling performance.

1.3 Overview of SODALITE architecture

We reproduce here the figures and overview of the SODALITE architecture for simple reference. Additional details of the figures can be found in the Architecture section of D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*).

The SODALITE platform is divided into three main layers. These layers are the Modelling layer, the Infrastructure as Code layer, and the Runtime layer. Figure 1 shows these layers together with their relationships defined in terms of interfaces. The Modelling layer exploits the interfaces offered by the other two layers to offer to the end users (Application Ops Experts, Resource Experts and Quality Experts) the needed information concerning the application deployment configuration and the corresponding runtime. In turn, it offers to the other layers the possibility to access the ontology and the application deployment model through the SemanticReasoningAPI. The Infrastructure as Code Layer offers to the Modelling layer the APIs for preparing the deployment, for verifying the IaC and for predicting defects. Finally, the Runtime layer offers APIs for controlling the orchestration of an application deployment and for monitoring the status of the system. In turn, this layer relies on the interfaces offered by the underlying technologies with particular reference to the ones shown in the figure.

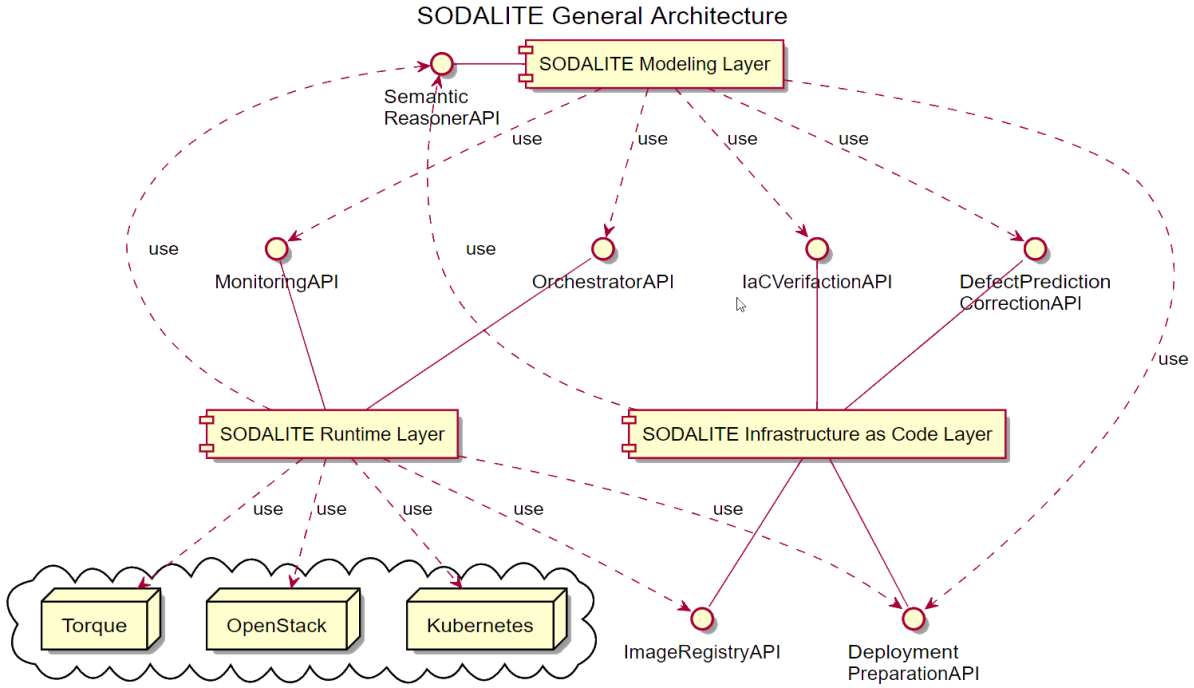

Figure 1 - SODALITE Overall Architecture.

1.3.1 Modelling Layer

Figure 2 shows the internal architecture of the SODALITE Modelling layer.

The SODALITE IDE provides complete support for the authoring lifecycle of abstract application deployment models. The Semantic Knowledge Base (KB) is SODALITE's semantic repository that hosts the models (ontologies). The Semantic Reasoner is a middleware facilitating the interaction with the KB. In particular, it provides an API to support the insertion and retrieval of knowledge to/from the KB, and the application of rule-based semantic reasoning over the data stored in the KB.

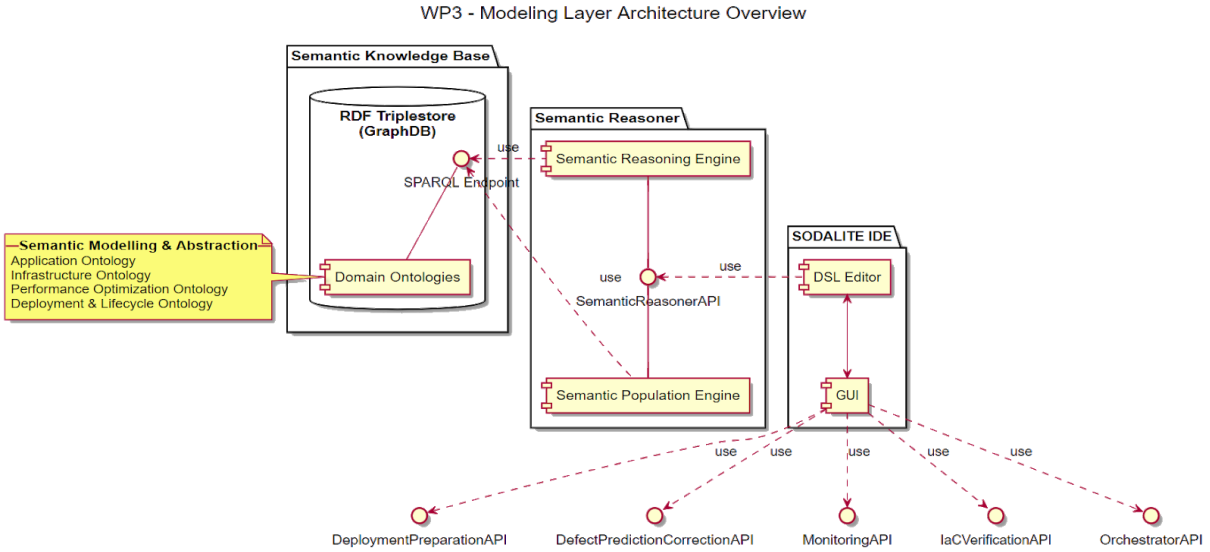

Figure 2 - Modelling Layer Architecture.

SODALITE IDE goes beyond existing approaches, by extending the modeling support to both Cloud and HPC domains, and by incorporating specific modeling assistance for the optimization of application deployments.

Beyond providing a simplified DSL for Resource Model authoring, SODALITE releases the Resource Experts from the complexity of the TOSCA YAML specification.

Additional details can be found in deliverable D3.1.

1.3.2 Infrastructure as Code (IaC) Layer

The main task of the IaC layer is to take the modelling information provided by the SODALITE IDE and produce an IaC blueprint. Deployment Preparation involves a number of operations to build an IaC blueprint. These operations are handled by sub-components depicted in Figure 3. Additional components verify correctness of the provided model, predict possible bugs in the provided model, and optimise the application for a given target execution platform. During development a part of the architecture was redesigned which was also reported in deliverable D4.1.

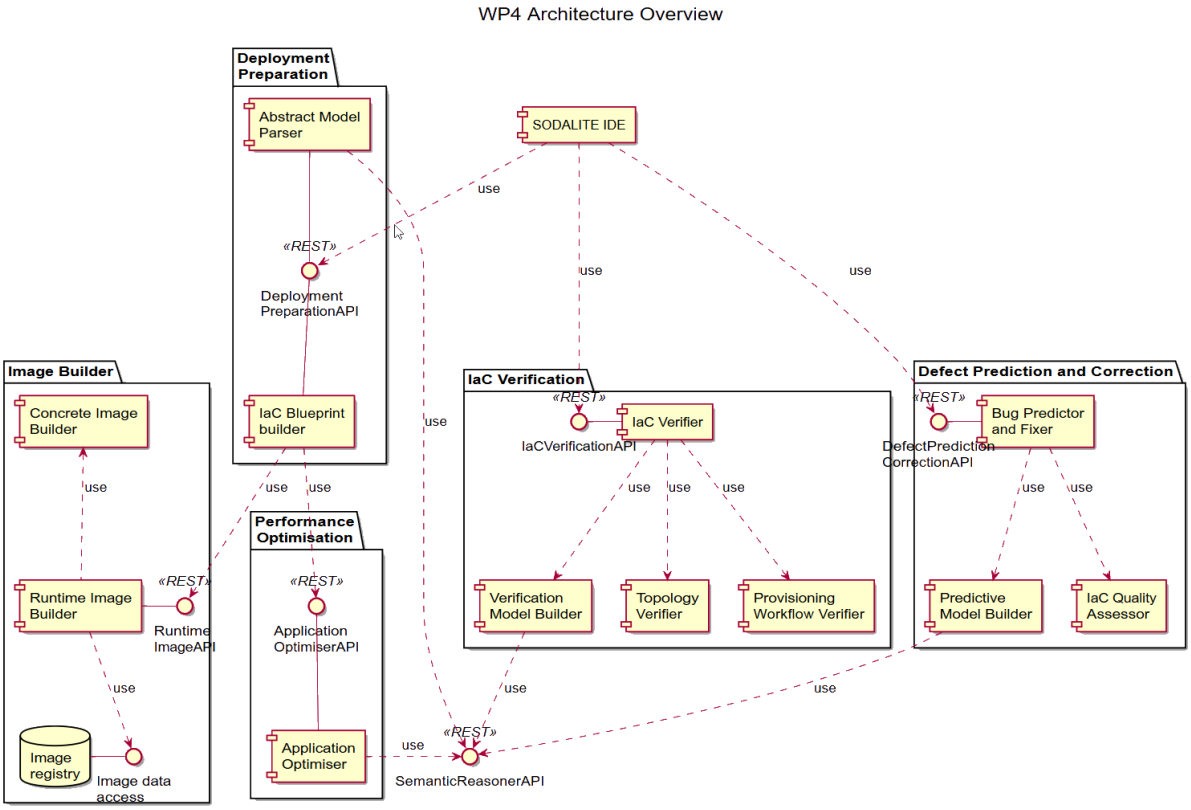

Figure 3 - Infrastructure as Code Layer Architecture.

SODALITE allows use of heterogeneous execution environments ranging from edge devices to classical clouds and HPC clusters. Our deployment preparation and infrastructure management framework focuses on the optimization of execution containers and on the verification and improvement of IaC.

The automation of application optimisation on both HPC and cloud systems requires models that can be used for performance prediction and to study how different hardware components affect

performance. SODALITE prepares and uses these models for both pre-deployment (static) performance optimization and runtime (dynamic) performance optimization.

SODALITE develops a bug and smell taxonomy for IaC as well as cloud and HPC applications based on a systematic literature review and a qualitative analysis of bug fix commit messages. We employ semantic technologies for verifying structural constraints and semantics of IaC, and also support explaining errors and their causes, recommending the resolutions, (semi) automating the correction of erroneous IaC code through model-to-model transformations.

Additional details can be found in deliverable D4.1.

1.3.3 Runtime Layer

The Runtime layer of SODALITE, shown in Figure 4, orchestrates the deployment of an application, monitors its execution and proposes changes to the application's runtime. It is composed of three main blocks: Orchestrator, Monitoring and Refactoring. The Orchestrator manages the lifecycle of an application deployed in heterogeneous infrastructures. The Monitoring component gathers metrics from the heterogeneous infrastructures. These metrics are used to determine to what extent the application is running as expected. The Deployment Refactorer refactors the deployment model of an application in response to violations in the application goals.

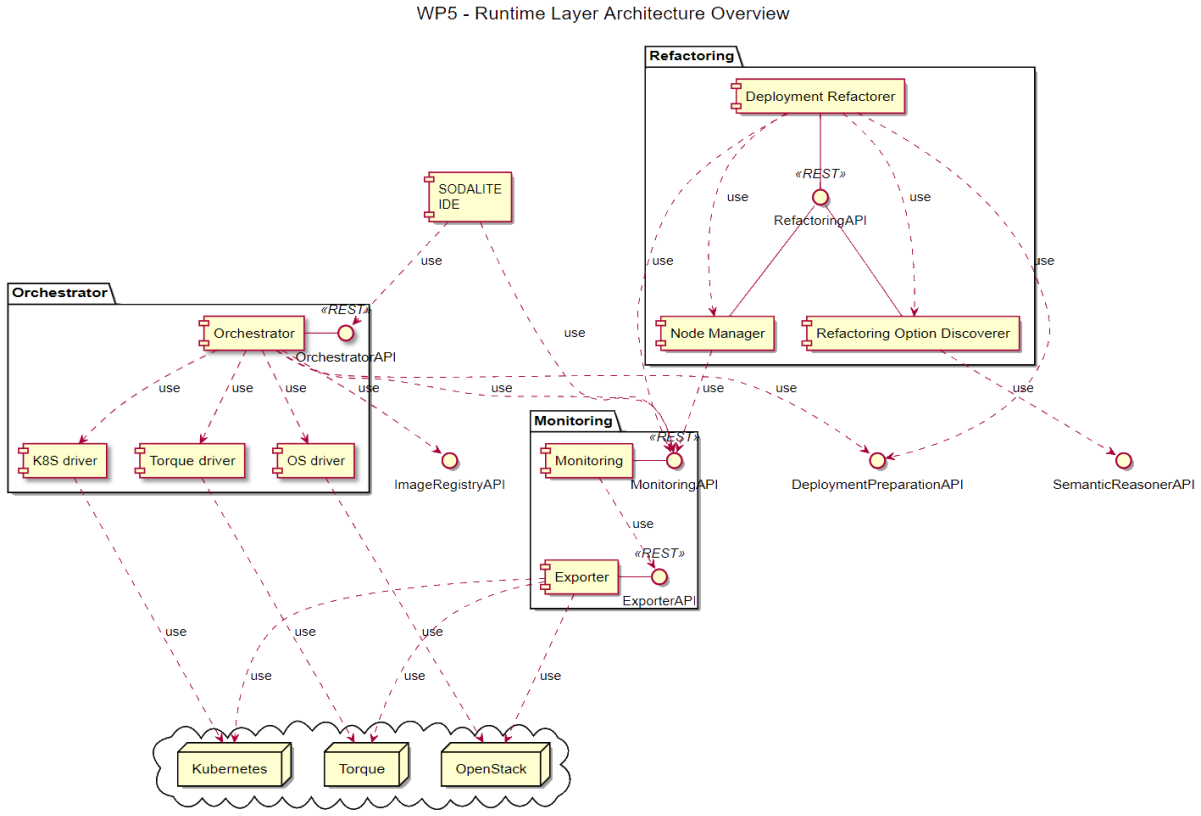

Figure 4 - Runtime Layer Architecture.

The SODALITE Orchestrator differs from other approaches that follow an intrusive architecture that require modifications to the infrastructure configuration. Our approach is to orchestrate resources via the existing resource managers and execution platforms.

We have a novel approach for the deployment optimization problem by combining rule-based, machine-learning based, and control-theory based techniques. We adopt the dynamic software product lines view, finding the optimal deployment variant between a set of allowed variants. To this end, we develop machine-learning based predictors that accurately predict performance of the whole population of the deployment variants based on a small sample of measured deployment variants. This allowed set of deployment variants is evolved at runtime by discovering and integrating new resources, nodes, and components. Moreover, we develop control-theory based planners to support fast vertical elasticity for containerized applications. We also aim to detect performance anti-patterns, being able to correct them in application deployments at runtime.

Additional details can be found in deliverable D5.1.

2 Modelling Layer

This section lists the components of the Modelling layer, describing each component's function, implementation status, and code repository.

2.1 Semantic Knowledge Base

2.1.1 Description of component

The Semantic Knowledge Base (KB) is the semantic database management system of SODALITE that enables storing, querying and managing structured data. It follows the semantic data schema paradigm, called ontology, which is stored and managed independently from the data (see D3.1, *First version of ontologies and semantic repository*, for more details).

2.1.2 Status of implementation

KB uses an existing RDF triple store (GraphDB[2]) to persist and index the first version of the ontologies developed in T3.1 (*Application Semantic Modelling*) and T3.2 (*Infrastructure Semantic Modelling*) regarding the modelling of applications and resources. The current version of the ontologies include modules that provide:

- The formal schema, i.e. classes and properties that can be used to capture application and resource models (TOSCA[3] ontology).
- The ontology pattern that should be followed (SODALITE meta-model) in order to define modular and reusable knowledge graphs.

2.1.3 Location of repository and how to build the code

This component is in the following Github repository. The readme file of the repository includes the build instructions.

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

2.2 Semantic Reasoner

2.2.1 Description of component

The population of the KB, i.e. the instantiation of the respective ontology patterns to capture resources and applications (AADM), is performed by the Semantic Reasoner, which encapsulates the necessary logic to translate the DSL composed in the IDE by the users to the conceptual model of SODALITE. In addition, the Semantic Reasoner provides all the necessary interfaces to retrieve data from the KB, as well as to expose reasoning functionality developed in the IaC layer with respect to searching and validation services (see D3.1, *First version of ontologies and semantic repository*, for more details).

2.2.2 Status of implementation

A number of REST API endpoints have been developed in order to assist users in defining models in the IDE. It should be noted that this REST API exposes functionality that has been mainly developed in T4.4 (*Analytics and Semantic Decision Support*) relevant to searching and validation. More details on the backend implementation of the REST API are provided in D4.1 (*IaC Management - initial version*). The component also supports the population of the KB with node templates.

[2] http://graphdb.ontotext.com/
[3] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

2.2.3 Location of repository and how to build the code

This component is in the following Github repository. The readme file of the repository includes the build instructions.

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

2.3 SODALITE IDE

2.3.1 Description of component

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

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

2.3.2 Status of implementation

The development status of the IDE is as follows:

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

DSL editor: current version implements the following features:

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

2.3.3 Location of repository and how to build the code

This component is in the following Github repository: https://github.com/SODALITE-EU/ide.

The readme file of the repository includes instructions to build and install this IDE within an Eclipse instance.

3 IaC Management Components

This section lists the components of the IaC Management layer, describing each component's function, implementation status, and code repository.

3.1 Abstract Model Parser

3.1.1 Description of component

This component parses the abstract application deployment model (AADM) retrieved from the Semantic Reasoner and prepares an internal representation of topology tree based on the model retrieved. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

3.1.2 Status of implementation

The preliminary version of the component was implemented.

3.1.3 Location of repository and how to build the code

The initial version of the component is implemented inside the following Github repository: https://github.com/SODALITE-EU/iac-blueprint-builder.

3.2 IaC Blueprint Builder

3.2.1 Description of component

This component generates a TOSCA/Ansible[4] blueprint based on the inner abstract tree provided by the Abstract Model Parser. After the blueprint is generated it calls the orchestrator REST API endpoint to register the blueprint thus making it ready for deployment.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

3.2.2 Status of implementation

The preliminary version of the component was implemented.

3.2.3 Location of repository and how to build the code

The initial version of the component is implemented inside the following Github repository: https://github.com/SODALITE-EU/iac-blueprint-builder.

3.3 Runtime Image Builder

3.3.1 Description of component

This component encapsulates the process of building a SODALITE runtime image based on the data provided by the user (Application Ops Expert). The image is then pushed to the Image Registry to be later pulled and deployed by the orchestrator at deploy time.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

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

3.3.2 Status of implementation

The initial version of the component that comprises the IaC TOSCA/Ansible definitions of sources and targets and a REST API endpoint for invoking the Image Builder was implemented.

3.3.3 Location of repository and how to build the code

The initial version of the component is implemented inside this Github repository: https://github.com/SODALITE-EU/image-builder.

3.4 Concrete Image Builder

3.4.1 Description of component

This component is a specific implementation of a runtime image (docker - for instance). The built image is then pushed to the Image Registry to be later pulled and deployed by the orchestrator at deploy time.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

3.4.2 Status of implementation

The initial version of the component that comprises the IaC TOSCA/Ansible definitions of sources and targets and a REST API endpoint for invoking the Image Builder was implemented for building docker images and pushing them to the SODALITE Image Registry. The plan is to setup a singularity registry and integrate a singularity image builder component.

3.4.3 Location of repository and how to build the code

The initial version of the component is implemented inside this Github repository: https://github.com/SODALITE-EU/image-builder.

3.5 Image Registry

3.5.1 Description of component

This component implements a private SODALITE image registry. The images built within SODALITE are stored in the registry and later on pulled and deployed by the orchestrator at deploy time.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

3.5.2 Status of implementation

The initial version of the private docker registry was set up, configured and secured with TLS using IaC TOSCA/Ansible blueprint, which was deployed using xOpera. A similar implementation is planned for the singularity registry.

3.5.3 Location of repository and how to build the code

The initial version of the component is implemented as IaC inside this Github repository: https://github.com/SODALITE-EU/iac-management. This repository also comprises examples and deployment blueprints for SODALITE use cases deployed through SODALITE platform.

3.6 Application Optimiser

3.6.1 Description of component

This component tries to build a performance optimized runtime given the target platform and configuration and predefined optimisation options.

See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

3.6.2 Status of implementation

The initial version of performance model training is released for building a performance model.

3.6.3 Location of repository and how to build the code

The initial version of the component is implemented and described inside this Github repository: https://github.com/SODALITE-EU/application-optimisation.

3.7 IaC Verifier

3.7.1 Description of component

This component coordinates the verification of the application deployment topology and the provisioning workflow described in the IaC artifacts. It provides a uniformed REST API for all types of verifications. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version)* and D4.1 (*IaC Management - initial version*) for more information.

3.7.2 Status of implementation

The preliminary version of REST API was implemented.

3.7.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository. The readme file of the project includes the build instructions.

https://github.com/SODALITE-EU/verification

3.8 Verification Model Builder

3.8.1 Description of component

This component builds the formal models required for the verification of a given set of IaC artifacts (TOSCA and Ansible). See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*)and D4.1 (*IaC Management - initial version*) for more information.

3.8.2 Status of implementation

The current version of this component can build semantic models required for verifying the deployment topology described in a TOSCA file.

3.8.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository. The readme file of the project includes the build instructions.

https://github.com/SODALITE-EU/verification

3.9 Topology Verifier

3.9.1 Description of component

This component verifies the constraints over the structures of the TOSCA blueprints and Ansible scripts. This includes assessing the substitutability of the nodes in the deployment topology (TOSCA substitution mappings). It also provides the infrastructure for semantic search and reuse of content from the KB. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

3.9.2 Status of implementation

Preliminary support for verification of TOSCA topology has been implemented.

3.9.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository. The readme file of the project includes the build instructions.
https://github.com/SODALITE-EU/verification

3.10 Provisioning Workflow Verifier

3.10.1 Description of component

This component verifies the constraints over the deployment workflow of the application described in the Ansible (IaC) scripts. As of now, we employ Petri net as the formal modelling language, which is widely used for verifying workflows and business processes. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

3.10.2 Status of implementation

The mappings from Ansible to Petri net have been defined to support the verification of the workflow described in the Ansible playbooks and roles. The implementation of the automated mapping tool is work-in-progress.

3.10.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository. The readme file of the project includes the build instructions.
https://github.com/SODALITE-EU/verification

3.11 Bug Predictor and Fixer

3.11.1 Description of component

This component predicts the different types of bugs in IaC (including TOSCA). This includes a bug taxonomy covering a wide-range of bugs and software smells for Iac such as code smells, design smells, security smells, and linguistic anti-patterns. Then, the tools support for detecting and correcting those bugs are also developed. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

3.11.2 Status of implementation

Based on a multivocal literature review, a bug taxonomy for Ansible has been created. It includes code smells, design smells, and security smells. For TOSCA, the taxonomy includes security smells. To detect Ansible code and security smells, we have extended the Ansible-Lint tool. To detect Ansible design smells, we have implemented the metric-based heuristics. To detect security smells

in TOSCA, we have developed a semantic-rule based approach by utilizing the semantic modelling and reasoning support developed in the SODALITE Modelling layer.

3.11.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository: https://github.com/SODALITE-EU/defect-prediction. The readme file of the project includes the build instructions.

3.12 Predictive Model Builder

3.12.1 Description of component

This component builds the models that can find the smells in TOSCA and Ansible artifacts. The model can be a rule-based model, a heuristics based model, and a data-driven (machine learning) model. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

3.12.2 Status of implementation

A semantic rule based model was built to detect the security smells in TOSCA. An informal rule based model (Ansible-Lint Rules) was built to detect the code and security smells in Ansible. A set of metrics-based heuristics were built to detect the design smells in Ansible.

3.12.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository: https://github.com/SODALITE-EU/defect-prediction. The readme file of the project includes the build instructions.

3.13 IaC Quality Assessor

3.13.1 Description of component

This component can calculate different software quality metrics for TOSCA and Ansible artifacts. These metrics are used by the heuristics that predict the design smells in TOSCA and Ansible. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D4.1 (*IaC Management - initial version*) for more information.

3.13.2 Status of implementation

The IaC metrics required for detecting design smells in Ansible have been developed.

3.13.3 Location of repository and how to build the code

This component is a sub-project under the following Github repository: https://github.com/SODALITE-EU/defect-prediction. The readme file of the repository includes the build instructions.

4 Runtime Layer Components

This section lists the components of the Runtime layer, describing each component's function, implementation status, and code repository.

4.1 Orchestrator -> xOpera

4.1.1 Description of component

SODALITE uses xOpera as a base orchestrator. xOpera is a pre-existing lightweight TOSCA 1.2/1.3 simple yaml compliant orchestrator which uses Ansible as the actuation scripting language for implementation of application lifecycle management including agentless deployments on different platforms.

4.1.2 Status of implementation

xOpera aims to be a simple and lightweight TOSCA 1.3 compliant orchestrator. A dockerized version of the REST API including xOpera was deployed to the testbed by a prepared IaC deployment blueprint. The development of xOpera is ongoing as it aims to implement TOSCA yaml 1.3 standard.

4.1.3 Location of repository and how to build the code

This component is referenced as a github submodule in the following Github repository: https://github.com/SODALITE-EU/orchestrator.

4.2 xOpera REST API

4.2.1 Description of component

xOpera REST API implements an endpoint interface for xOpera deployments with blueprint persistence, session management, status of deployment, history of deployment, documented with swagger. The REST API is dockerized and enables transportability and easier integration with other SODALITE components.

4.2.2 Status of implementation

xOpera REST API is developed inside the SODALITE project and supports deployments to SODALITE testbed platforms (HPC/Openstack). Currently the REST API supports blueprint registration and persistence, session management, status of deployment, history of deployment, documented with swagger. Extensions to xOpera and the REST API are planned to support the targeted functionalities needed in SODALITE with special focus on security.

4.2.3 Location of repository and how to build the code

This component is implemented and developed in the following Github repository: https://github.com/SODALITE-EU/xopera-rest-api. The readme files provide information on setting up a working REST API environment also provided as a dockerized component.

4.3 Deployment Refactorer

4.3.1 Description of component

The refactoring of the deployment model of a running application is performed in response to the potential violations of the application goals, which is determined using the monitoring data. The refactoring can find and enact a new deployment model for the application that can resolve the detected goal violations. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First*

version) and D5.1 (*Application deployment and dynamic runtime - Initial version*) for more information.

4.3.2 Status of implementation

Rule-based refactoring decision making has been implemented to support the first version of the Vehicle IoT SODALITE use case. The modifications to one of widely used cloud benchmark applications (namely RUBiS web application[5]) were introduced to support the variations in resources and deployment topology. A preliminary version of a machine-learning based framework for predicting the performance of a deployment model variant was developed.

4.3.3 Location of repository and how to build the code

This component is in the following Github repository. The readme file of the repository includes the build instructions.

https://github.com/SODALITE-EU/refactoring-ml

4.4 Node Manager

4.4.1 Description of component

Node manager can perform vertical scalability (i.e., change the resource allocation of running components) using control-theory. Its goal is to fulfill requirements on the response time (e.g., response time < 0.5s) by reconfiguring the CPU and GPU allocation of running containers.

4.4.2 Status of implementation

Node Manager is currently implemented as a Kubernetes pod that uses Docker-out-of-Docker to re-configure other containers/pods of the same machine. It targets TensorFlow applications that can exploit both GPUs and CPUs. NodeManager was tested with five benchmark applications: Skyline Extraction from SnowUC, GoogLeNet, ResNet, AlexNet and VGG-16.

4.4.3 Location of repository and how to build the code

This component is in the following Github repository. The readme file in the repository contains architecture details and run instructions.

https://github.com/SODALITE-EU/refactoring-ct

4.5 Refactoring Option Discoverer

4.5.1 Description of component

Refactoring Option Discoverer can discover new refactoring options as well as the changes to existing refactoring options. The search criteria can be user-defined constraints, infrastructure design patterns and anti-patterns. See D2.1 (*Requirements, KPIs, evaluation plan and architecture - First version*) and D5.1 (*Application deployment and dynamic runtime - Initial version*) for more information. Refactoring Option Discoverer utilizes the semantic matchmaking capabilities developed in the Modelling layer.

4.5.2 Status of implementation

Refactoring Option Discoverer supports the discovery of the refactoring options that satisfy a given set of constraints (e.g., an open stack VM with X number of CPUs and deployed in a data center in Germany).

[5] https://github.com/uillianluiz/RUBiS

4.5.3 Location of repository and how to build the code

This component is in the following Github repository. The readme file of the repository includes the build instructions.

https://github.com/SODALITE-EU/refactoring-option-discoverer

4.6 ALDE

4.6.1 Description of component

ALDE[7] (Application Lifecycle Deployment Engine) is a REST API responsible for managing the workload scheduling and execution of applications, primarily intended for HPC environments.

4.6.2 Status of implementation

It is an outcome of the TANGO[6] project, supporting Slurm and creation and deployment of Singularity containers.

4.6.3 Location of repository and how to build the code

This component is in the following Github repository. The code will be moved to a SODALITE repository when new commits are pushed. The readme file of the repository includes the build instructions.

https://github.com/TANGO-Project/alde

4.7 Prometheus

4.7.1 Description of component

Prometheus[7] is an open-source system monitoring and alerting toolkit. Agents on the various machines in a cluster periodically collect system information, which is stored in a database. The data can then be filtered and analyzed according to the needs of the user.

4.7.2 Status of implementation

Prometheus is a standalone open source project and maintained independently of any company.

4.7.3 Location of repository and how to build the code

Detailed information on the tool capabilities is available on the Prometheus web site https://prometheus.io/. Source code may be downloaded from https://prometheus.io/download/. Instructions to configure and run Prometheus are in the following location.

https://prometheus.io/docs/prometheus/latest/getting_started/.

4.8 Node exporter

4.8.1 Description of component

Node exporter[8] is a piece of software that exposes hardware metrics to be gathered by Prometheus. It gathers metrics related to CPU (such as usage, frequency, etc), memory, I/O and network, among others, providing a good insight on resource utilization.

[6] http://www.tango-project.eu/content/application-lifecycle-deployment-engine-alde
[7] https://prometheus.io
[8] https://github.com/prometheus/node_exporter

4.8.2 Status of implementation

Similar to Prometheus, Node Exporter is a standalone open source project and maintained independently of any company.

4.8.3 Location of repository and how to build the code

A list of the metrics that Node Exporter exposes can be found in its repository together with its requirements and installation process. (https://github.com/prometheus/node_exporter).

4.9 IPMI exporter

4.9.1 Description of component

IPMI exporter[9] is another Prometheus exporter created to expose the power measurements of the physical nodes given by the output of one command. It can also be adapted to expose any metric that is outputted by any CLI command.

4.9.2 Status of implementation

IPMI exporter is an open source project and can accept modifications performed or suggested by the community. In its current state, it only exposes one metric, but this can be easily expanded if more are required.

4.9.3 Location of repository and how to build the code

The repository (https://github.com/SODALITE-EU/ipmi-exporter) is hosted inside of the SODALITE Github page and it contains the build procedure and a description of its functioning.

4.10.1 Skydive

4.10.1 Description of component

Skydive[10] is a real-time network topology and protocol analyzer that provides detailed network topology and performance information. Skydive agents collect topology information and flows and forward them to a central agent for further analysis. A more detailed summary is in deliverable D5.1 (*Application deployment and dynamic runtime - initial version*) and additional details are available on the skydive site (http://skydive.network/documentation/).

4.10.2 Status of implementation

Skydive is a stable Open Source project with community support. New features are added on an on-going basis. IBM team members are regular contributors to the Skydive community. Over the past year, IBM team members implemented a generic flow exporter for Skydive to expose the Skydive flow data in a convenient form to various consumers. The skydive-prometheus connector is built on top of the flow exporter and is still work-in-progress.

4.10.3 Location of repository and how to build the code

The Skydive code repository is at https://github.com/skydive-project/skydive. Details on how to deploy Skydive are at http://skydive.network/documentation/getting-started. Pre-compiled versions of Skydive are availabe at https://github.com/skydive-project/skydive/releases. Details on how to compile the code from source are at http://skydive.network/documentation/build. The flow

[9] https://github.com/SODALITE-EU/ipmi-exporter
[10] http://skydive.network

exporter code repository is at https://github.com/skydive-project/skydive-flow-exporter, including instructions on how to build the code.

5 Conclusion

This document summarizes the SODALITE Framework at the end of the first year of the project. This comprises what has been achieved for the first SODALITE prototype embodied in Milestone 4 (MS4) of the project. This includes components from all three layers of the Sodalite architecture: the Modelling layer, the Infrastructure as Code layer, and the Runtime layer. Our first prototype is up-and-running on the testbed. Use-Cases have been defined and can be partially executed on the prototype. Additional details can be found in D6.2 (*Initial implementation and evaluation of the SODALITE platform and use cases*). Instructions are provided individually for each component on how to compile and use the component. For the next release of the Sodalite Framework (scheduled at M18 of the project) we intend to provide an integrated end-to-end Sodalite stack, including tests, built and deployed using CI/CD tools.

Video demonstrations of many of these components are presented on the Project's youtube channel.

The next major milestones are:

- MS5: Use cases can all be executed on the prototype. (M18)
- MS6: First advanced features, more integrated prototype running. Use-Cases are clearly improved. Second public release of the complete stack. (M24)

Future updates of this document are D6.6 *SODALITE framework - Second version*, scheduled for M24, and D6.7 *SODALITE framework - Final version*, scheduled for M30 of the project.