SOftware Defined AppLication Infrastructures managemenT and Engineering

D2.2

Requirements, KPIs, evaluation plan and architecture - Intermediate version

IBM and POLIMI

31/01/2021

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

Deliverable data			
Deliverable	Requirements, KPIs, evaluation plan and architecture - Intermediate version		
Authors	Luciano Baresi (POLIMI), Elisabetta Di Nitto (POLIMI), Alexander Maslennikov (XLAB), Dragan Radolović (XLAB), Alfio Lazzaro (HPE), Jesús Gorroñogoitia (Atos)		
Reviewers	Nejc Bat (XLAB) Yosef Moatti (IBM)		
Dissemination level	Public		
History of changes	Name	Change	Date
	v1	first release ready for internal review	21/01/2021
	v2	final release	28/01/2021

© Copyright Beneficiaries of the SODALITE Project

Executive Summary

This deliverable is the continuation of deliverable D2.1 and provides the consolidated evolution of requirements, KPIs, evaluation plan and architecture over the second year. More specifically, it presents the current status of the requirements identified in the original document, along with the new use case and new requirements highlighted in the second year. It provides a new description of the architecture of the SODALITE environment, where we identify the major changes we applied in the second phase of the project and also the new elements that have been released since the first milestone. The architecture described here complies with the release of the SODALITE environment at month 24, that is, Milestone MS6. As for KPIs, we now provide a detailed, ameliorated description of each KPI, its scope, and the evaluation workflow we have put in place.

© Copyright Beneficiaries of the SODALITE Project

Glossary

This section provides a reference for the main terms used in this document. Most of the terms are defined the first time they are used in the document, but their definition is also reported here for the sake of simplicity and speed. Reported terms are classified under seven main categories.

Acronyms

AADM	Abstract Application Deployment Model
AAI	Authentication and Authorization Infrastructure
ADM	Application Deployment Model
AM	Ansible Model
AOE	Application Ops Expert
CPU	Central Processing Unit
DSL	Domain Specific Language
GPU	Graphic Processing Unit
HPC	High Performance Computing
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAM	Identity and Access Management
JWT	JSON Web Token
KB	Semantic Knowledge Base
KPI	Key Performance Indicator
LRE	Lightweight Runtime Environment
MOM	Message oriented middleware
OASIS	Organization for the Advancement of Structured Information Standards
OM	Optimization Model
OWL	Web Ontology Language
PBS	Portable Batch System
QE	Quality Expert
RDF	Resource Description Framework
RE	Resource Expert

© Copyright Beneficiaries of the SODALITE Project

RM	Resource Model
SD	SODALITE Design-time
SR	SODALITE Runtime
Torque	Terascale Open-source Resource and QUEue Manager
TOSCA	Topology and Orchestration Specification for Cloud Applications

General terms

Adaptation plan	An ordered set of actions that modify the current deployment of a system.
Anti-pattern	A common design solution/decision that generates known negative consequences onto the design.
Blueprint	A plan or set of proposals to carry out some work. An IT blueprint is an artifact created to guide priorities, projects, budgets, staffing and other IT strategy-related initiatives. As for IaC, a blueprint is the scripting code that enables resource provisioning, configuration, and application deployment.
Code smell	Any characteristic in the code that possibly indicates a potential defect/bug.
Design pattern	Recurring solution that carries positive consequences onto the design.
Design smell	Any element in the design that indicates violation of fundamental design principles and negatively affects design quality.
Domain Specific Language	A design language that is specific to a particular domain.
Infrastructure as Code	Code that does not define the application logic but, instead, defines how a computational infrastructure is to be provisioned and configured and the way an application is to be deployed on top of it.
IaC artifacts	These are the documentation and models associated with Infrastructure as Code, as well as the code itself.
Infrastructure as a Service	A specific service model that corresponds to offering virtualized hardware, that is, virtual machines and similar abstractions.
Lightweight application base image	A container image (e.g., Docker or Singularity image).
Models@runtime	Indicates maintaining the models of a system at runtime to reason on the system.

© Copyright Beneficiaries of the SODALITE Project

Over-provisioning	The allocation of more computing resources (e.g., virtual machines and CPUs) than strictly necessary.
Playbook	Ansible recipe (or script) for executing a series of steps.
Use case	A possible case of usage of a certain piece of software. SODALITE distinguishes between UML use cases, those reported in this document, and Demonstrating use cases, that is, the specific application we exploit to demonstrate the SODALITE environment. These last ones are also called SODALITE case studies.

SODALITE human actors

Application Ops Expert (AOE)	The actor in charge of operating the application and, as such, of all the aspects that refer to the deployment, execution, optimization and monitoring of the application.
Quality Expert (QE)	The actor in charge of the quality of service both provided by the execution infrastructure and required by the executing application.
Resource Expert (RE)	The actor in charge of dealing with the different resources required to deploy and execute the application.

Resources managed by SODALITE

Application component	An executable the application of interest is partitioned in.
Container Engine	An engine for running lightweight containers. It enables operating-system-level virtualization and the existence of multiple isolated container instances.
Edge/Fog computing	A distributed computing paradigm that brings computation and data storage closer to the location where they are needed, to improve response times and save bandwidth.
Execution platform	Provides the means to execute the different application components; e.g., HPC, GPU, Openstack Cloud, etc.
Lightweight Runtime Environment	A "simple" execution environment provided by operating systems or by virtualization technologies.
Message oriented middleware	Software infrastructure that supports sending and receiving messages among distributed elements.
Middleware framework	The underlying glue that helps both storing the different data and artifacts and making the different elements communicate.
Monitoring agent	Software entity that collects usage and performance statistics about system resources.

© Copyright Beneficiaries of the SODALITE Project

Resource	Any computing artifact needed to deploy and run an application.
Serverless computing	A cloud-computing execution model in which the user submits only the tasks to be executed to the cloud provider, which manages the computing infrastructure transparently.

Specific targeted technologies

Docker	An open platform for developing, shipping, and running applications. Docker provides the ability to package and run an application in a loosely isolated environment called a container.
Istio	A Service Mesh on top of a cluster manager such as Kubernetes.
Kompose	Kompose is a conversion tool for Docker Compose to container orchestrators such as Kubernetes.
Kubernetes	An open-source system for automating deployment, scaling, and management of containerized applications.
OpenStack	An open source cloud operating system.
OpenWhisk	A popular and highly scalable serverless computing / cloud functions platform that allows for functional logic to be written and triggered in response to events or directly via a REST API.
Portable Batch System	A job scheduler that is designed to manage the distribution of batch jobs and interactive sessions across the available nodes in the HPC cluster.
Singularity	A container solution like Docker that is created specifically for scientific applications and workflows in an HPC environment.
Skydive	A software tool that produces network monitoring metrics.
Slurm	An open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters.
Terascale Open-source Resource and QUEue Manager (Torque)	A distributed resource manager that provides the functionality of PBS but also extends it to provide scalability, fault tolerance, usability and functionality.

SODALITE elements

Abstract Application Tuple	An Abstract Application tuple comprises an abstract description of the application, its infrastructure, and its non-functional requirements.

© Copyright Beneficiaries of the SODALITE Project

Application Deployment Model/Abstract Application Deployment Model	An abstract model defined through the use of SODALITE DSL with concrete definitions for constraints, parameters, functional and non-functional requirements and goals, thus defining an instance of the DSL model.
Infrastructure Abstract Pattern	A defined set of infrastructure resource types, interlinked with known relationship types (dependencies, compatibility, etc), aimed at supporting the recommendation generating mechanism of the Semantic Reasoner.
Semantic Knowledge Base	All modelling artefacts made available to the SODALITE users.
SODALITE Design-time	All SODALITE components made available to the user to support the design and development of Infrastructure as Code (IaC).
SODALITE DSL	The modelling language offered to the SODALITE users to support design and development of IaC.
SODALITE Runtime	All SODALITE components supporting the execution of applications on top of heterogeneous resources.
Taxonomy of Infrastructure Bugs/Defects and Resolutions	A classification of the common bugs and their resolutions for infrastructure designs and IaC code specifications.

Interchange languages

OWL2	An ontology language for the Semantic Web with formally defined meaning. OWL2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents.
TOSCA	An OASIS standard that defines the interoperable description of services and applications hosted on the cloud and elsewhere, thereby enabling portability and automated management across cloud providers regardless of underlying platform or infrastructure; thus, expanding customer choice, improving reliability, and reducing cost and time-to-value.

© Copyright Beneficiaries of the SODALITE Project

Table of contents

Executive Summary 2

Glossary 3

Table of contents 8

List of Figures 11

1. Introduction 12

2. Year-two innovation 12

2.1 Vision 12

2.2 Innovations introduced in the second project year 13

3. Updates on requirement elicitation 15

3.1 Initial requirements 16

3.2 New UC 24

UC17: Platform Resource Discovery 24

Associated requirements 25

3.3 New requirements and their impact on UCs 26

3.4 Workflow 31

3.4.1 The Resource Expert Workflow 31

3.4.2 The Application Ops Expert Workflows 32

3.4.3 The Quality Expert Workflow 33

3.4.4 The System Administrator Workflow 34

4. Architecture 34

4.1 General architecture changes 34

4.2. Modelling layer 35

4.2.1 Component descriptions 36

4.2.1.1 SODALITE IDE 36

4.2.1.2 Semantic Reasoner (Knowledge Base Service - KBS) 37

4.2.1.3 Semantic Knowledge Base (KB) 38

4.2.2 Use Case Sequence diagrams 38

4.2.2.1 UC13: Model Resources 39

4.2.2.2 UC1: Define Application Deployment Model 40

4.2.2.3 UC2: Select Resources 41

4.2.2.4 UC12: Map Resources and Optimisations 42

4.2.2.5 UC14: Estimate Quality Characteristics of Applications and Workload 43

4.3 Infrastructure as Code Layer 43

4.3.1 Component Descriptions 44

4.3.1.1 Abstract Model Parser 44

4.3.1.2 IaC Blueprint Builder 45

4.3.1.3 IaC Model Repository 45

© Copyright Beneficiaries of the SODALITE Project

4.3.1.4 Runtime Image Builder 46
4.3.1.5 Concrete Image Builder 46
4.3.1.6 Application Optimiser - MODAK 46
4.3.1.7 IaC Verifier 47
4.3.1.8 Verification Model Builder 47
4.3.1.9 Topology Verifier 47
4.3.1.10 Provisioning Workflow Verifier 48
4.3.1.11 Bug Predictor and Fixer 48
4.3.1.12 Predictive Model Builder 48
4.3.1.13 IaC Quality Assessor 49
4.3.1.14 Platform Discovery Service 49
4.3.2 Use Case Sequence diagrams 49
4.3.2.1 UC3: Generate IaC 49
4.3.2.2 UC4: Verify IaC 50
4.3.2.3 UC5: Predict and Correct Bugs 51
4.2.2.4 UC11: Define IaC Bugs Taxonomy 52
4.3.2.5 UC15: Statically Optimise Application and Deployment 53
4.3.2.6 UC16: Build Runtime images 54
4.3.2.6 UC17: Platform Discovery Service 55
4.3 Runtime Layer 56
4.3.1 Component Descriptions 57
4.3.1.1 xOpera REST API 57
4.3.1.2 Monitoring + Exporters 57
4.3.1.2 Monitoring Dashboard 58
4.3.1.2 Alert Manager 58
4.3.1.3 Deployment Refactorer 58
4.3.1.4 Node Manager 59
4.3.1.5 Refactoring Option Discoverer 59
4.3.2 Sequence Diagrams 60
4.3.2.1 UC6: Execute Provisioning, Deployment and Configuration 60
4.3.2.2 UC7: Start Application 62
4.3.2.3 UC8: Monitor Runtime 63
4.3.2.4 UC9: Identify Refactoring Options 64
4.3.2.5 UC10: Execute Partial Redeployment 65
4.4 Security Pillar 66
4.4.1 Security Pillar Toolkit 67
5. KPIs and evaluation plan 68
5.1 Technical KPIs 69
5.2 Quality metrics associated with the SODALITE development process 69
5.3 Evaluation plan 70
KPI 1.1 70

© Copyright Beneficiaries of the SODALITE Project

KPI 1.2 70
KPI 1.3 71
KPI 2.1 72
KPI 2.2 72
KPI 3.1 74
KPI 3.2 75
KPI 4.1 76
KPI 5.1 77
KPI 5.2 77
5.4 Controlled experiments 78
Experiment with students and other external stakeholders 78
Description of the Application used for the experiments 78
Exercise A - Development of a deployment blueprint in TOSCA 78
Exercise B - Development of an AADM using the SODALITE IDE 79
Experiment with case study owners 79
Experiment with TOSCA experts 79
6. Conclusions and Future Work 79
7. References 80

© Copyright Beneficiaries of the SODALITE Project

List of figures

List of Images

- Figure 1 - Resource Expert Workflow.
- Figure 2 - Design Time Workflow Executed by Application Ops Experts.
- Figure 3 - Runtime Workflow Executed by Application Ops Experts.
- Figure 4 - Workflow executed by the Quality Expert.
- Figure 5 - Updated SODALITE general architecture.
- Figure 6 - Updated SODALITE Modelling Layer Architecture.
- Figure 7 - Sequence diagram for UC13.
- Figure 8 - Sequence diagram for UC1.
- Figure 9 - Sequence diagram for UC2.
- Figure 10 - Sequence diagram for UC12.
- Figure 11 - Updated SODALITE IaC Layer Architecture.
- Figure 12 - Sequence diagram for UC3 Generate IaC.
- Figure 13 - Sequence diagram for UC4 Verify IaC.
- Figure 14 - Sequence diagram for UC5 Predict and Correct Bugs.
- Figure 15 - Sequence diagram for UC11 Define IaC Bugs Taxonomy.
- Figure 16 - Sequence diagram for UC15.
- Figure 17 - Sequence diagram for UC16.
- Figure 18 - Sequence diagram for UC17 Platform Discovery Service.
- Figure 19 - Updated SODALITE Runtime Layer Architecture.
- Figure 20 - Sequence diagram for UC6.
- Figure 21 - Sequence diagram for UC7.
- Figure 22 - Sequence diagram for UC8.
- Figure 23 - Sequence diagram for UC9.
- Figure 24 - Sequence diagram for UC10.
- Figure 25 - Sample IAM authentication workflow.
- Figure 26 - Structure of ML exemplar application.

© Copyright Beneficiaries of the SODALITE Project

1. Introduction

This document is the continuation of D2.1, where the consortium identified the first set of requirements behind the SODALITE environment, designed a first architecture, revised the KPIs stated in the description of work and provided some first hints on how to evaluate them. This document describes an updated and ameliorated intermediary version, which aims to collect, integrate, and present all the work done over the second year of the project and in preparation for the milestone at month 24 (MS6). The work presented here is not simply incremental, since it comes one year after the initial document, but it also contains new and original contributions developed by the consortium in this second leg of the project.

Even if the document complements D2.1, it is self-contained and supplies the reader with updated descriptions of the different aspects. First of all, we frame the document by summarizing the innovative elements conceived and developed over the second year. Then we move to the work done on requirements and we start by recalling all the requirements identified initially and setting their current fulfilment level. Diverse requirements proposed in D2.1 are now fully implemented; some requirements are now outdated, some will be addressed in the next, final iteration, and others have been modified due to the new/current needs. The final status of the requirements will be provided at M30 in deliverable D2.3.

The section on the architecture of the SODALITE environment is aimed to both refine the initial architecture and frame the novel elements. We also want to present a complete map of the environment, describe each component and clarify which component does what, dedicate special emphasis to the infrastructures we orchestrate, and show a complete pipeline.

The work done on KPIs in this second year was twofold. We first continued clarifying and refining the meaning of the different KPIs proposed in the description of work and initially reported in D2.1. We then continued by moving a step further and identifying the proper means, tools, and workflows to assess these indicators by proposed deadlines. This document is only in charge of paving the ground to the evaluation of these metrics. deliverable D6.3 provides the actual values we collected and discusses them.

Similarly, to the role played by D2.1, this document summarizes the activities carried out over the second year of the project and can be seen as the preamble of the different activities then carried out in the other work packages. Specifically, this document can be seen as an introduction for D4.2, and D5.2, and should also be considered in conjunction with D6.3, where the consortium presents the intermediate implementation and evaluation of the SODALITE platform and use cases.

The rest of the document is then structured around the organization highlighted above. Section 2 describes the innovation addressed in the second year. Section 3 summarizes the work done on requirements. Section 4 describes the new, ameliorated architecture, and develops around the three main technical work packages of the project. Section 5 describes the workflows identified to assess the different KPI already presented in D2.1. Section 6 concludes the document.

2. Year-two innovation

The main focus of the SODALITE consortium in the second year of the project has been on the extension and consolidation of the SODALITE vision and of the SODALITE results.

2.1 Vision

The context in which the project develops concerns the configuration, deployment and operation of complex applications. Often these are developed by specialists of particular application domains and particular development technologies that, however, are not necessarily expert of the resources from which applications could benefit for their execution. This implies that for such teams it is not easy to take care of IT-intensive tasks such as handling the deployment of complex applications on

© Copyright Beneficiaries of the SODALITE Project

multiple heterogeneous infrastructures, making this process repeatable with no errors, fine tuning the execution of applications in order to keep performance and costs under control.

There are many evidences of the complexity of such tasks that have led to the introduction of the DevOps lifecycle, to reinforce the importance and the advantages of a good cooperation between Dev and Ops, and to the emergence of the Infrastructure as Code (IaC) paradigm, which implies the possibility to write software that defines the way applications should be deployed, configured and executed.

Nowadays, while DevOps and IaC are revolutionizing the way IT-intensive companies work and are organized, they are still too complex and cumbersome to be adopted by other types of organizations which, indeed, would certainly take advantage of them. For instance, adopting the IaC paradigm today means, on the one side, gaining the ability to deploy and configure complex applications very quickly and automatically, but, on the other side, it means getting acquainted to multiple scripting languages and to the corresponding execution mechanisms and being able to manage the whole development process of IaC, including its verification and maintenance.

In this context, SODALITE **provides tools to enable simpler and faster development of IaC and deployment and execution of heterogeneous applications in HPC, Cloud & SW defined computing environments**. Particular focus of SODALITE is on performance, quality, and manageability of the applications on the underlying infrastructures as well as of the corresponding IaC.

Consistently with this vision, SODALITE offers smart modelling capabilities to help non-expert DevOps teams in defining Abstract Application Deployment Models, identification of potential issues in the defined models, automated generation of IaC code, optimization of application execution environments for HPC targets, automatic execution of IaC code so to lead to the deployment of the application, runtime monitoring, efficient scale-in and out or the application and reconfiguration if needed. These features are offered targeting multiple types of infrastructural resources as execution environments for application components. In particular, we focus on classical cloud-based Virtual Machines (VMs), High Performance Computing clusters (HPC), and Kubernetes edge clusters.

While the literature presents several approaches that support some DevOps and IaC activities in a cloud environment, the main novelty of SODALITE is essentially to create a complete framework tackling multiple DevOps aspects and targeting multiple types of resources.

2.2 Innovations introduced in the second project year

While in the first project year we have been focusing on developing the basic mechanisms behind SODALITE (a simple editor, the basis of the semantic support empowering the editor, the integration with a standard orchestration mechanism and an off-the-shelf monitoring system, as well as some initial idea concerning verification and runtime optimization), in the second project year the emphasis has been on extending these basic mechanisms to introduce significant innovations in each area. More specifically, the main results achieved target the following project aspects:

- Extension of the main services supporting the modelling activity and the corresponding generation of executable artifacts:
    - Multiview in-sync representation of AADM: textual, tree-based (read-only), form-based and graphical view representations of the same AADM (and their modelling elements) are available, offering different synchronized representations of the same model for different purposes. Tree-based representations are suitable for overviews, textual representations for fast and efficient modelling and graphical ones for communication purposes.
    - Automated platform discovery produces a TOSCA description of infrastructure: this new mechanism allows identification of specific resources and the creation of a TOSCA resource definition by querying the status of the available infrastructures at

© Copyright Beneficiaries of the SODALITE Project

a certain point in time. This is particularly important as it relieves the Resource Expert (RE) from the need to manually model and define resource types, thus saving significant modelling time. Further details are provided in deliverable D4.2.

- Improved context assistance in model authoring: Context assistance has been noticeably improved for authoring both Resource and Application models, but also implemented for Optimization and Ansible models.

  The user can get more recommendations and the model gets semantically validated under more cases. The context assistance is smarter (receiving context-sensitive suggestions) from the Knowledge Base (KB) in AADM authoring, but it is also available for RM. For the validation recommendations that convey associated solutions, quick fixes are offered to the users, who can then apply them. Context assistance and validation have been improved by 166%, and 200%, respectively.
- Support of TOSCA Policies in the Modelling layer: Ontologies have been enriched for supporting TOSCA Policies both in RMs and AADMs. Using the IDE, the RE and AoE can include those policies in their models. By supporting policies, not only a relevant aspect of the TOSCA specification is covered, but also, according to their definition, the cloud infrastructures could autoscale based on monitor parameters. Furthermore, during the runtime, resources or nodes offering specific policies could be selected.
- Model governance: through the IDE, models created by the user and stored in the KB can be browsed and retrieved for edition. CRUD operations on stored models are supported.
- Improved AADM creation: the process to create an AADM with associated textual and graphical editors is supported through a wizard.
- Improved AADM deployment process: an IDE wizard manages the AADM deployment process and the fulfilment of the required input parameters.
- Modularization of AADM and RM: infrastructure resources and application components can be grouped together in logical units or modules, stored in the KB. Access rights for modules are enabled, restricting the access to those modules to authorized users. KB assistance is also restricted to search in declared modules, improving assistance performance and accuracy.
- Improved scalability in the semantic services:

  Now, semantic services are more scalable through extensive experiments by tuning the configuration of the Knowledge Base, and also by refactoring the reasoning services.
- Support for the creation of Ansible scripts integrated with the Resource Models: this is an add-on that offers users content assistance mechanisms that guide developers in the creation of scripts that are coherent with the definition of the resource models in which context they are used. Further details are provided in deliverable 4.2.
- The MODAK package, a software-defined optimisation framework for containerised HPC and AI applications, is responsible for enabling the static optimisation of applications before deployment. MODAK aims to optimise the performance of application deployment to infrastructure in a software-defined way. Automation in application optimisation is enabled using performance modelling and container technology. Further details are provided in deliverable D4.2.
- Semantic and Analysis Support including: Bug taxonomy; Unified best and bad practices catalog; Unified smell catalog; Linguistic anti-pattern detection via NLP and deep learning; Improved support for detecting smells via a semantic approach. Further details are provided in deliverable 4.2.

© Copyright Beneficiaries of the SODALITE Project

- Extension of the main runtime mechanisms:
  - Advanced orchestration features, including the possibility to reconfigure part of the infrastructure or the deployed application, the parallelization of deployment execution to make it faster, the possibility to restart or resume a failed deployment from the point of failure as well as the definition of a well-designed REST API and support for the newly introduced orchestration features. Further details are provided in deliverable 5.2.
  - Dynamic Monitoring supports the dynamic allocation and configuration of specialized monitoring probes upon the deployment of applications on target infrastructure resources and their decommission when those applications complete their operations. Those probes collect specific metrics on the application and resources activity, which are aggregated into the common monitoring database. Further details are provided in deliverable 5.2.
  - Refactoring: A machine learning based methodology for predicting the performance of alternative deployments of an application; Improved policy-based high-level support for deployment adaptation; TOSCA compliant resource discovery using semantic matchmaking; and Improved control theoretic solutions for the coordinated management of heterogeneous resources through both smart load balancing and fine-grained resource management. Further details are provided in deliverable 5.2.
- Productization of the whole platform:
  - Extension of the whole SODALITE architecture and corresponding platform to cope with authentication and authorization of users as well as to handle the infrastructure secrets such as RSA keys, tokens, passwords needed to access the underlying infrastructures. This aspect is described in Section 4.4 of this deliverable.
  - Development of TOSCA/Ansible blueprints and scripts to automate the deployment of the SODALITE platform itself, in order to simplify adoption and usage by external users. This aspect is described in [D6.3].
  - Introduction to code quality analysis tools (integrated in SonarCloud), definition of quality gates and identification of those metrics to be kept into account to increase code quality. This aspect is described in deliverable 6.3.

3. Updates on requirement elicitation

This section describes the activities carried out in the second year and related to requirements management. The work we did can be organized around three main dimensions:

- We have maintained the initial set of requirements and modified it properly to keep it up-to-date. The evolution of the project imposed us to rethink some requirements, retarget a few, and modify others. This is why Section 3.1 provides the initial set of requirements, which we presented in D2.1. along with the use cases they belong to, and for each requirement we briefly describe its current status.
- We also worked on assessing the use cases presented in D2.1 and this activity resulted in the identification of a new use case, which is reported in Section 3.2.
- Finally, we have continued collecting requirements from our demonstrators, from the technology providers, and also for the market, interpreted in a wider sense, to always complement our requirements with possibly new needs and requests. These new requirements are then presented in Section 3.3.

© Copyright Beneficiaries of the SODALITE Project

3.1 Initial requirements

This section lists all the requirements identified in D2.1 and provides an indication of their implementation. On average, we use simple percentages to quantify the fulfilment of each requirement, and add clarification notes to explain possible delays, deviations, future plans, or deletions.

Id.	Title	Realization (percentage, deleted, or Y3)	Comments
UC1.R1	The SODALITE Design-time environment requires an API to the application/Infrastructure abstract pattern repository	100%	Application and infrastructure abstract patterns can be stored/retrieved to/from the repository
UC1.R2	DSL: specification of application patterns and models	100%	All modelling needs of UCs have been addressed.
UC1.R3	Authoring of application abstract models (part of abstract tuple)	100%	AADM/RM for all UCs are available
UC1.R4	Integration of Application Developer Editor with SODALITE SD	100%	The specification of the application abstract models takes place within the same IDE that the developer uses for designing and implementing her application.
UC1.R5	IntelliJ IDEA IDE extension	deleted	SODALITE IDE uses XText technology that supports the migration of the SODALITE AADM/RM textual editors to IntelliJ. However, the lack of resources prevents us from addressing this requirement without compromising others ranked higher. Besides, this support does not ease the development of the other IDE features for IntelliJ
UC1.R6	Description of application and standard build and run options	75%	Compilation of application in to IaC artifacts is achieved through interface operations that include Ansible DSL
UC1.R7	Support for microservice-oriented architecture	90%	The deployment of microservice-oriented architecture developed artifacts is supported.
UC1.R8	Abstractions and Mechanisms for Enforcing Performance, Security, and Privacy	deleted	The abstractions such as Load Balancer, Queue, Policy Enforcement Points are

© Copyright Beneficiaries of the SODALITE Project

			implemented by the case studies as necessary and are not part of SODALITE stack.
UC1.R9	Augment Application Models, IaC Models, and Infrastructure Models for Predicting Control Objectives	deleted	We use benchmarking for building performance models. In case of the performance modelling for deployment refactoring, the variants in the deployment models are models represented using the feature modelling (a separate model). Thus, so far, there is no need for augmenting the IaC models.
UC1.R10	Modelling language allowing modelling of all the necessary information to enable the generation of deployable IaC	75%	Complete Ansible DSL integration in IDE and RM/AADM DSL pending
UC2.R1	DSL: specification of optimization patterns and models	75%	Partially complete. Abstracting optimizations strategies can be improved further.
UC2.R2	Concretization of abstract models into deployment/configuration plans	50%	Scenarios of concretization - resolving node instance requirements at deployment time using KB inference not fully explored
UC2.R3	OpenFaas modelling for serverless computing actions	Y3	This feature is planned for Y3 of the project.
UC2.R4	SLURM/Torque modelling	100%	Supported
UC2.R5	OpenStack modelling	100%	Supported
UC2.R6	Use context-aware search and discovery, matchmaking and reuse of cloud applications and infrastructures	100%	Supported
UC3.R1	SODALITE Runtime (SR) should support Ansible playbooks and TOSCA node definitions for application deployment in public cloud	75%	AWS and Publicly accessible Openstack instances are currently supported by the SODALITE Runtime Layer
UC3.R2	SR should support Ansible playbooks and TOSCA node definitions for application deployment in HPC environment	100%	Both TORQUE and SLURM deployments are supported
UC3.R3	SR should support Ansible playbooks and TOSCA node definitions for application deployment on edge	70%	Currently support through Kubernetes Edge cluster

© Copyright Beneficiaries of the SODALITE Project

UC3.R4	SR should support Ansible playbooks and TOSCA node definitions for application deployment in fog	70%	Currently support through Kubernetes managed and SSH accessible devices
UC3.R5	Support for SODALITE DSL	100%	Supported
UC3.R6	Generation of correct, complete and deployable IaC artifacts	100%	Supported
UC3.R7	Generation of IaC which exploits heterogeneous architectures	100%	Supported
UC3.R8	Reporting of errors in input models which support error prone and prevent invalid IaC generation	60%	TOSCA and Ansible scripts can be checked for syntax errors. The improved support for control flow errors and the integration with IDE are missing.
UC3.R9	Generation of IaC enabling configuration of runtime components (monitoring, optimization and refactoring) as well as of runtime management policies (refactoring policies, security policies, etc.).	75%	Runtime management policies not yet implemented (Refactoring policies, security policies) Note that, for Refactoring policies, we use one of the widely used business rule languages, Drools, which have a modelling support, that can be installed in SODALITE IDE.
UC3.R10	Generation of IaC which exploits serverless computing artifacts (cloud functions)	Y3	Planned for Y3
UC3.R11	Orchestrator input	90%	Enforcement of TOSCA policies is pending
UC4.R1	Verification of deployment descriptions for syntax and semantic errors	80%	Most syntax and semantic errors for TOSCA are supported. The syntax error detection capability needs to be improved to support the TOSCA 1.3 version.
UC4.R2	Verification of provisioning workflows derived from/specified in the deployment model descriptions	25% (Y3)	Requirement made explicit. This feature is planned for Y3 of the project. The basic support has been developed.
UC5.R1	Predict and Correct Performance Defects in Deployment Models	25% (Y3)	We renamed the requirement and this feature is planned for Y3 of the project. The basic support has been developed.

© Copyright Beneficiaries of the SODALITE Project

UC5.R2	Predict and Correct Security and Privacy Defects in IaC Artifacts	100%	Merged with a UC4 requirement. Detection of common security/privacy smells in TOSCA and Ansible are supported.
UC5.R3	Build an Infrastructure Code Quality Framework	75%	Originally associated with UC4. Structural IaC metrics can be calculated. Control flow metrics are planned in Y3
UC6.R1	SODALITE Runtime supporting various architectures	90%	OpenFaaS support planned in Y3
UC6.R2	Support for extension plugins	90%	OpenFaaS support planned in Y3
UC6.R3	SR should support Ansible playbooks and TOSCA node definitions for application deployment in private cloud.	100%	Private Openstack cloud supported
UC6.R4	SR plugin supporting Docker Compose	100%	SR dos not support docker compose but instead enables modelling and execution of docker hosts, registries, volumes, networks, containers through TOSCA topology models
UC6.R5	Heterogeneous infrastructure	90%	Full support for OpenFaaS and Kubernetes pending
UC7.R1	Lightweight open source Message oriented middleware (MOM) for intra-service communication	deleted	Application owners control and model their own inter-component information flow and middleware - this is an internal application design feature
UC7.R2	Smart application scheduling	100%	Application requests are efficiently scheduled by the Node Manager on fast GPUs or CPUs according to application needs (e.g., SLA, current workload)
UC8.R1	IDE Infrastructure dashboard (monitoring, deployment, reconfiguration)	50%	The Grafana based dashboard is available, showing the monitoring metrics. Additional deployment and reconfiguration information will be rendered in the IDE governance view, released in Y3.
UC8.R2	Collect network metrics	75%	At present, Skydive metrics are simply reported to Prometheus and shown in a dashboard. Y3 - We want to

© Copyright Beneficiaries of the SODALITE Project

			make dynamic changes based on these metrics. This requirement is closely connected with UC8.R9.
UC8.R3	Collect host metrics (CPU, memory)	75%	Node exporters are providing the export of data from the supported infrastructures. The improvements depend on the use cases - their further development is moved to Y3.
UC8.R4	Monitor Overprovisioning (Performance), Security, and Privacy Metrics	deleted	Other monitoring requirements include this requirement
UC8.R5	Monitoring levels	75%	Current monitoring support the collection of metrics a different levels: a) at application level, for HPC jobs, through the HPC exporter, b) at runtime environment, both for VMs, through the Node exporter, and K8s pod, through the K8s exporter, and c) and infrastructure level, through the IPMI and Skydive exporter. In Y3, additional metrics required by validation scenarios will be collected.
UC8.R6	Monitoring infrastructures	75%	Current support for the monitoring from some the supported platforms, including Cloud (through the Node Exporter), HPC (through the HPC exporter) and EDGE (through the K8s exporter). The Y3 plans include to improve collected metrics for all required infrastructures, the detection of anomalous behaviour in monitored metrics and their dashboard visualization.
UC8.R7	End-to-end audit logging	25% (Y3)	Basic support for per-component logging is in-place, but non-repudiation of the audit logs will be done in Y3.
UC8.R8	Visualization of service deployment and adaptations	Y3	IDE View for service deployment management is planned for Y3. View for deployment metrics is

© Copyright Beneficiaries of the SODALITE Project

			currently available through the monitoring dashboard.
UC8.R9	Absorb Skydive metrics	75%	At present, these metrics are simply reported to Prometheus and shown in a dashboard. Y3 - We want to make dynamic changes based on these metrics. This requirement is closely connected with UC8.R2.
UC9.R1	Model Control/Optimization Objectives (Performance, Privacy, and Security)	Y3	The required Quality of Service for the application should be specified.
UC9.R2	Model Design (Adaptation) Choices	100%	The adaptation choices can be modelled using the feature modelling and policy-based adaptation language.
UC9.R3	Find an Optimal Design Solution Considering Control Objective Tradeoffs	75%	An alternative deployment model can be selected. But, the selection process does not consider tradeoffs in quality goals.
UC9.R4	Forecast Workload (Multi-class/tenant)	75%	Regression models are used to forecast workloads. The implementation reuses the existing open source libraries.
UC9.R5	Forecast Infrastructure Dynamics	Y3	This feature is planned for Y3 of the project.
UC9.R6	Predict Violations of Control Objectives (Performance, Security, and Privacy)	50%	The violations of deployment policies and performance goals can be predicted.
UC9.R7	Generate Application and Infrastructure Adaptation Plans	deleted	These requirements are not anymore relevant for the refactorer. The refactorer generates a new deployment model variant and sends it to xOpera, which does the actual adaptation.
UC9.R8	Enact Application and Infrastructure Adaptation Plans	deleted	
UC9.R9	Detect and Correct Defects at Runtime	25% (Y3)	This feature is planned for Y3 of the project. The basic support has been developed.
UC9.R10	Static Provisioning of Heterogeneous Resources	75%	xOpera supports the provisioning of heterogeneous resources (e.g., OpenStackVMs, AWS VMs, HPC resources, and Edge resources). The enforcement of TOSCA policies (on scalability) is missing.

© Copyright Beneficiaries of the SODALITE Project

UC9.R11	Elastic Provisioning of Heterogeneous Resources	75%	Node Manager dynamically scales existing resources according to applications' needs. Deployment Refactorer can perform ad-hoc changes to the deployment topology of the application. The provisioning of the deployment topology can be done at runtime by xOpera. The integration between Node Manager and Deployment Refactoring is missing
UC9.R12	TOSCA inputs to SR	90%	Refactoring Through reconfigured TOSCA blueprints
UC9.R13	Dynamic Policy-based restrictions on resource access from the Edge	50%	Blueprint to support AlertManager integration missing
UC10.R1	Create and Maintain Runtime Models	75%	The runtime models maintained by the refactoring support need to be improved to cover all SODALITE scenarios.
UC10.R2	Horizontal Resource Scalability	Y3	TOSCA auto-scaling policies need to be implemented for cluster node types.
UC10.R3	Vertical Resource Scalability	100%	Node Manager is able to vertically scale resources at runtime according to applications' needs
UC11.R1	Create a Taxonomy of Infrastructure Bugs and Resolutions	90%	Bug and smell taxonomies need to be validated with a survey with developers.
UC12.R1	Select Optimisations for Application and Infrastructure targets	75%	QE selects the optimizations that can be applied to the application based on the available target resources
UC13.R1	Docker Modelling	100%	Container runtime is supported
UC13.R2	Kubernetes Modelling	70%	Currently Covered by deployment to Kuberentes through Helm charts
UC13.R3	Istio Modelling	deleted	It can be handled as a Helm chart. The time was added, Kubernetes were not fully in scope.

© Copyright Beneficiaries of the SODALITE Project

UC13.R4	Ontology Serialization	100%	The semantic model is compliant with OWL2 language
UC13.R5	TOSCA Compliance	75%	SODALITE DSLs are TOSCA compliant
UC13.R6	Authoring of infrastructure abstract models (part of abstract tuple)	75%	KB content-assistance not completed for RMs
UC13.R7	IaaS Modelling	90%	Abstractions of IaaS
UC13.R8	IaC deployment management Modelling	90%	Generation of effective infrastructure code
UC13.R9	Description of the available hardware	90%	Partially covered by automated Platform Discovery Service and extended manually by the RE, if needed
UC14.R1	Estimate Performance of Designs	90%	The performance of the application with many deployment alternatives can be estimated using ML based performance models.
UC14.R2	Estimate Security Level of Designs	75%	The number and type of security smells can be calculated.
UC14.R3	Estimate Privacy Level of Designs	75%	The number and type of security smells can be calculated.
UC14.R4	Assess the Impact of a Design Choice	75%	The performance of the changes to a deployment model variant in terms of selecting or deselecting deployment options can be estimated via ML models.
UC15.R1	Delivery of optimized application	90%	Static application optimization is supported through the definition of the optimization model targeting specific infrastructure and executed via MODAK through the selection of a specific optimized image to be deployed on the infrastructure.
UC15.R2	Optimize Application and Deployment	30% (Y3)	The full implementation of this feature is planned for Y3 of the project. The Application Optimiser currently supports optimisations of AI applications built from source using AI TensorFlow, PyTorch,

© Copyright Beneficiaries of the SODALITE Project

			MXnet frameworks, as well as optimisations through the usage of graph compilers as available for a given framework.
UC16.R6	Lightweight application base images	100%	Application Ops Expert defines the base image from which to build the image

3.2 New UC

This section presents the new UC we added in the second year. It is about the discovery of available platform resources, which can then be used in the foreseen workflows. The approach simplifies the requirement to model the targeted infrastructure, by providing its model automatically.

UC17: Platform Resource Discovery

Actors:	Resource Expert (RE)
Entry condition:	A defined Application Deployment Project Domain (ADPD), defined platform access tokens and undefined platform node types describing for example: computation nodes, networking, volumes, security groups ...
Flow of events:	• RE uses IDE to select an ADPD to which s/he adds the platforms • RE uses IDE to select the platform type (OpenStack, HPC Torque/Slurm, Edge), enters the needed access tokens and connectivity details per platform instance • RE chooses the triggering for Platform Discovery Process updates (e.g. manually, time scheduled, by web hook token) through IDE • RE saves the Platform Discovery Access Definition (PDAD) to the KB from IDE • RE starts the platform discovery process (PDP) for the selected ADPD through the IDE • RE gets the results of platform discovery process in TOSCA and checks the validity of the platform definition • RE saves the valid TOSCA Platform Resource Definition (PRD) to KB for the selected ADPD
Exit condition:	The platform's resource definition is stored in the KB for the selected ADPD
Exceptions:	Errors in the access token definition (PDAD), inaccessible platform endpoint, errors in the TOSCA file definition gathered from the PDP, errors while submitting the PRDs to the KB

© Copyright Beneficiaries of the SODALITE Project

Associated requirements

Id: UC17.R1	**Title** OpenFaaS modelling for serverless computing actions	
Rationale OpenWhisk and OpenFaaS a popular and highly scalable serverless computing / cloud functions platform that allows for functional logic to be written and triggered in response to events or directly via a REST API. These functions play a key role in modern mobile application deployment, and must be managed and utilized alongside other types of conventional infrastructure.	**Scope** Application Components Library	**Use Case** Select Resource (WP3)

Id: UC17.R2	**Title** SLURM/Torque modelling	**Description** Modelling must support SLURM/Torque for HPC.
Rationale Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm requires no kernel modifications for its operation and is relatively self-contained.	**Scope** Application Components Library	**Use Case** Select Resource (WP3)

Id: UC17.R3	**Title** OpenStack modelling	**Description** Modelling must support besides container based deployments also Bare Metal and VM abstractions such as OpenStack.
Rationale OpenStack is an open source cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.	**Scope** Application Components Library	**Use Case** Select Resource (WP3)

Id: UC17.R4	**Title** Use context-aware search and discovery, matchmaking and reuse of cloud applications and	**Description** The existing application designs (or components) and infrastructure should be able to be dynamically discovered and used when optimizing the application.

© Copyright Beneficiaries of the SODALITE Project

	infrastructures		
Rationale The SD needs to use the rule-based semantic reasoning techniques that are developed by CERTH for context-aware search and discovery, matchmaking and reuse of cloud applications and Infrastructures. The deployment improvement module should be able to query and update the semantic repository.		**Scope** Application Components Library	**Use Case** Select Resource (WP3)

3.3 New requirements and their impact on UCs

This section presents the new requirements we collected over the second year. We identified these requirements in different ways: by working on the first version of the environment, by continuing listening to the needs of our first users (demonstrators), and also by analysing the state of the art, which is a continuous activity carried out in the project. Because of the different maturity level of the project, the way they were identified, and also the process we followed, these requirements are broader than the previous ones and encompass multiple UCs.

Id: Y2_R1	**Title:** Alternative distributed field testing	**Deadline:** M36
Use cases AOE: UC1-10, UC16, UC15, RE: UC 12-13, QE: UC14	**Rationale** SODALITE should support alternative distributed deployment configurations for field testing	
Description SODALITE should support the design of multiple deployment models, transformations, execution environments, monitoring and feedback infrastructures. The usage of runtime optimization should identify new deployments. Within the context of a specific deployment model, the usage of the control theory-based optimization should adjust resource usage at runtime. Comparison of HPC and cloud-based scenarios.		

Id: Y2_R2	**Title:** Debugging and testing-oriented pipelines	**Deadline:** M36
Use cases AOE: UC1-10, UC16, UC15, RE: UC12-13, QE: UC14	**Rationale** SODALITE should support the ability of running a reduced pipeline for debugging or test runs	
Description The idea is to have the possibility of creating a "playground environment" for running some components during testing and experiments. We also envision the possibility of defining debug-level policies that allow us to gather monitoring data needed to check the status of the execution. The components can be tagged as skipped (similar to xUnit) in the DSL. Then these components will not be included in the AADM that is sent to WP4 for deployment. If A->B, and B should be skipped, then the KB returns an error.		

© Copyright Beneficiaries of the SODALITE Project

Id: Y2_R3	Title: Advanced component connectors	Deadline: M36
Use cases AOE: UC1-3, UC16, RE: UC12-13	**Rationale** SODALITE should support a more general implementation of component connectors to cope with changing workloads	
Description The modelling support should suggest more scalable and de-coupled ways to connect components so that the portability of the use case and its runtime management could be more effective.		

Id: Y2_R4	Title: Self-adjusting component connectors	Deadline: M36
Use cases AOE: UC1-3, UC16, RE: UC12-13	**Rationale** SODALITE should support the adaptation of connectors when needed at runtime	
Description The runtime support should be able to self-adjust connectors and involved component instances given the actual workload.		

Id: Y2_R5	Title: Parallel workflow executions	Deadline: M36
Use cases AOE: UC1-3, UC16, RE: UC12-13	**Rationale** SODALITE should support modelling of HPC workflows	
Description Besides supporting HPC workflow execution based on simple dependency mechanisms, SODALITE should provide a more extensive support to TOSCA workflows. The runtime layer should recognise the components that can be executed in parallel and execute accordingly. It would be good to have parallel deployment, e.g. in the cases of offline ML training or deployment of independent components for speeding up the deployment time.		

Id: Y2_R6	Title: Incremental workflow re-execution	Deadline: M36
Use cases AOE: UC1, UC3, UC4, UC5, UC7, UC8, UC9, UC10	**Rationale** SODALITE should support restarting a workflow from a failed component	
Description For the cases like in HPC, where the job execution can take several hours (even days), it is not feasible to restart the workflow from the beginning if the workflow failed at some point. The components that were already executed before the failed component can be tagged as		

© Copyright Beneficiaries of the SODALITE Project

"skipped". The IDE should refer to previous deployment for workflow re-execution (or for deployment updates in general) and it is up to the runtime layer to calculate the deltas based on the current deployment.

Id: Y2_R7	**Title:** Easy integration of external components	**Deadline:** M36
Use cases AOE: UC1-7, UC16 RE: UC12-13	**Rationale** SODALITE should provide easier integration of components by providing guidance and suggestion of integration points for a particular component type	
Description Users will always be allowed to integrate new components by hand (e.g. in refactoring), and the specific glue code will not be generated automatically. The IDE should however force the definition of key parameters. For example, for a cloud component, the IDE should stress the "exposed ports" parameter if there is a dependency between another cloud component and this one; for HPC components, the "input/output data" should be stressed.		

Id: Y2_R8	**Title:** Heterogenous edge resources	**Deadline:** M36
Use cases AOE: UC1-UC10, UC17 RE: UC12-13 QE: UC11, UC14	**Rationale** SODALITE should support the discovery of heterogeneous resources at the Edge	
Description SODALITE must be able to understand capabilities of specific Edge Gateways (e.g., through parsing of custom node labels). It can be triggered by the refactoring. Resources should then be introduced in the KB. Three aspects: discovery of instances, discovery of their capabilities (lazy update of the KB based on what is discovered in that particular instance), discovery of resource availability. Nodes must be provisioned before they can advertise their capabilities. The system must also check separately whether a device is available and whether it has already been provisioned.		

Id: Y2_R9	**Title:** Runtime model inference	**Deadline:** M36
Use cases AOE: UC6, UC9	**Rationale** SODALITE should support the runtime discovery of components	
Description SODALITE should be able to automatically get a model from the running infrastructure such as current available resources, their amounts and their types.		

© Copyright Beneficiaries of the SODALITE Project

Id: Y2_R10	Title: Edge-optimized containers	Deadline: M36
Use cases AOE: UC1-UC7, UC16, UC17 RE: UC12-13 QE: UC11, UC14	**Rationale** SODALITE should support containers optimized for edge resources and should allow users to incorporate in these containers the needed elements (e.g., trained models) so that they can be shipped to the edge	
Description Edge infrastructures require specific containers. Base models can be trained in the cloud, but post-processing of the model is required for targeting specific accelerators. This requirement is also related to data movement and compliance.		

Id: Y2_R11	Title: Singularity containers	Deadline: M36
Use cases AOE: UC1-3, UC16	**Rationale** SODALITE should support prebuilt optimized Singularity containers configuration during the IaC generation so that the container can be executed on any infrastructure and can be configured for different optimization	
Description SODALlTE should develop containers for the specific case studies. This means developing containers for different and specific technologies.		

Id: Y2_R12	Title: Multi-arch containers	Deadline: M36
Use cases AOE: UC6-7, UC9-10, UC16	**Rationale** SODALITE should support multi-arch container images. These images also include edge deployment; we could also have multiple variants of containers for different optimizations.	
Description Image builder and container registry must be able to support the different possible target architectures for deployment.		

Id: Y2_R13	Title: Support for automated optimization options	Deadline: M36
Use cases QE: UC14	**Rationale** SODALITE should automate the definition of optimization options for a target infrastructure	

© Copyright Beneficiaries of the SODALITE Project

Description
SODALITE should allow QEs to specify a target infrastructure, the performance model of which shall be derived in an automated way. Based on this model, the optimisation options shall be suggested and/or applied.

Id: Y2_R14	**Title:** Dynamic probing of running components		**Deadline:** M36
Use cases AOE: UC8-9		**Rationale** SODALITE should support the validation/monitoring/alerting that the modelled/discovered components are working properly	
Description The monitoring system must send alerts whenever something does not work. In addition, the RE should be able to define policies and check the behaviour against them.			

Id: Y2_R15	**Title:** Cloud-to-edge refactoring		**Deadline:** M36
Use cases AOE: UC1-10, RE: UC12, UC13		**Rationale** SODALITE should support an extension of deployment refactoring for Cloud-to-Edge deployment. In particular, it should support deployment of microservices for hybrid multi-architecture clusters	
Description SODALITE should target the provisioning and deployment of Edge instances, microservice deployment with suitable container images for hybrid multi-arch clusters (Kubernetes), and serverless functions (OpenFaaS).			

Id: Y2_R16	**Title:** GDPR support		**Deadline:** M36
Use cases AOE: UC4-6, UC8-10		**Rationale** SODALITE should support GDPR awareness and compliance	
Description SODALITE must check privacy compliance and latency at runtime. This also implies the integrated assessment of TOSCA policies and OPA policies. Policies for GDPR compliance needs already exist and can be reused.			

Id: Y2_R17	**Title:** Data privacy		**Deadline:** M36
Use cases AOE: UC1-10, RE: UC12, UC13, QE:		**Rationale** SODALITE should support data privacy	

© Copyright Beneficiaries of the SODALITE Project

UC11, UC14	
Description SODALITE should support outgoing data to check the application against possible data leaks.	

3.4 Workflow

This section presents the main workflows supported by the SODALITE platform. They are focused on three major users of SODALITE - Application Ops Experts, Resource Experts and Quality Experts - plus a secondary user, that is, the system administrator in charge of deploying and configuring the SODALITE platform itself.

In the following we present the workflows associated with these types of users and highlight the artifacts produced in these workflows and where they are located during a normal execution of the SODALITE platform.

3.4.1 The Resource Expert Workflow

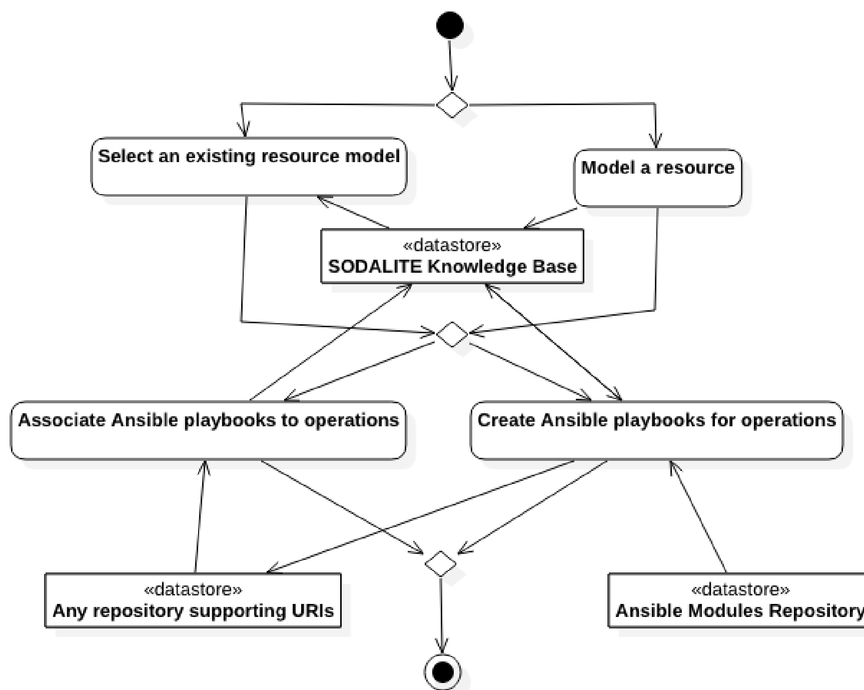

Figure 1 - Resource Expert Workflow.

Figure 1 presents the workflow typically followed by the Resource Expert. He/she is in charge of creating resource models and Ansible playbooks to support the execution of the corresponding operations. In the case a model of the resource under consideration is already available, for instance, because the Platform Discovery has automatically defined the resource, the Resource Expert will limit his/her work to the selection of a specific resource and to the creation or the selection, in case they are already available, of the Ansible Playbooks that implement the operations to be executed for that resource if needed.

The Resource Expert performs his/her activities by exploiting two SODALITE tools, the IDE for all modelling/editing activities and, indirectly, the Platform Discovery.

© Copyright Beneficiaries of the SODALITE Project

The Knowledge Base is the main data store used in this workflow. It includes the resource models and it is updated with the URL of the Ansible scripts associated to such resource models. The Ansible Modules Repository is an off-the-shelf directory offered by the Ansible community and including all available modules[1]. The Ansible playbooks used or produced within the context of SODALITE can be made available on any datastore, including a git repository, that supports their identification through a proper URI.

3.4.2 The Application Ops Expert Workflows

Application Ops Experts are involved in two types of activities within the context of SODALITE, those concerning the design of AADMs and those concerning the execution of the corresponding TOSCA and Ansible scripts and the application runtime.

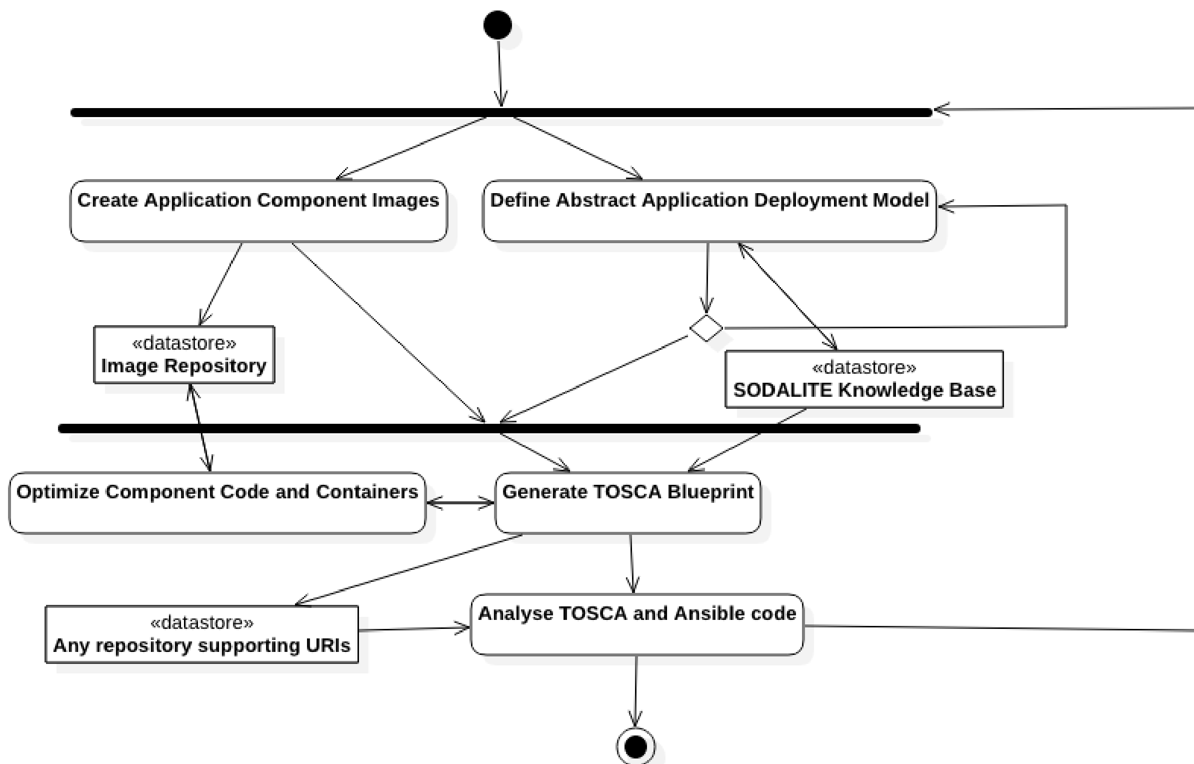

Figure 2 - Design Time Workflow Executed by Application Ops Experts.

Figure 2 shows the design time activities performed by Application Ops Experts to prepare the deployment of a complex application. At the beginning they focus on packaging in proper execution containers the individual application components (we say that they prepare the application component images), this activity is supported by the Image Builder, and, in parallel, on defining the Abstract Application Deployment Model (AADM) through the SODALITE IDE. This last one is an iterative activity that requires the interaction with the SODALITE Knowledge Base and terminates when the user is satisfied by his/her AADM. When images and the AADM are saved in the Image Repository and Knowledge Base, respectively, the AOE generates the TOSCA blueprint. If needed, the optimization of component code and associated containers is performed as part of this phase. The resulting TOSCA blueprint is stored in any repository, e.g., Git, that offers a URI-based mechanism for identifying its elements. Finally, the TOSCA Blueprint, together with the associated Ansible playbooks (defined by the Resource Experts) are analysed to assess the presence of possible problems and bug smells that, if revealed, bring the AADM back into the modelling phase.

© Copyright Beneficiaries of the SODALITE Project

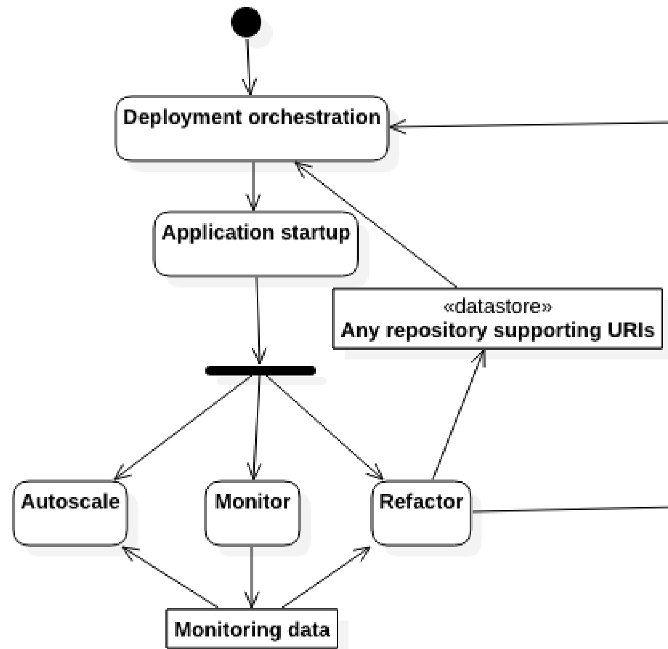

Figure 3 - Runtime Workflow Executed by Application Ops Experts.

Figure 3 describes the runtime activities that are overseen by AOEs. They are all automated, but their result can be inspected through proper dashboards. The process starts with the orchestration of a TOSCA blueprint and the associated Ansible Playbooks. The result of this step, when successful, is the complex application ready to start its execution. After execution starts, the continuous activities concerning monitoring, autoscaling and refactoring are performed. Refactoring can result in changes in the TOSCA blueprint that trigger a new deployment orchestration step.

In this process, monitoring data are produced by the monitoring platform and exploited by the autoscaling mechanism for short-term finetuning and by the refactoring for identifying longer term potential issues. TOSCA blueprints are retrieved and stored, when changed, in any suitable repository as already discussed in reference to the design time activities.

3.4.3 The Quality Expert Workflow

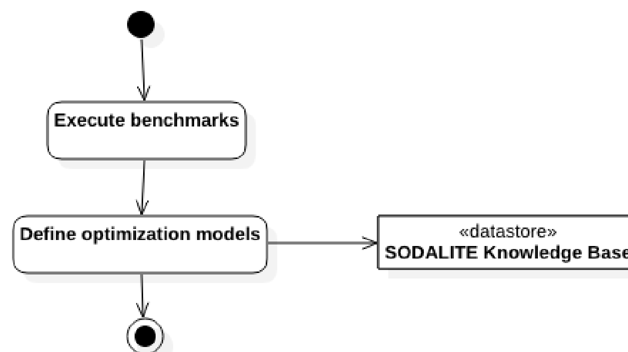

Figure 4 - Workflow executed by the Quality Expert.

The Quality Expert is in charge of developing proper optimization models that constitute the inputs to Application Optimiser (MODAK). He/she is assumed to run, externally to SODALITE, benchmarks to measure the characteristics of available resources. Based on these, he/she defines the optimization models based on the data acquired during the benchmark phase. The creation of

© Copyright Beneficiaries of the SODALITE Project

Optimization Models is supported by IDE while the models are stored in the SODALITE Knowledge Base. Figure 4 provides an overview of the described workflow.

3.4.4 The System Administrator Workflow

The last workflow associated with the usage of SODALITE concerns the activities carried out by the system administrator in charge of making available the SODALITE platform to other users. Given that this platform comprises multiple components, it is, by itself, a complex application. As such, its deployment and configuration has been automated through a proper TOSCA blueprint. This workflow is then completely automated and it is accomplished by following the instructions available on the SODALITE GitHub repository[2].

4. Architecture

4.1 General architecture changes

In the second year of the project there were some additional general architecture changes. Several APIs were refactored and released during this period and some new components introduced.

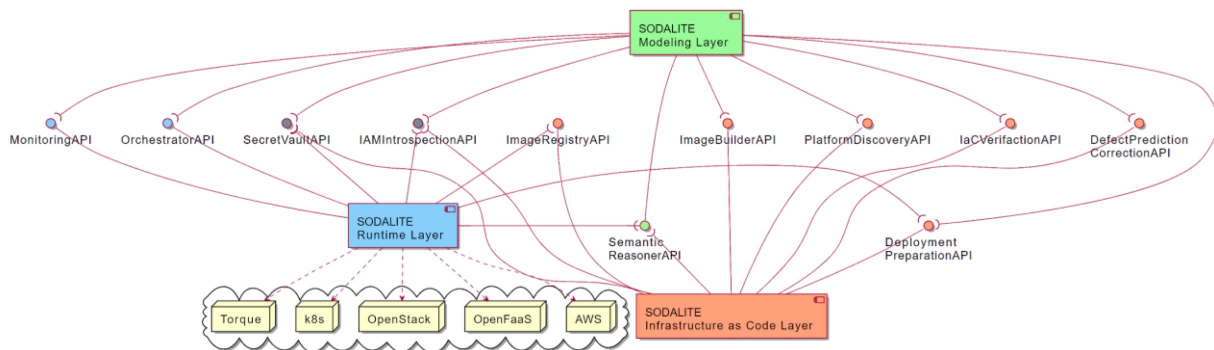

Figure 5 - Updated SODALITE general architecture.

© Copyright Beneficiaries of the SODALITE Project

4.2. Modelling layer

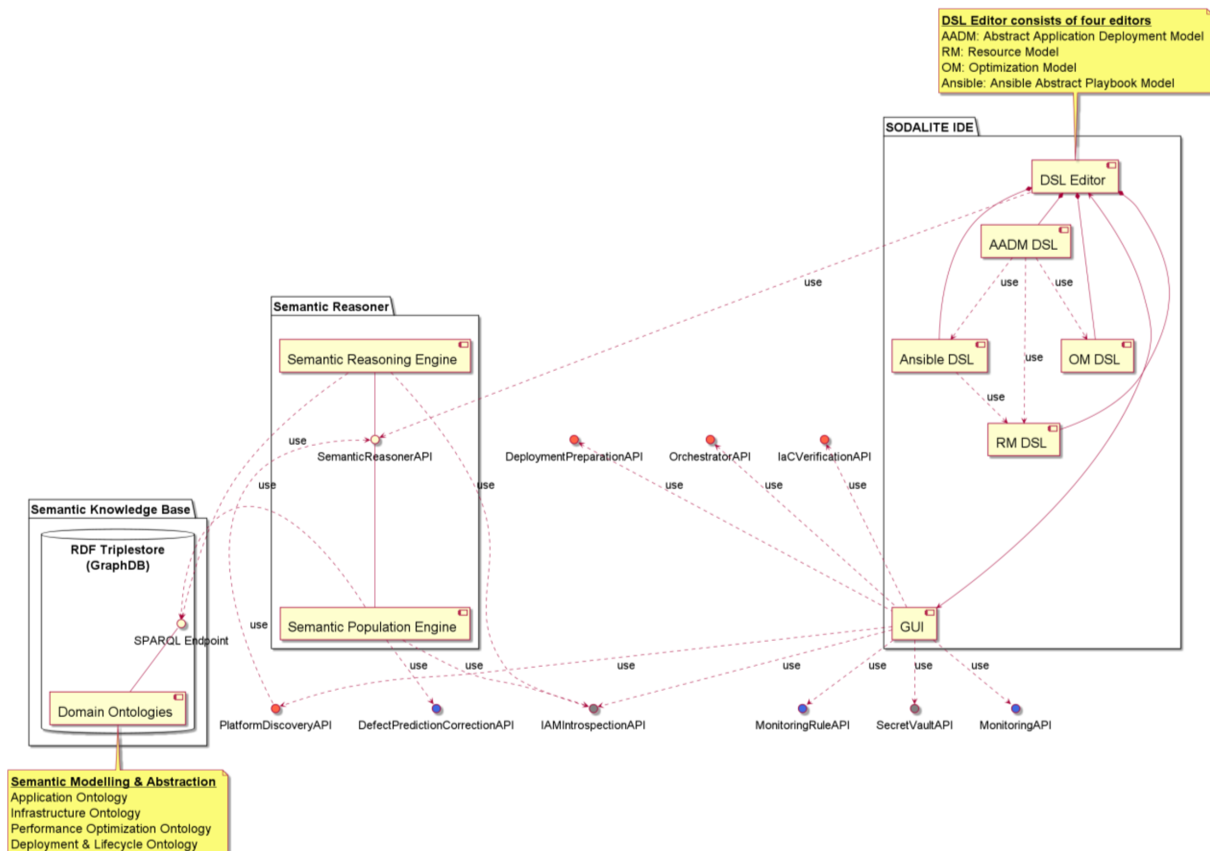

Figure 6 - Updated SODALITE Modelling Layer Architecture.

Figure 6 shows the internal architecture of the SODALITE Modelling Layer. The interfaces offered by other components are also highlighted. A set of SODALITE domain ontologies, resulting from the abstract modelling of the related domains (applications, infrastructure, performance optimisation and deployment), will be hosted in a SPARQL-served RDF Triplestore (GraphDB), constituting SODALITE's Semantic Knowledge Base. A dedicated middleware (Semantic Reasoner) will enable the exploitation of this repository, mediating for the population of data and the application of rule-based Semantic Reasoning. Last but not least, an IDE will provide a user interface with a DSL editor, for the design of deployment models using knowledge retrieved from the Semantic Reasoner. The IDE will also communicate with other system APIs for the monitoring of the deployment lifecycle.

The main changes introduced in the Modelling architecture compared with D2.1 report are the following:

- Semantic Reasoner block: Two new components are interacting with Semantic Reasoner. Namely, it calls the Defect predictor for forwarding warnings of the model to the IDE and is called by Platform Discovery for saving the new discovered models in KB. Also, the access to KB is only permitted to authenticated and authorized users, hence the reasoner calls IAM Introspection API before any operation.
- IDE block: IDE is interacting with two new external components through MonitoringRule API and PlatformDiscoveryService API. The access to IDE is only permitted to authenticated and authorized users (AuthN/AuthZ API).

© Copyright Beneficiaries of the SODALITE Project

4.2.1 Component descriptions

4.2.1.1 SODALITE IDE

Functional Description:

The SODALITE IDE provides complete support for the authoring lifecycle of abstract application deployment models (AADM in the following), Resource Models (RM), Optimization Models (OM) and Ansible Models (AM).

The IDE enables Application Ops Experts (AOE in the following) to create AADMs for their applications. The IDE also permit Resource Experts (RE) to create RMs that defines types of infrastructure reusable resources, Quality Experts (QE) to define OMs that improves the runtime performance of application components in target computing infrastructures, and AOEs to create AMs that defines implementations for the operations of the interfaces adopted by infrastructure resources and applications.

The IDE assists AOEs, REs and QEs in the textual authoring of the AADM (for these models graphical authoring is also supported), RMs, OMs, and AMs, thanks to features such as: a) syntax highlighting, b) autoformatting, c) autocompletion and quick fixes, d) syntactic and semantic validation/error checking, e) scoping (cross-references), f) outlining, g) context-aware smart content-assistance, etc.

AOEs can describe in the AADM the application topology in terms of components and services, their constraints and inter-component boundaries, and also express optimisation requirements or constraints (adopting the QE role) and Ansible implementations for interface operations. RE can describe in the RM reusable types for infrastructure resources, their properties and attributes, the capabilities they offer, the requirements they need, or the policies they adhere to.

The IDE checks the authored models for DSL conformance (syntactic validation) and relies on the Semantic Reasoner for semantic validation (i.e., inconsistencies and/or recommendations). They are presented to the user in the IDE for further inspection. Eventually, the user can refine/amend the model based on them. Additionally, the IDE can request the Semantic Reasoner for existing infrastructure resources that may fulfil requirements expressed in application components or in other resources. Matching resources are presented to the user in the IDE.

Models can be stored into the Semantic KB. Complete CRUD operations on stored models are supported from the IDE. Entities (e.g. application components, infrastructure resources) stored in the KB can be shared with other users.

The IDE also supports the deployment of AADMs into the SODALITE runtime layer by using the IaC Layer.

Input:

- AADM: AOE knowledge, other reusable resources taken from the KB, references to OMs to optimize concrete application components, references to AMs for implementations of operations in interfaces
- RM: RM knowledge, other reusable resources taken from the KB
- OM: OM knowledge
- AM: AOE knowledge, Ansible modules

Output:

1. An AADM to be sent to the Abstract Model Parser and the Orchestrator for deployment
2. An AADM to be sent to the Semantic KB for storage
3. A RM to be sent to the Semantic KB for storage
4. A OM to be bound to AADM components for optimization
5. A AM to be bound to interface operations for AADM components or RM types

Programming languages/tools:

- SODALITE DSL: XText, EMF

© Copyright Beneficiaries of the SODALITE Project

- SODALITE IDE: Eclipse
- SODALITE IDE DSL Editor: XText, Sirius, Java

Dependencies:

1. Semantic Reasoner REST API
2. Semantic Reasoner query language and OWL notation (Turtle)
3. Semantic Reasoner response schema (JSON)
4. IaC Builder REST API
5. xOpera Orchestrator
6. AAI Keycloak REST API

Critical factors:

The latency accessing the SODALITE KB (and retrieving request responses) from the IDE may prevent IDE Editor to present real time recommendations, node targets, etc in the code assistance without some delay. Similar delay could be present when saving models into the KB, or when deploying AADM into the SODALITE runtime layer.

Models (AADM, RM) need to be serialized in the selected OWL Turtle notation before being submitted to the Semantic KB for sharing/reutilization. Therefore, SODALITE DSL and KB Schema must be semantically compatible.

Eclipse DSL technology (XText, EMF, Sirius) might not be fully compatible with a full-fledged Web-based IDE.

4.2.1.2 Semantic Reasoner (Knowledge Base Service - KBS)

Functional Description:

The KBS is middleware facilitating the interaction with the semantic knowledge base (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.

Input:

1. Requests from the SODALITE IDE for the insertion of domain knowledge from Application Ops Experts and Resource Experts (abstract and target resource types, resource patterns, dependencies, inconsistencies, etc.).
2. Requests from the SODALITE IDE for knowledge retrieval in order to present appropriate content in the IDE, to assure alignment with the DSL, etc.
3. Requests from the SODALITE IDE for the qualitative validation of user input (with the help of semantic reasoning).
4. Requests from the SODALITE IDE for recommendations based on the user requirements.
5. Requests from the Platform Discovery Service for inserting of the discovered infrastructure resources into KB.
6. Requests from the Refactoring Option Discoverer for discovering new nodes and resources.

Output:

1. Domain knowledge (abstract and target resource types, resource patterns, dependencies, inconsistencies, etc.)
2. Detected inconsistencies in a given deployment model.
3. Generated recommendations based on user requirements.

Programming languages/tools:

- *Semantic Reasoner API:* Java, JAX-RS REST API
- *Semantic Population Engine:* Java, SPARQL query language

© Copyright Beneficiaries of the SODALITE Project

- *Semantic Reasoning Engine:* Java, SPARQL query language

Dependencies:
- Alignment with SODALITE IDE and its DSL
- Bug Predictor REST API
- AuthN/AuthZ REST API

Critical factors: KB Schema and SODALITE DSL must be semantically compatible.

4.2.1.3 Semantic Knowledge Base (KB)

Functional Description:
The KB is SODALITE's semantic repository that will host the models (ontologies) created in WP3. The ontologies will be populated with domain knowledge, i.e., abstract and target resource types, resource patterns, deployment patterns, dependencies, inconsistencies, etc. This component will interact with the KBS and will offer capabilities for knowledge storage and manipulation.
Input:
Queries from the KBS for the insertion, update, deletion and retrieval of knowledge. More complex queries will also allow the execution of rule-based semantic reasoning and the inference of recommendations and/or inconsistencies.
Output:
Requested domain knowledge, recommendations and inconsistencies.
Programming languages/tools:
1. Semantic triplestore with SPARQL support (GraphDB Free version).
2. SPARQL query language.

Dependencies: /
Critical factors:
The triplestore's scalability needs to be studied, as performance issues might occur upon a great increase in data and querying load. Currently, we improved the KB performance by using different configurations, to have the fastest query response.

4.2.2 Use Case Sequence diagrams

The core activity associated with the modelling layer is the one associated with UC1. However, it depends on the fact that the resources to be used for deploying an application have been specified. For this reason, we focus first on UC13 – model resources.

© Copyright Beneficiaries of the SODALITE Project

4.2.2.1 UC13: Model Resources

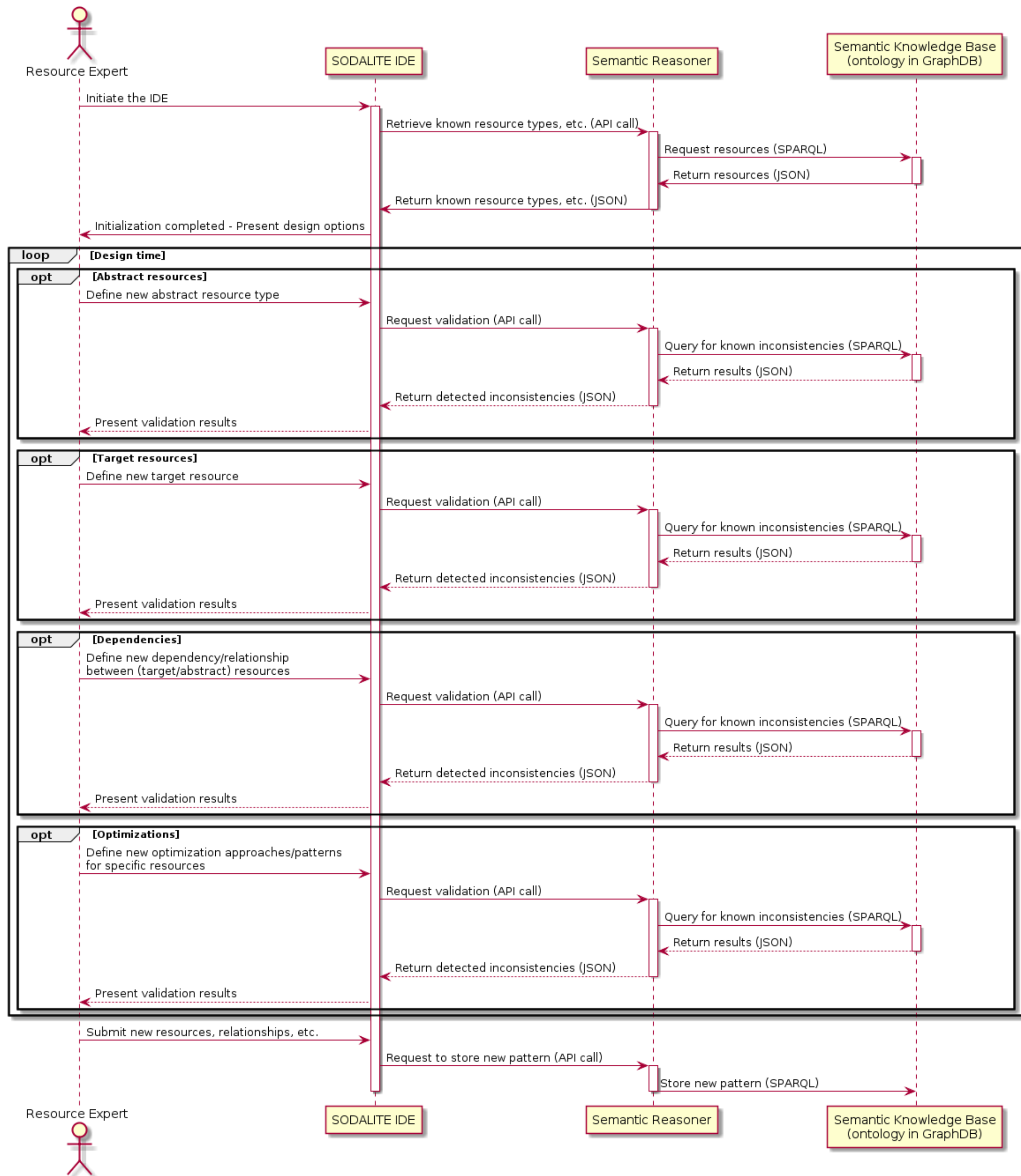

Figure 7 - Sequence diagram for UC13.

Figure 7 describes how the SODALITE components cooperate to implement the features offered as part of UC13 - Model Resources. This use case is initiated by the Resource Expert in order to populate and enrich the KB with new definitions of resource types. New knowledge could regard abstract and/or specific resource types, relationships between known entities (e.g., dependencies between resources), patterns and optimisation approaches. The whole process takes place with the use of the SODALITE IDE and its DSL, assisted by the Semantic Reasoner for the qualitative validation of input and the interaction with the KB.

© Copyright Beneficiaries of the SODALITE Project

4.2.2.2 UC1: Define Application Deployment Model

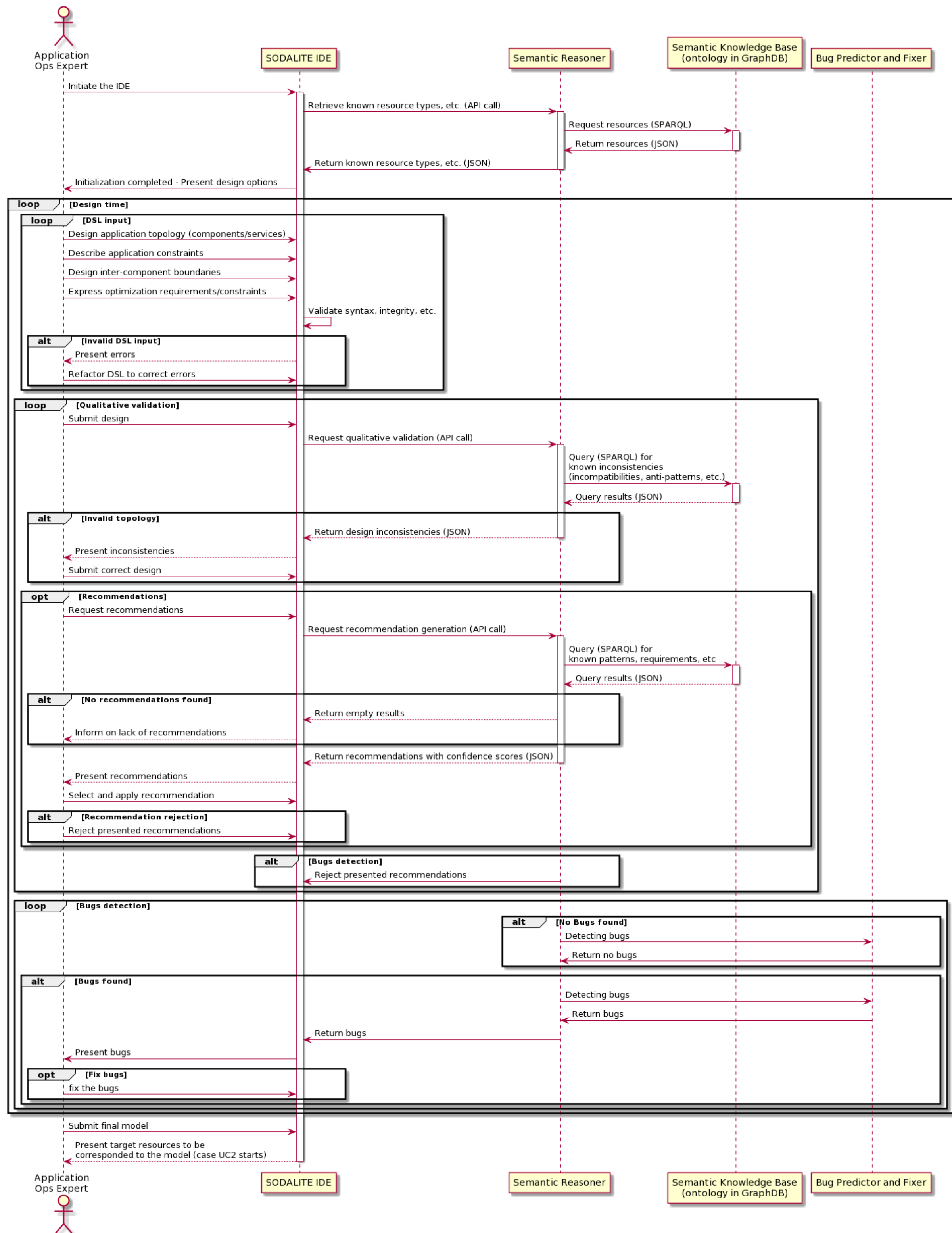

Figure 8 - Sequence diagram for UC1.

Figure 8 models the collaboration between the SODALITE components to implement the features required in UC1. The Application Ops Expert (AOE) uses the SODALITE IDE in order to define an application deployment model (ADM). The IDE is charged with presenting existing knowledge (e.g. resource types), validating user DSL input by detecting inconsistencies, and generating

© Copyright Beneficiaries of the SODALITE Project

recommendations. The required interaction with the KB is served by the Semantic Reasoner component. Finally, bugs and software smells are detected by Bug Predictor. The use case output is a valid ADM.

4.2.2.3 UC2: Select Resources

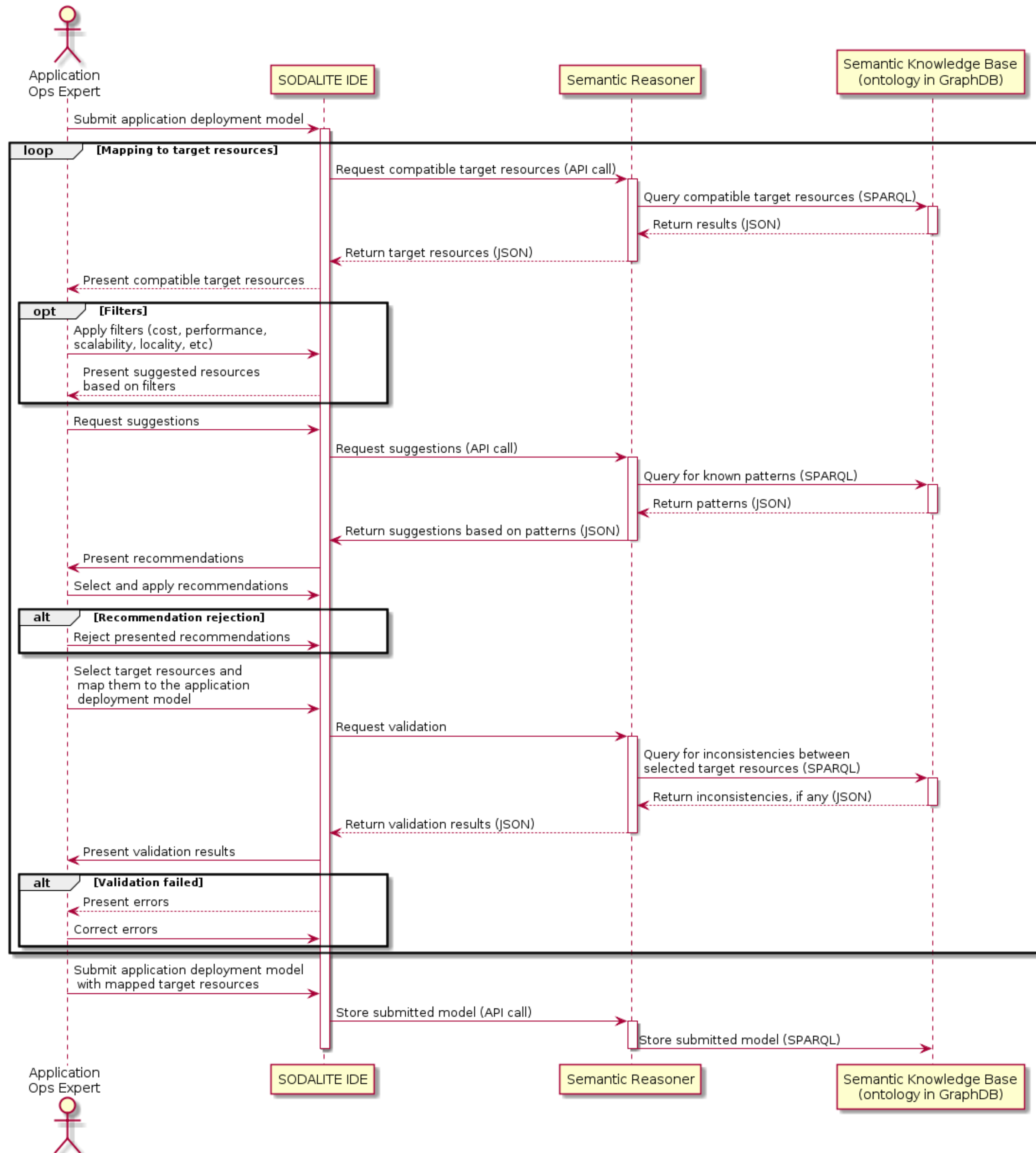

Figure 9 - Sequence diagram for UC2.

Figure 9 models the interaction between the SODALITE components when implementing the features offered within UC2 - Select Resources. As soon as an application deployment model, incorporating abstract resource types, has been defined, a selection of target resources needs to be made and mapped to the abstract types, in order to enable the deployment process. This flow includes the generation of suggestions regarding compatible resources and patterns - to which the

© Copyright Beneficiaries of the SODALITE Project

user will be able to apply filters - and the validation of provided input, with the support of the Semantic Reasoner and information stored in the Semantic Knowledge Base.

4.2.2.4 UC12: Map Resources and Optimisations

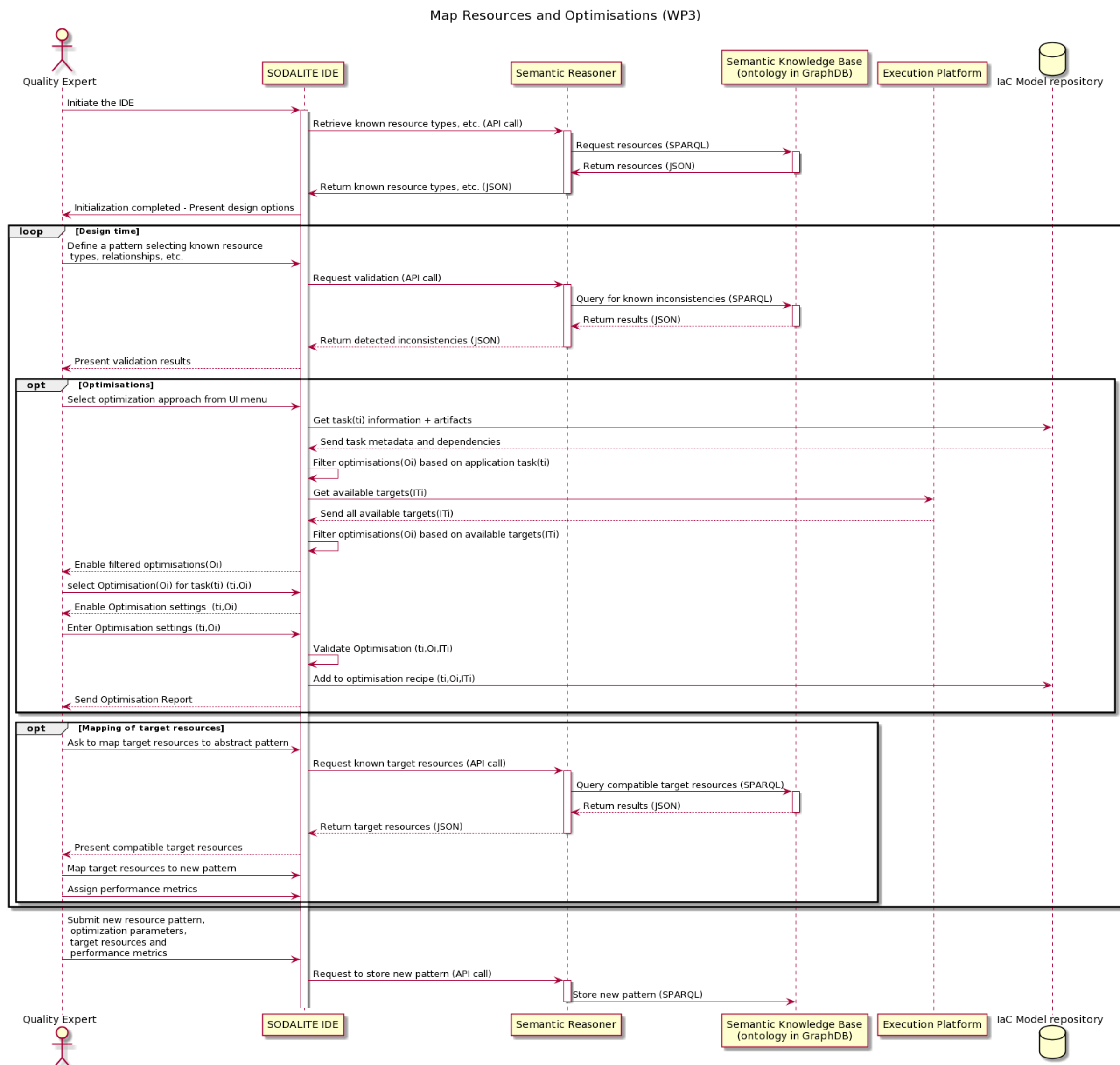

Figure 10 - Sequence diagram for UC12.

Figure 10 describes the interaction between the SODALITE components while implementing UC12 - Map Resources and Optimisations. This use case describes the process of defining abstract resource patterns by a Quality Expert (QE). Additionally, actual (target) resources can be mapped to these patterns. To these ends, the SODALITE IDE retrieves and presents known resource types using the Semantic Reasoner. Finally, the newly generated knowledge is stored in the Semantic Knowledge Base and becomes available in related use cases, such as the aforementioned Select Resources.

Moreover, based on the application and available resource types, different optimisations will be enabled for the QE to select from. The QE will also have to enter the settings for any selected optimisation. This will be stored in the IaC Model repository.

© Copyright Beneficiaries of the SODALITE Project

4.2.2.5 UC14: Estimate Quality Characteristics of Applications and Workload

We do not include a separate sequence diagram for this use case as the Quality Expert in this case performs the quality assessment experiments. In doing so, he/she exploits the whole SODALITE framework to define the Application Deployment Model (UC1) associated with the experimental prototypes used in the assessment:

- select the resources he/she wants to assess for performance (UC2),
- generate the IaC code (UC3) and possibly verify it (UC4),
- execute provisioning, deployment and configuration (UC6),
- start the prototype (UC7),
- run the monitor to collect data (UC8) and, finally,
- edit the resource and application models (UC13) and (UC1) to include additional information about performance.

Alternatively, the Quality Expert could run the experiments in a simulated environment outside the SODALITE framework and then exploit UC13 and UC1 to update the corresponding models in SODALITE.

4.3 Infrastructure as Code Layer

The Infrastructure as Code Layer (IaC Layer) is the layer that connects the SODALITE modelling layer functionalities to Runtime blueprint execution of the models in the SODALITE Runtime Layer. It offers APIs and data to support the optimization, verification and validation process of both Resource Models (RM) and Abstract Application Deployment Models (AADM). However, one of the most important tasks of the IaC Layer is preparing a valid and deployable TOSCA blueprint.

In the second year of the project some of the components were initially released and several were refactored and significantly improved. During this period Platform Discovery Service has been added to the layer's architecture, to expose a REST API which helps to automate the tasks of the Resource Expert by creating a valid TOSCA platform resource model to be stored into the SODALITE's Knowledge Base. These RMs can then be used during the design of the application deployment models (AADM).

In this period Application Optimizer component exposing a REST API (MODAK) was released and integrated into the pipeline enabling the SODALITE users to statically optimize the application for a given target execution platform.

Automation of application optimisation on both HPC and cloud systems requiring models used for performance prediction have been improved. SODALITE prepares and uses these models for both pre-deployment (static) performance optimization and runtime (dynamic) performance optimization.

Additionally, IAM (Identity and Access Management) API and Secret Vault API have been added and partially integrated into IaC Layer and used by the components that have to protect secrets stored by the user such as Platform Discovery Service and IaC Blueprint Builder.

During development in the second year of the project a part of the architecture was redesigned which was also reported in the deliverable D4.2 (IaC Management - Intermediate version) and shown here in Figure 11.

© Copyright Beneficiaries of the SODALITE Project

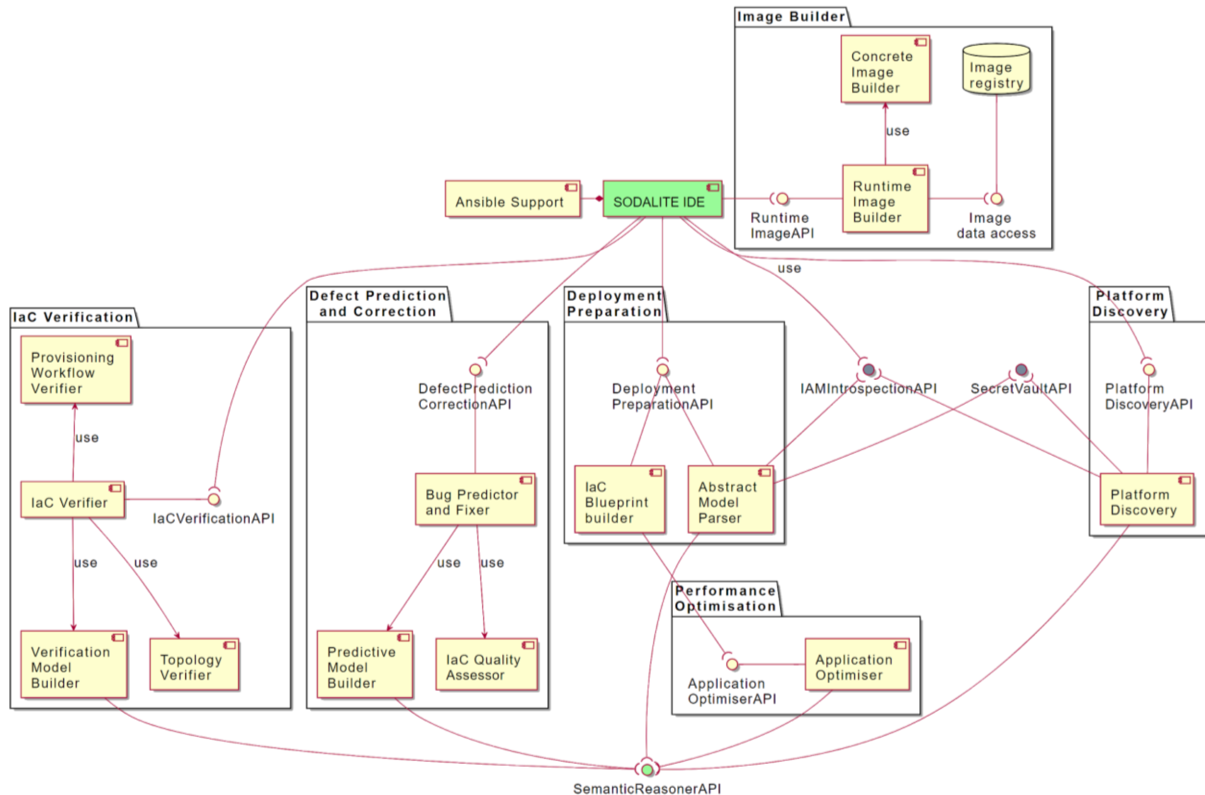

Figure 11 - Updated SODALITE IaC Layer Architecture.

4.3.1 Component Descriptions

4.3.1.1 Abstract Model Parser

Functional Description: The Abstract Model Parser is the central component for the preparation of the deployable IaC blueprint and related Actuation scripts.

Its main function is to abstract the parsing of the abstract deployment model from building the deployable IaC. It feeds the IaC Builder component with all the data provided by the App Ops Expert and needed for the selection and building of IaC Nodes (Blueprint) and preparation of the Actuation scripts (playbooks).

Input: Takes input from the SODALITE IDE as the reference to the abstract application deployment model. It is based on the POLIMI extensive knowledge of modelling and parsing UML deployment diagrams into IaC blueprints, e.g., TOSCA deployment blueprint.

The component allows the SODALITE IDE to:

- start the parsing process
- cancel the parsing process at any given time
- return resulting build time information to the user in a human readable form

Output: Produces the output for the user based on the process of parsing abstract application deployment model.

Programming languages/tools: Java

Dependencies: This component interacts with different components enabling the user to parse the abstract application deployment model and build IaC code through REST API calls to other SODALITE components:

- IaC Blueprint Builder
- IaC Resources Model

© Copyright Beneficiaries of the SODALITE Project

Critical factors: This component should be able to take input from the SODALITE IDE through a web API allowing the user to cancel the parsing process at any given time.

4.3.1.2 IaC Blueprint Builder

Functional Description: This component internally produces the IaC blueprint based on the input provided in the abstract application deployment model passed to the Abstract Model Parser. It flattens the application model topology in a node list and for any given node:

- returns the best matching IaC node definition from the IaC Resources Model repository
- sets provided parameters
- internally builds relations to other nodes

For any selected node it then checks the artefacts to be deployed on that node.

In case the abstract model holds information about the artefact source and the source is available, it triggers the call to the Application Optimiser component in order to try to start the compilation and optimisation, defined in the model.

After all the artefacts are built as runtime binaries and configured, this component calls the Image Builder component to build and pack the artefact images ready for deployment. This component integrates with MODAK which updates the optimized images that are deployed.

At the end of the process of creation of the IaC and the building of Artefact images, it saves the resulting IaC in the IaC Repository and returns the build time information in a human readable form.

Input: Abstract application deployment model, IaC Resources Model

Output: IaC blueprint (TOSCA) with actuation scripts (Ansible playbooks). Returns information about the IaC building process in human readable form to be shown to the user.

Programming languages/tools: Python

Dependencies:

- SODALITE IDE
- Abstract Model Parser
- IaC Resources Model
- Application Optimiser
- IaC Repository

Critical factors: This component should be able to take input from the SODALITE IDE through a web API allowing the user to cancel the IaC building process at any given time.

4.3.1.3 IaC Model Repository

Functional Description: IaC Model repository is a part of the Knowledge Base and contains:

- Performance Model of an infrastructure based on benchmarks.
- Performance Model of an Application based on scaling runs done in the past.
- Mapping of optimisations and applications and their suitability for a particular infrastructure.
- Optimisation recipe for a particular deployment. This contains selected optimisations by the user for an application and infrastructure target.

Input: Application type, node type

Output: Performance model and optimization recipe

Programming languages/tools: Python/MySQL

Dependencies: IaC Model repository interacts with the SODALITE IDE and contains the Performance Model of infrastructure and application (offline analysis).

Critical factors: N/A

© Copyright Beneficiaries of the SODALITE Project

4.3.1.4 Runtime Image Builder

Functional Description: Runtime image builder builds the runtime images used by the orchestrator at application deployment

Input: Target architecture and artifact definition

Output: A runtime image equipped with configuration, artifact executable binary, configuration metadata, possibly monitoring artifact. The image is released to the Image repository for deployment.

Programming languages/tools: Python

Dependencies: Concrete Image Builder

Critical factors: N/A

4.3.1.5 Concrete Image Builder

Functional Description: Implementation of concrete image builder for the execution platform to handle specifics regarding configuration, deployment, monitoring.

As it seems there can be significant differences between the images built targeting HPC/Cloud/Kubernetes, Concrete Image Builder implements an adapter pattern to satisfy and bridge the different approaches for targeting the above-mentioned execution platforms.

The built image could also include monitoring artefacts allowing the post deploy configuration by the Orchestrator.

Input: Runtime Image Builder configuration and definition of binary runtime.

Output: Runtime Image

Programming languages/tools: Yaml (Docker, Kompose, HPC container technology), Python

Dependencies: Runtime Image Builder

Critical factors: N/A

4.3.1.6 Application Optimiser - MODAK

Functional Description: The MODAK package, a software-defined optimisation framework for containerised HPC and AI applications considered within the SODALITE use cases, is the SODALITE component responsible for enabling the static optimisation of applications before deployment.

Input: MODAK requires the following inputs:

1. Job submission options for batch schedulers such as SLURM and TORQUE
2. Application configuration such as application name, run and build commands
3. Optimisation DSL with the specification of the target hardware, software libraries, and optimisations to encode. Also contains inputs for auto-tuning and auto-scaling.

An image registry contains MODAK optimised containers while performance models, optimisation rules and constraints are stored and retrieved from the Model repository. Singularity container technology was chosen to provide a portable and reproducible runtime for the application deployment, due to better performance and native support for HPC.

Output: MODAK produces a job script (for batch submission) and an optimised container that can be used for application deployment.

Programming languages/tools: Python, Ruby, CRESTA Autotuning framework

Dependencies: IaC model repository, Runtime Image Builder, Execution Platform

Critical factors: Overhead time for optimisation of an application. Validation of Optimisation may require support from the execution platform.

© Copyright Beneficiaries of the SODALITE Project

4.3.1.7 IaC Verifier

Functional Description: This component coordinates the processes of verification of the application deployment topology and provisioning workflow/plan.

Input:

- IaC models
- Correctness criteria such as well-structuredness, soundness, and application specific constraints

Output:

- Verification Errors (for invalid artifacts)
- Verification Summary (for valid artifacts)

Programming languages/tools: Java and Python

Dependencies:

- SODALITE IDE
- Verification Model Builder
- Topology Verifier
- Provisioning Workflow Verifier

Critical factors: N/A

4.3.1.8 Verification Model Builder

Functional Description: This component builds the models required to verify the IaC models, for example, a knowledge base instance for ontological (semantic) reasoning on the topology, and a petri net representation for the provisioning workflow.

Input: Abstract IaC models; Verification knowledge (from Semantic Knowledge Base)

Output: Verification Models

Programming languages/tools: Java and Python

Dependencies:

- Semantic Knowledge Base
- Topology Verifier

Critical factors: N/A

4.3.1.9 Topology Verifier

Functional Description: This component verifies the deployment topology of the application against given correctness criteria and application specific constraints.

Input:

- Formal model of the topology
- Correctness criteria
- Application specific constraints

Output:

- Topology Verification Errors (for an invalid topology)
- Topology Verification Summary (for a valid topology)

Programming languages/tools: Java and Python

Dependencies:

- IaC Verifier
- Verification Model Builder

Critical factors: N/A

© Copyright Beneficiaries of the SODALITE Project

4.3.1.10 Provisioning Workflow Verifier

Functional Description: This component verifies the provisioning workflow of the application against given correctness criteria and application specific constraints.

Input:

- Formal Model of the Provisioning Workflow
- Correctness criteria
- Application specific constraints

Output:

- Topology Verification Errors (for an invalid provisioning workflow)
- Topology Verification Summary (for a valid provisioning workflow)

Programming languages/tools: Java and Python

Dependencies:

- IaC Verifier
- Verification Model Builder

Critical factors: N/A

4.3.1.11 Bug Predictor and Fixer

Functional Description: This component is responsible for predicting bugs/smells in IaC models, suggesting corrections or fixes for the detected bugs/smells, and correcting the bugs/smells applying the fix selected by the Application Ops Expert.

Input: Abstract IaC models

Output: Bugs/Smells, Fixes

Programming languages/tools: Java and Python

Dependencies:

- SODALITE IDE
- Semantic Knowledge Base
- Predictive Model Builder
- IaC Quality Assessor

Critical factors: N/A

4.3.1.12 Predictive Model Builder

Functional Description: This component builds the models that can be used to detect bugs/smells in IaC models and suggest corrections. The models can include rule-based models, semantic models, and data-driven (machine learning and deep learning).

Input:

- IaC artifacts
- Bug/Smell and resolution knowledge (ontology and rules)
- IaC datasets
- IaC metrics

Output:

- Ontological Predictive Models
- Data-Driven Predictive Model
- Rule-based Models

Programming languages/tools: Java and Python

Dependencies:

© Copyright Beneficiaries of the SODALITE Project

- Bug Predictor and Fixer
- Semantic Knowledge Base

Critical factors: N/A

4.3.1.13 IaC Quality Assessor

Functional Description: This component can calculate different IaC metrics for IaC artifacts.

Input: IaC artifacts

Output: IaC metrics

Programming languages/tools: Java and Python

Dependencies:

- Bug Predictor and Fixer

Critical factors: N/A

4.3.1.14 Platform Discovery Service

Functional Description: Platform Discovery Service takes the data needed as input platform such as platform namespace, project and credentials to access the platform to create a usable TOSCA Resource Definition from a target. This model can be stored in the SODALITE KB and reused by the AOE at Application Deployment design time.

Input: Target Namespace, Project, Platform Access Credentials .

Output: TOSCA resource definition template

Programming languages/tools: Python, TOSCA,

Dependencies: xOpera, Ansible, Target platforms, IAM API, Vault Secret API

Critical factors: N/A

4.3.2 Use Case Sequence diagrams

During the intensive development phase of the project several architectural changes were needed to improve the overall SODALITE IaC Layer integration. The changes to the Sequence Diagrams are explained in detail in the next subsections describing changes in the UML Use Case Sequence diagrams in detail.

4.3.2.1 UC3: Generate IaC

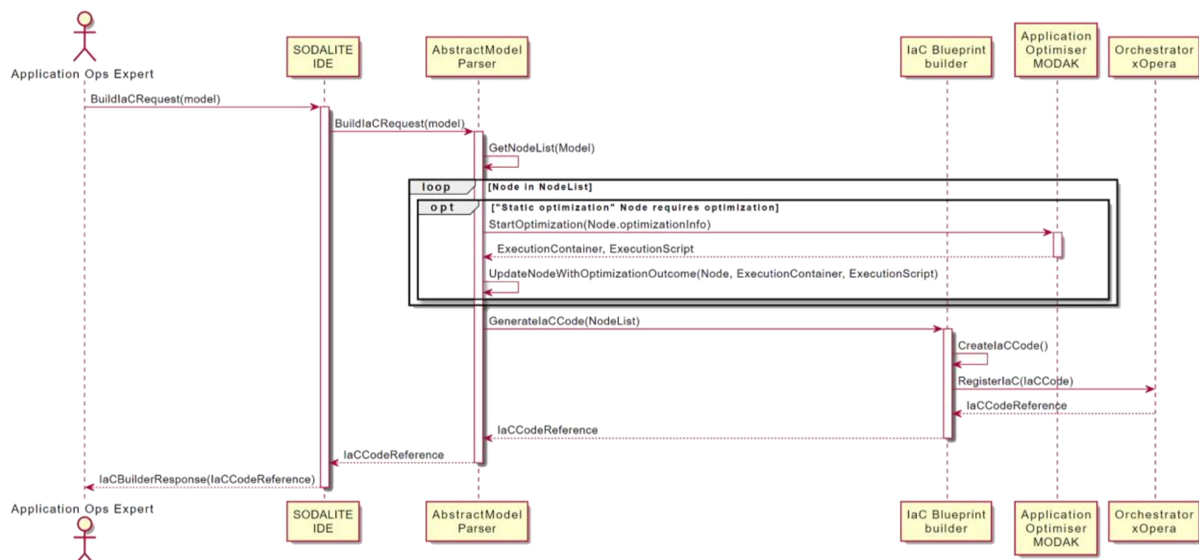

Figure 12 - Sequence diagram for UC3 Generate IaC.

© Copyright Beneficiaries of the SODALITE Project

Figure 12 describes the updated Sequence Diagram showing interaction between the SODALITE components while in the process of generating IaC Code. The prerequisites for the IaC blueprint to be built are a well-defined abstract application deployment model and definition of artifacts, be it source (scripts) or executable binaries with configuration, to be deployed on the infrastructure. Application Ops Experts initiates the generation of the IaC blueprint through SODALITE IDE with the reference to the model definition. Abstract Model Parser parses the model and replaces the abstract node definitions with IaC node definition from the IaC Resource Model which is built into the IaC Blueprint Builder. Each step is tracked and recorded for subsequent IaC changes reflecting the model. For each node, artifacts definitions with source code are then optimally compiled by the Application Optimiser component into an executable binary and optimized images targeting a specific infrastructure platform. IaC Blueprint builder represents a central point of the AADM to IaC transformation. After providing optimal artifacts from the Application Optimiser it creates and registers the TOSCA blueprint with the Orchestrator returning a blueprint registration token. The build-time results are then returned back to the Application Ops Expert through a registered blueprint token provided.

4.3.2.2 UC4: Verify IaC

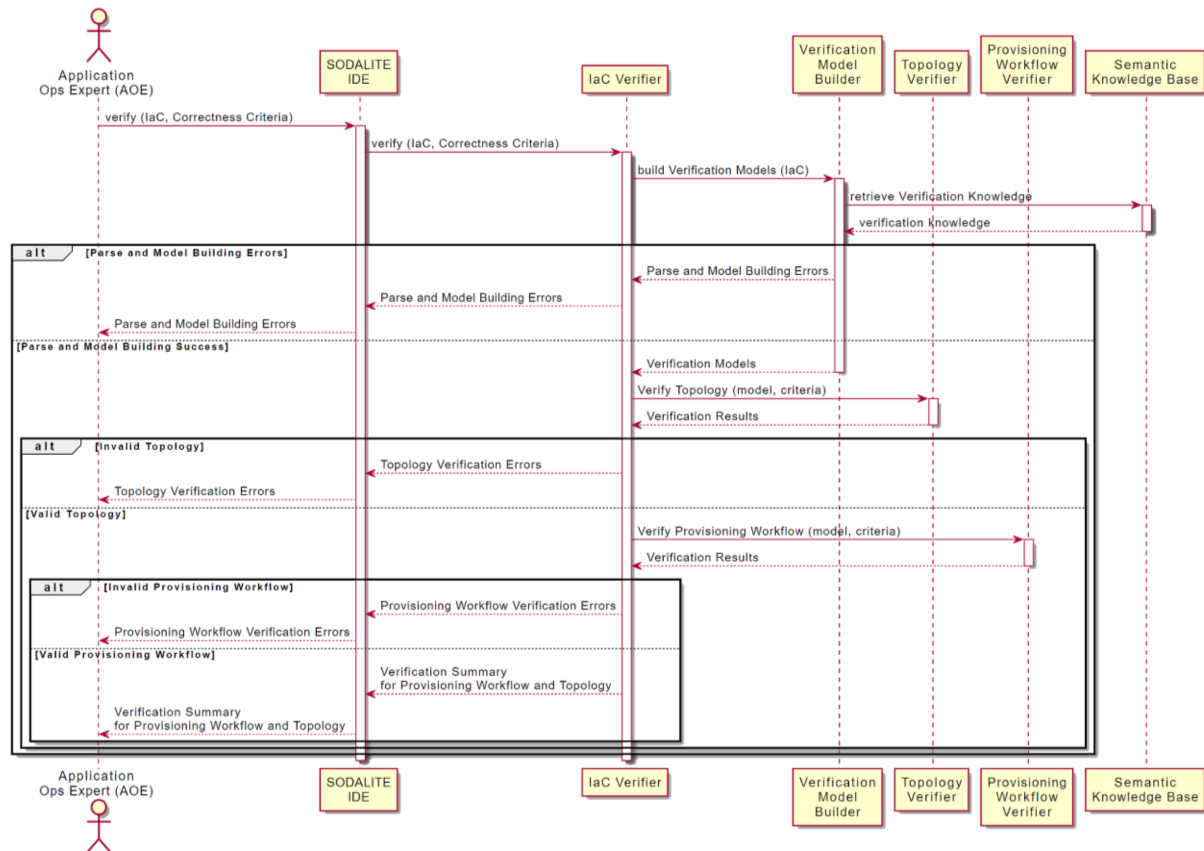

Figure 13 - Sequence diagram for UC4 Verify IaC.

Figure 13 describes the interaction between the SODALITE components while implementing UC4 - Verify IaC. Application Ops Expert provides the abstract IaC modelling artifacts to the IaC Verifier to formally verify the artifacts with respect to given correctness criteria. Both the deployment model and the provisioning workflow of the application needs to be verified. The provisioning workflow includes the provisioning and configuring of the infrastructure, deployment of the application components on the infrastructure, and configuring the infrastructure and the application

© Copyright Beneficiaries of the SODALITE Project

components. Verification Model Builder builds the formal verification models using the verification knowledge from the Semantic Knowledge Base. Topology Verifier verifies the topology whereas the Provisioning Workflow Verifier verifies the provisioning workflow. The verification results are returned back to the Application Ops Expert.

4.3.2.3 UC5: Predict and Correct Bugs

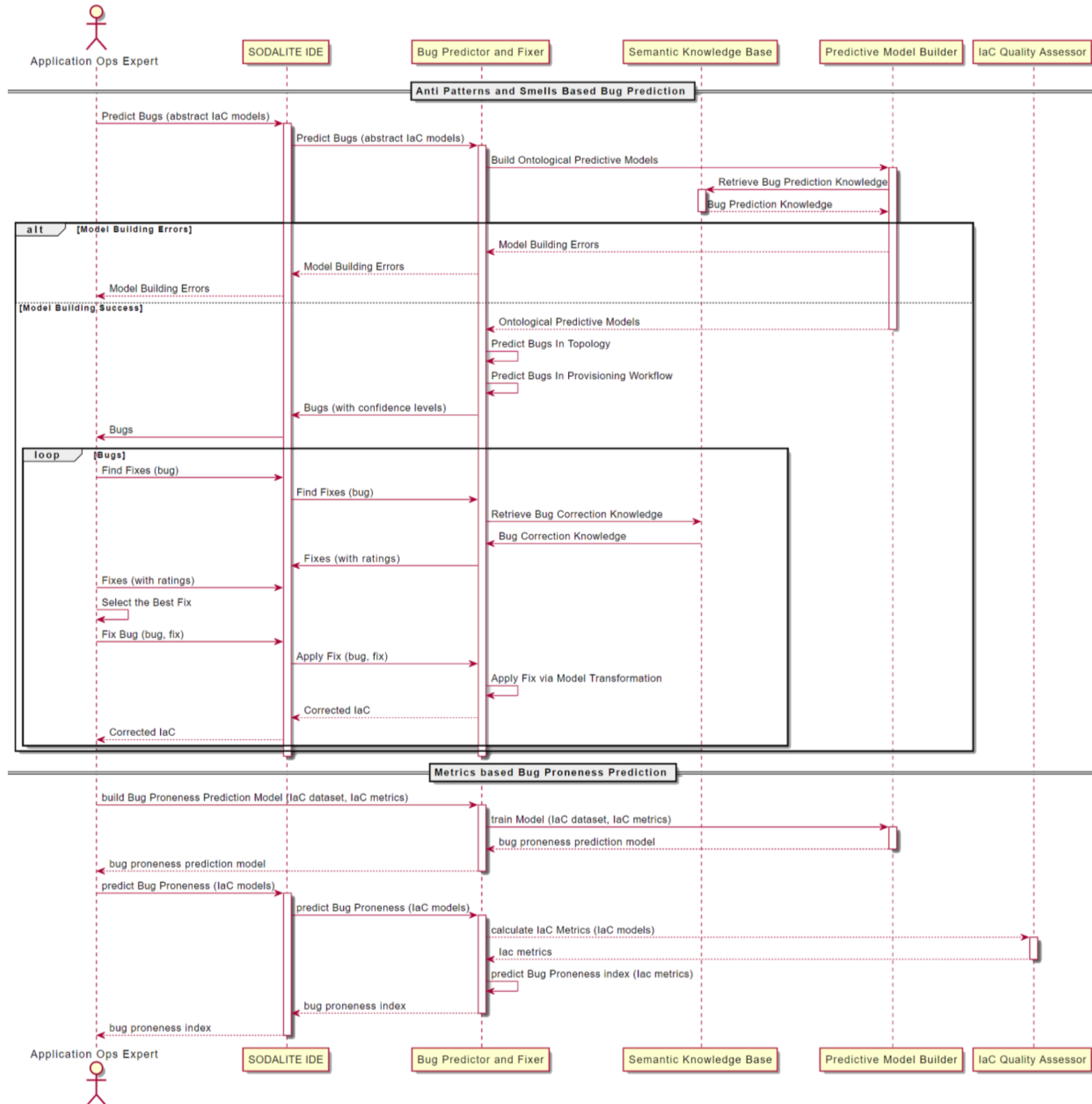

Figure 14 - Sequence diagram for UC5 Predict and Correct Bugs.

Figure 14 describes the interaction between the SODALITE components while implementing UC5 - Predict and Correct Bugs. Application Ops Expert submits the abstract IaC models via SODALITE IDE to Bug Predictor and Fixer for detecting the bugs in the application topology and the provisioning workflow. Bug Predictor and Fixer uses Predictive Model Builder to parse the received IaC models, and builds the predictive models required for predicting the bugs in them. The bugs are anti-patterns, design smells, and code smells for security, privacy and performance. The bug prediction results are shown in SODALITE IDE. Application Ops Expert can select one or more bugs, request potential fixes for each selected bug, and choose and apply the desired fixes. The Semantic

© Copyright Beneficiaries of the SODALITE Project

Knowledge Base contains the knowledge required to predict bugs and to recommend corrections/fixes. Bug Predictor and Fixer can also assess the quality of concrete IaC artifacts in terms of IaC quality metrics and use the IaC metrics to predict the defect-proneness indices in the IaC artifacts.

4.2.2.4 UC11: Define IaC Bugs Taxonomy

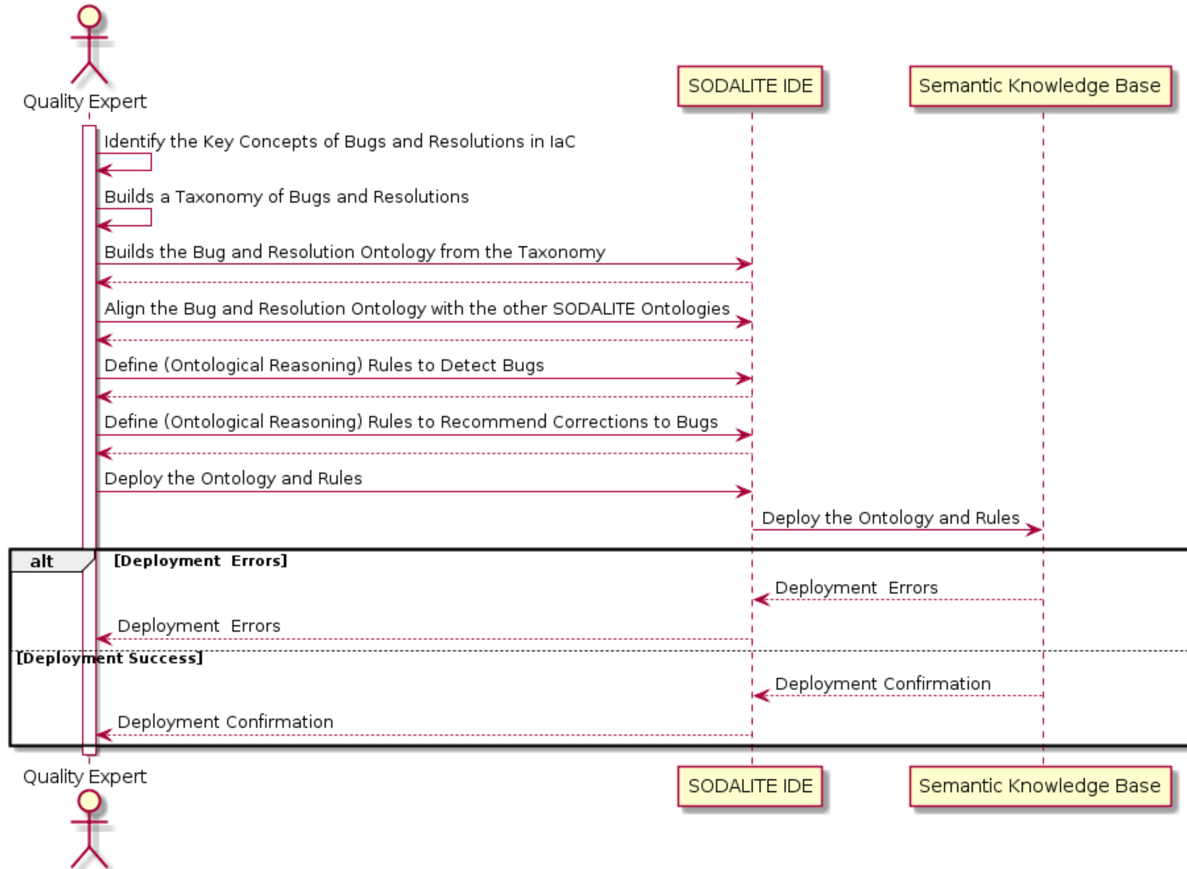

Figure 15 - Sequence diagram for UC11 Define IaC Bugs Taxonomy.

Figure 15 describes the interaction between the SODALITE components while implementing UC11 - Define IaC Bugs Taxonomy. Quality Expert identifies, defines, and organizes a vocabulary of the key concepts used to describe bugs and their resolutions for IaC. Next, Quality Expert uses the vocabulary to classify, relate, and combine all bugs and resolutions, and builds a taxonomy. Then, the taxonomy is defined formally as an ontology. The ontological reasoning rules required for detecting bugs and suggesting fixes for bugs are also defined. Finally, Quality Expert deploys the ontology and rules in Semantic Knowledge Base.

© Copyright Beneficiaries of the SODALITE Project

4.3.2.5 UC15: Statically Optimise Application and Deployment

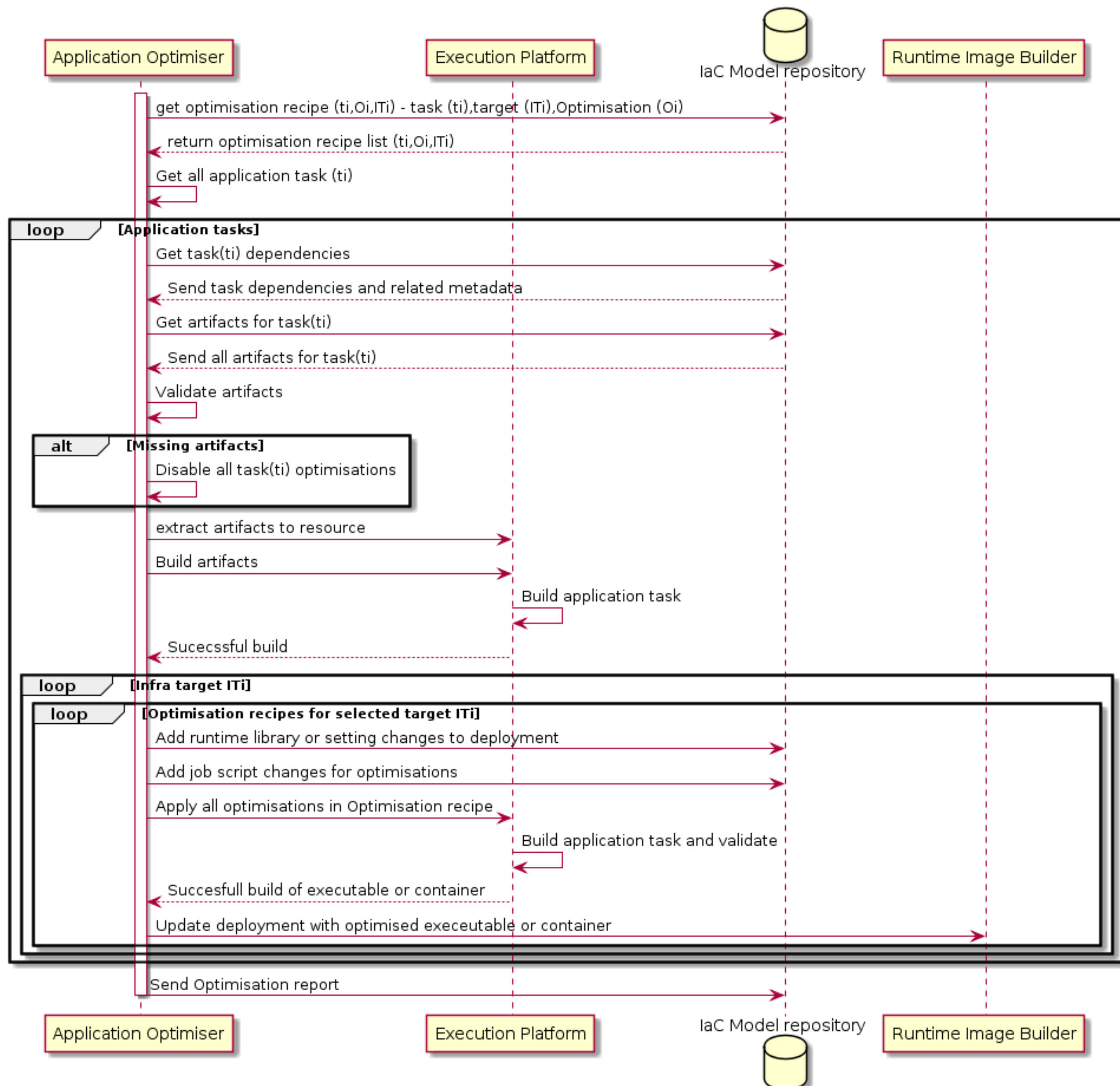

Figure 16 - Sequence diagram for UC15.

Figure 16 describes the interaction between the SODALITE components while implementing UC15 - Statically Optimise Application and Deployment. This use case describes the process for optimising the Application and deployment statically. Static optimisation refers to the optimisation before deployment of the application. The input for this use case is the optimisation recipe created as part of the Map resources and optimisation (WP3) use case and the output will be an optimised application executable or a container. The optimisation recipe stored in the IaC Model repository is retrieved and extracted. For all the tasks in the recipe, the tasks are optimised for different targets based on the optimisations selected. An optimisation report will be made at the end of this process.

© Copyright Beneficiaries of the SODALITE Project

4.3.2.6 UC16: Build Runtime images

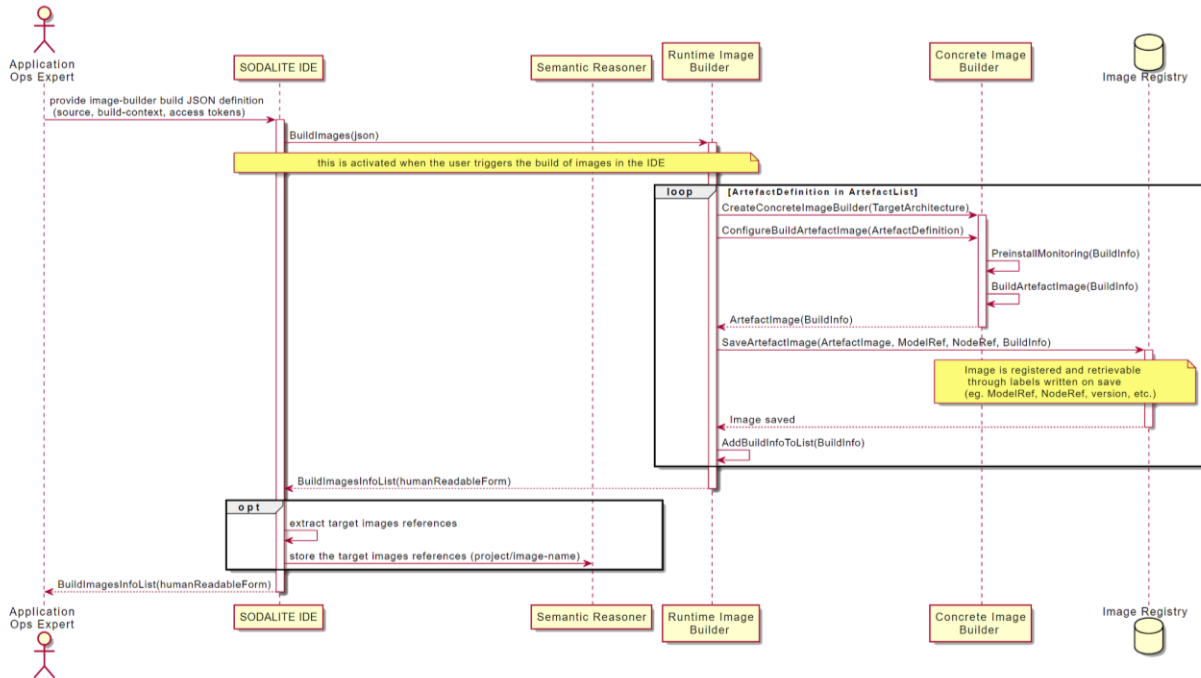

Figure 17 - Sequence diagram for UC16.

Figure 17 describes the interaction between the SODALITE components while implementing UC16 - Build Runtime Images. This is an internal process initiated in UC3 - Generate IaC. Runtime Image Builder builds a runtime image based on tuple definition of target architecture and artifact list for that architecture. Runtime Image Builder activates a specific Concrete Image Builder based on target architecture to prepare a runtime image of the artifact and its configuration with added SODALITE monitoring artifact. The built runtime image is then stored in the Image registry for later deployment. The build-time information is returned to the calling component IaC Blueprint builder.

© Copyright Beneficiaries of the SODALITE Project

4.3.2.6 UC17: Platform Discovery Service

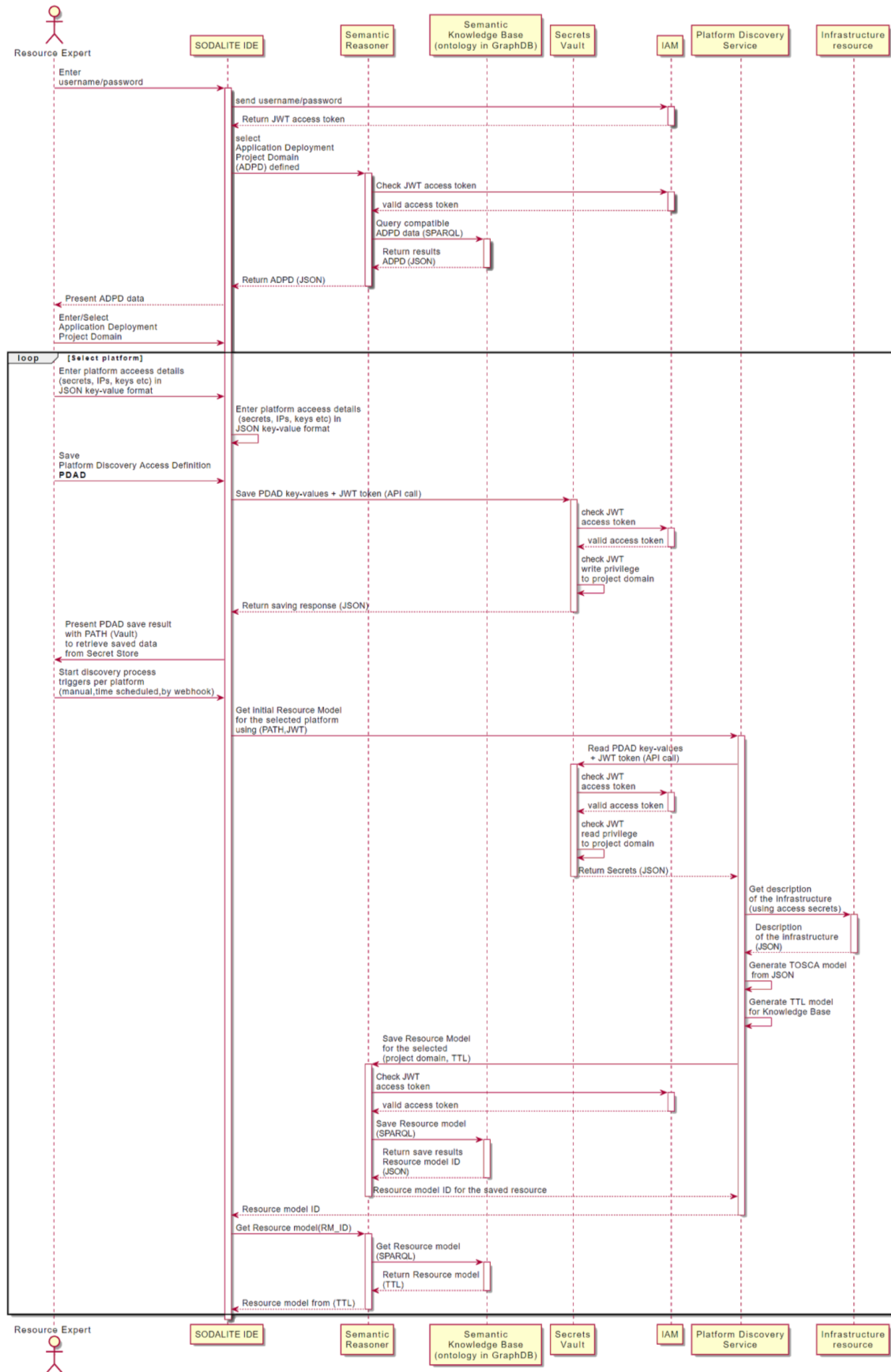

Figure 18 - Sequence diagram for UC17 Platform Discovery Service.

Figure 18 describes the interaction between the SODALITE components while implementing UC16 - Platform Discovery Service. The process is initiated by Resource Expert (RE) by selecting the project

© Copyright Beneficiaries of the SODALITE Project

domain to which a specific set of platform resources will be added. The data about defined project domains is retrieved by Semantic Reasoner API which, based on user privileges defined in the access token, checks the validity and the list of project domains the user is authorized to approach. For every platform added to the project domain the RE adds a specific set of values needed by the Platform Discovery Service to execute the platform discovery. The set of values includes a namespace definition for the created resource, project domain, and platform access keys. The data is stored in a central Secret Vault and accessible by the Platform Discovery Service provided a valid access token for retrieving data in the Secret Vault. The user can manually initiate the discovery of the infrastructure resources through IDE. Once the Platform Discovery Service has access to the platform it executes the discovery using platform specific tools that return a JSON description of the resources. In the next step Platform Discovery Service executes a JSON to TOSCA service template transformation and returns the results to the caller as TOSCA service template definition of the platform or stored into the Knowledge base, depending on the call parameters.

4.3 Runtime Layer

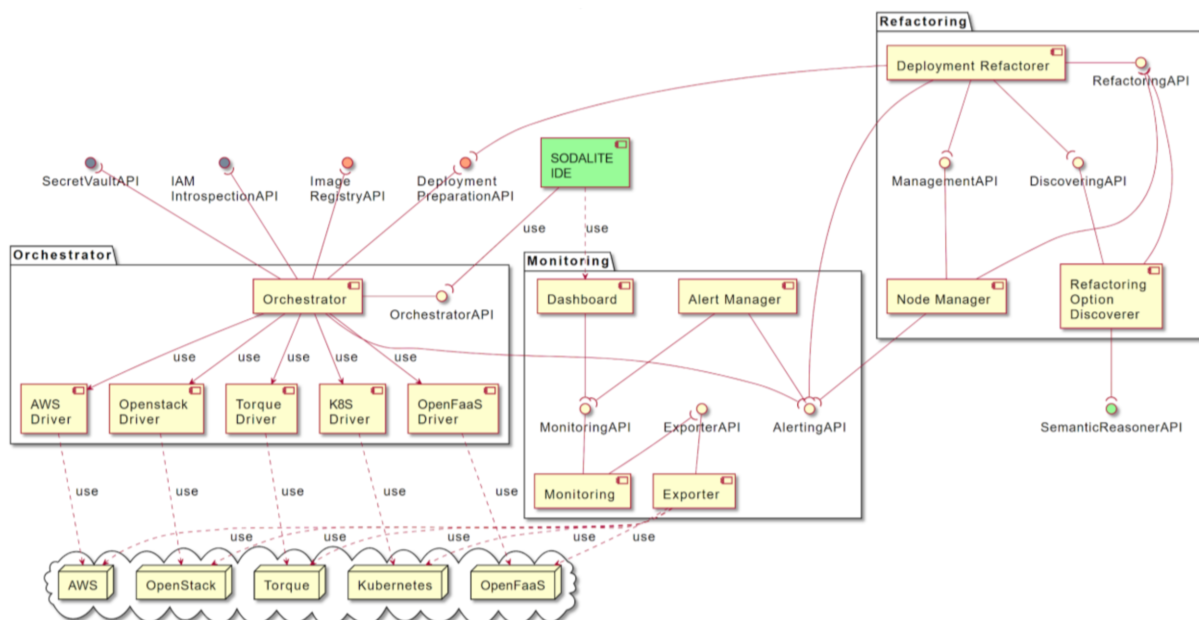

Figure 19 - Updated SODALITE Runtime Layer Architecture.

The Runtime layer of SODALITE (see Figure 19) is in charge of the deployment of SODALITE applications into heterogeneous infrastructures, its monitoring and the refactoring of the deployment in response to violations in the application goals. It is composed of the following main blocks:

- Orchestrator - receives the application to be deployed or re-deployed as a blueprint expressed in TOSCA, deploying the application components on the appropriate infrastructure.
- Monitoring - monitors the application components and the infrastructure where they are deployed to be used by Refactoring and the interested SODALITE actors.
- Refactoring - is able to propose a new application model to fulfil the application goals. When it modifies the model in the Semantic Reasoner, it calls the Deployment Preparation, which will trigger the generation of a new blueprint that arrives to the Orchestrator to initiate the redeployment.

© Copyright Beneficiaries of the SODALITE Project

The main changes introduced in the runtime architecture w.r.t. the version reported in D2.1 are the following:

- Orchestration block: the drivers supported and integrated within the orchestrator are made explicit, namely the Openstack (OS), Torque, K8S, OpenFaaS and the AWS drivers, which interfaces Openstack HPC schedulers, Kubernetes, OpenFaaS and AWS infrastructures, respectively. It also incorporates additional required interfaces for AAI (e.g. IAMIntrospectionAPI) and for the retrieval of deployment secrets (e.g. SecretVaultAPI).
- Monitoring block: this block includes new components not previously included in the former architecture:
    - Monitoring Dashboard: this frontend provides monitoring specialized visual reports for selected target application components and execution environments. Dashboard uses the MonitoringAPI REST interfaces to query monitoring data.
    - Monitoring Alert Manager: manages defined alerting rules to trigger notifications to subscribers when the rule condition holds for the target monitoring statistics. The AlertingAPI REST interface is used by the Orchestrator, Node Manager and Deployment Refactorer components to subscribe themselves to concrete alerts.
- Refactoring block: the internal interactions among this block components have been made explicit in the architecture, through the REST API exposed by each component.

4.3.1 Component Descriptions

4.3.1.1 xOpera REST API

Functional Description: The xOpera Orchestrator manages the lifecycle of an application deployed in heterogeneous infrastructures. xOpera REST API is deployed as a dockerized component that encapsulates xOpera orchestrator and provides additional functionalities to the API clients, such as TOSCA blueprint registration, deployment session handling, blueprint and session persistence with the possibility to share registered TOSCA blueprints among different users.

Input: TOSCA/Ansible Blueprint Deployment plan

Output: Configuration of target infrastructures and applications +

Programming languages/tools: Python, Ansible

Dependencies: Target infrastructures: HPC (TORQUE/SLURM), OpenStack, AWS, Kubernetes

Critical factors: Each orchestrator has its own limitations. This results in limitations concerning the possibility to apply certain actions on the managed application. For instance, since Cloudify does not support SubstitutableNodes, it does not support the Refactoring feature.

4.3.1.2 Monitoring + Exporters

Functional Description: Gathers metrics from the heterogeneous infrastructure and application execution, provided by standard and specialized exporters, allowing query and aggregation on them.

Input: Heterogeneous infrastructure

Output: Metrics

Programming languages/tools: Prometheus, Prometheus query language and API, Go for exporters

Dependencies: Probes or exporters that monitor the target infrastructure components and reports back metrics .

Critical factors: Certain metrics could be difficult to get in some infrastructures.

© Copyright Beneficiaries of the SODALITE Project

4.3.1.2 Monitoring Dashboard

Functional Description: offers visual, customizable, specialized views that renders different monitoring facets of target infrastructures or applications.

Input: monitoring metrics

Output: monitoring views

Programming languages/tools: Grafana

Dependencies: Monitoring metrics collected from querying the monitoring component

Critical factors: Certain metrics could be difficult to get in some infrastructures.

4.3.1.2 Alert Manager

Functional Description: this component is notified by the monitoring component upon the detection of monitoring violations specified in registered alert rules. The manager conducts an analysis of the accompanying metrics and reports associated alerts to the interested registered subscribers.

Input: monitoring alert + associated metrics

Output: alert notification send to subscribers

Programming languages/tools: Python, Flask, Gunicorn

Dependencies: registration in monitoring as alert manager, SODALITE subscribers registered.

Critical factors: N/A.

4.3.1.3 Deployment Refactorer

Functional Description: This component refactors the deployment model of an application in response to violations in the application goals. It also derives the node-level goals from the application goals. The goals are monitored at runtime by collecting the necessary metrics. A machine learning based predictive model is used to predict the performance of multiple alternative deployment model variants, and to select a suitable deployment model variant for the application (the new deployment model) if the current deployment model leads performance violations. The new deployment model is deployed through Orchestrator. The new refactoring options can also be discovered at runtime, enabling deriving new deployment model variants. The Refactorer can also detect various anomalies in the performance of a given application deployment at runtime using machine learning based models, and generate the alerts, enabling executing the corrective actions.

Input:

- IaC topology model
- Refactoring option model
- Application goals
- QoS metrics

Output: (Topology) Adaptation Plan or New Deployment Model (in TOSCA and IaC Scripts)

Programming languages/tools: Java

Dependencies:

- Refactoring Option Discoverer
- Node Manager
- Deployment Preparation
- Orchestrator
- Semantic Reasoner
- Monitoring Agent

Critical factors: N/A

This component addresses the following application goals:

© Copyright Beneficiaries of the SODALITE Project

- satisfy performance (latency and throughput)
- minimize cost/price
- minimize resource usage

And uses these data retrieved from the monitoring infrastructure:

- application workload, latency, and throughput
- cost/price
- infrastructure resource usage metrics such as CPU, Memory, and Network
- other metrics such as energy metrics, and HPC-specific metrics

If data is not sufficient or of good quality, the accuracy and effectiveness of the refactoring/adaptation decisions may decrease. Thus, as necessary, the Refactorer will use the existing techniques [8][9] for handling uncertainty and variable quality of the monitored data.

4.3.1.4 Node Manager

Functional Description: This component is responsible for managing node resources including the overall node capacity/throughput while maintaining the node goals assigned by the Deployment Refactorer. The node goals are monitored at runtime by collecting the necessary metrics. The Node Manager oversees multiple concurrent applications. The Node Manager schedules incoming requests for execution on GPUs and CPUs exploiting custom heuristics and continuously scales CPU cores using control-theory according to applications' needs.

Input:

- Node goals
- QoS metrics
- Available resources

Output:

- Load balancing on heterogeneous resources
- Resource Allocation

Programming languages/tools: Python, Kubernetes

Dependencies:

- Deployment Refactorer
- Orchestrator
- Semantic Reasoner
- Monitoring Agent

Critical factors: N/A

4.3.1.5 Refactoring Option Discoverer

Functional Description: This component is responsible for discovering new refactoring options and changes to existing refactoring options. To select refactoring options, it uses various matchmaking criteria based on the properties, capabilities, requirements, usage policies of resources. For example, a new node that may offer a specific security policy (e.g., the node is placed only on a data center in a given set of regions) or scaling policy (e.g., the node can be autoscaled up to 5 instances).

Input: Search (matchmaking) criteria

Output: Refactoring options

Programming languages/tools: Java

Dependencies:

- Deployment Refactorer
- Monitoring Agent
- Semantic Reasoner

© Copyright Beneficiaries of the SODALITE Project

Critical factors: N/A

4.3.2 Sequence Diagrams

4.3.2.1 UC6: Execute Provisioning, Deployment and Configuration

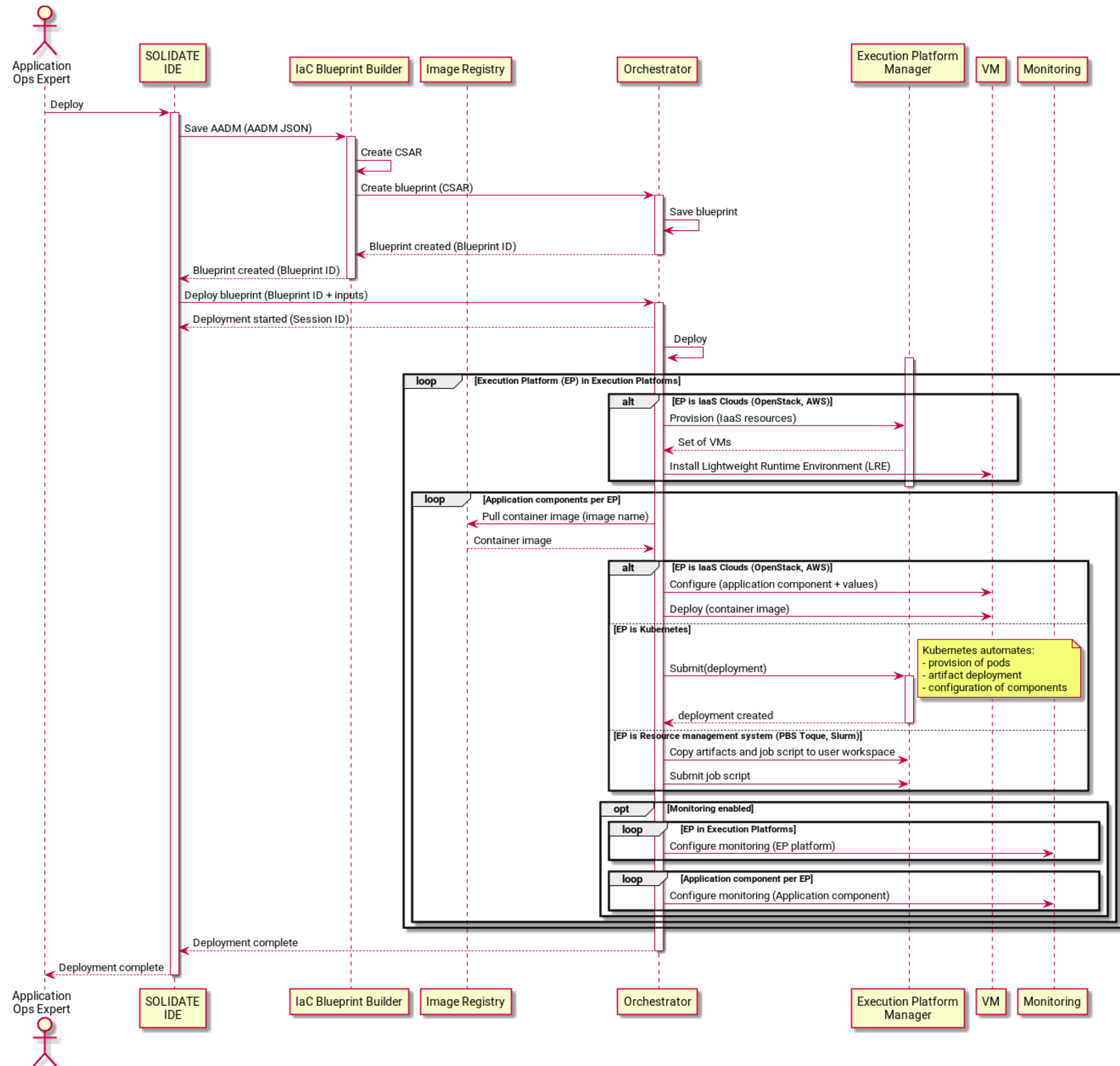

Figure 20 - Sequence diagram for UC6.

Figure 20 describes the interaction between the SODALITE components while implementing UC6 - Execute Provisioning, Deployment and Configuration. Once an AADM (abstract application deployment model) has been defined, an Application Ops Expert initiates the deployment via the SODALITE IDE. The SODALITE IDE provides the AADM to the IaC Blueprint Builder, which in turn creates a CSAR (Cloud Service Archive - an archive that contains TOSCA blueprints and metadata, and the deployment and implementation artifacts) and passes it to the Orchestrator.

The Orchestrator saves the CSAR and returns a Blueprint ID to the IaC Blueprint Builder, which is further passed back to SODALITE IDE. At this point, the deployment is registered in the Orchestrator system and can be later referred through its Blueprint ID, e.g. for the deployment execution or deployment updates.

© Copyright Beneficiaries of the SODALITE Project

Once the Blueprint ID is received by the SODALITE IDE, it directly requests the Orchestrator to start the deployment and receives a Session ID to monitor the deployment progress. Optionally, a set of inputs can be passed along the request to parameterise the deployment. Then, the Orchestrator starts the deployment by executing the deployment workflow specified in the blueprint.

For each execution platform specified in the blueprint, its resources are instantiated, and the application components are deployed on top of them. As such:

- In IaaS clouds, before the deployment of the application components, the virtual resources must be first provisioned. For that, the Orchestrator issues provision requests to the IaaS resource manager (e.g. OpenStack, AWS EC2) to create a set of virtual machines (VMs) and other resources that the application demands, e.g. security group, network and storage. The LRE is then installed on VMs as a runtime for the execution of the application components. Upon the installation, the Orchestrator configures and deploys application components, pulling them from specified image registries.
- Kubernetes automates the resource provisioning and the application deployment by exposing an endpoint, through which the deployment and configuration descriptions are passed. Hence, the Orchestrator submits these descriptions to Kubernetes, which configures and deploys the application components, pulled from specified image registries.
- When resource management systems are selected (e.g. Torque or Slurm), the Orchestrator pre-uploads the artifacts (e.g. pulling required container images) and the job description script to the user workspace (e.g. home directory of the user) on login (front-end) nodes and then submits the job to the batch system. UC7 describes the start of applications, deployed using one of the resource management systems.

At this point, the deployment of the heterogeneous application components is performed on different resources, and the application is started. Optionally, the configuration of a monitoring platform can be additionally performed if such mechanism is requested to the Orchestrator. First, for each allocated Execution Platform (EP), one or more different Monitoring Exporters (i.e. a monitoring probes) are registered and configured within the monitoring agent.

Then, a similar exporter registration and configuration process is conducted for each application component that requires a specialized exporter.

© Copyright Beneficiaries of the SODALITE Project

4.3.2.2 UC7: Start Application

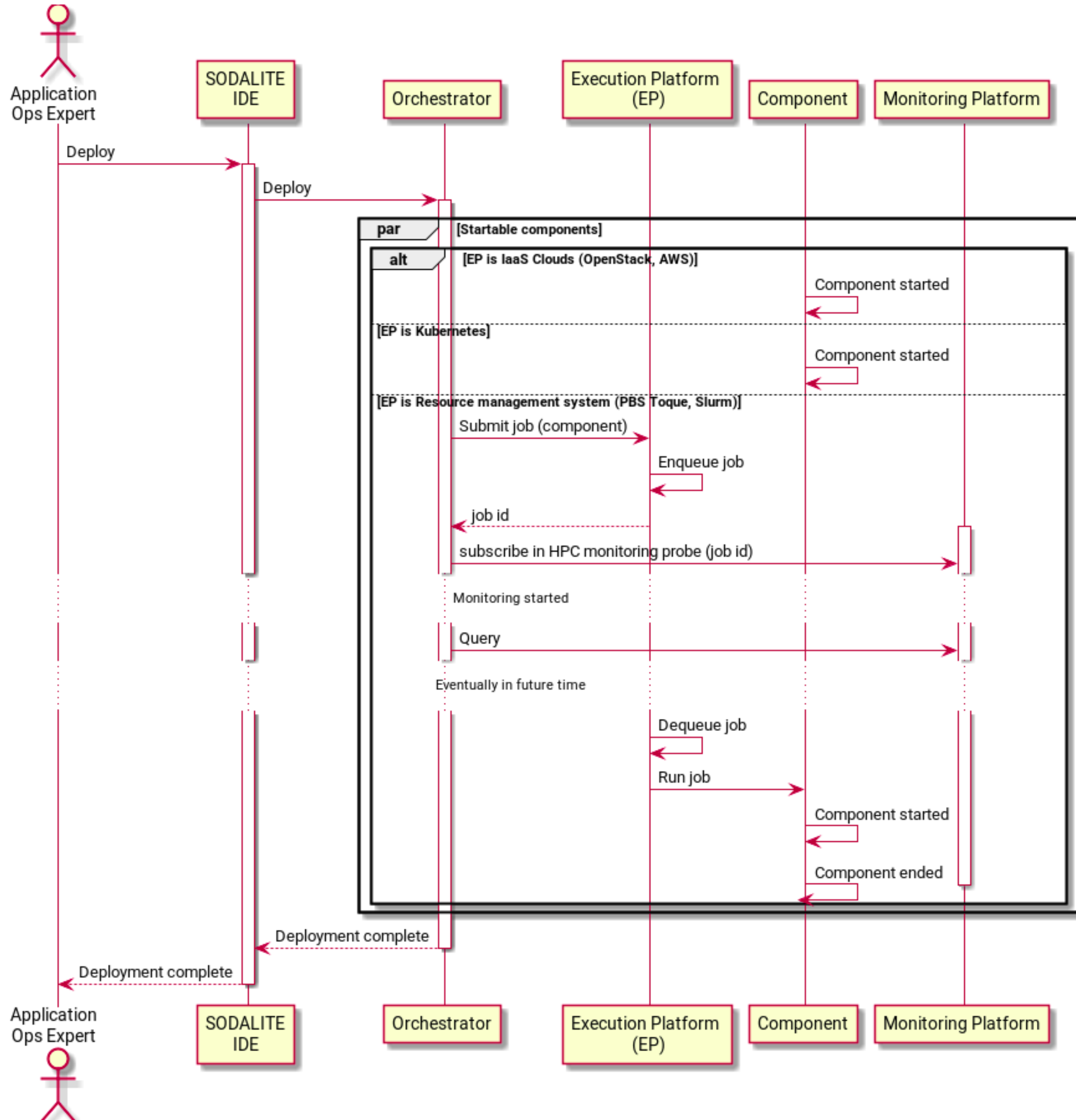

Figure 21 - Sequence diagram for UC7.

Figure 21 describes the interaction between the SODALITE components while implementing UC7 - Start Application. UC7 is accompanying UC6 by describing in detail the start of batch applications, i.e. applications that take an input, process it and give the results, unlike the services (e.g. web servers, REST APIs), which start right after deployment and run continuously. Furthermore, these applications can be deployed once and executed several times with different inputs.

When a job is submitted to the batch system, it does not start immediately, but it is firstly enqueued to specified or default queue. It starts, once the job's turn comes in the queue, and it leaves the queue for its execution. It may take some time depending on the resources' availability in the queue.

During the deployment (or redeployment), when the resource management system is selected as an execution platform, the Orchestrator submits a job and monitors its state, whether it is running or finished. When the job is finished, the Orchestrator determines whether the execution was

© Copyright Beneficiaries of the SODALITE Project

successful or failed. The deployment terminates when the execution is failed, otherwise the deployment continues with the next application component. In this case of HPC scheduling, the precise start time for job execution (after being dequeued) is unpredictable, as it depends on the eventual availability of the requested resources. Therefore, job monitoring must start at queue time, as it is requested by the orchestrator to the Monitoring System, which, at this point, starts collecting statistics describing the application job status.

From then on, the Orchestrator can initiate the collection of metrics for its purposes (e.g., to check application job health).

4.3.2.3 UC8: Monitor Runtime

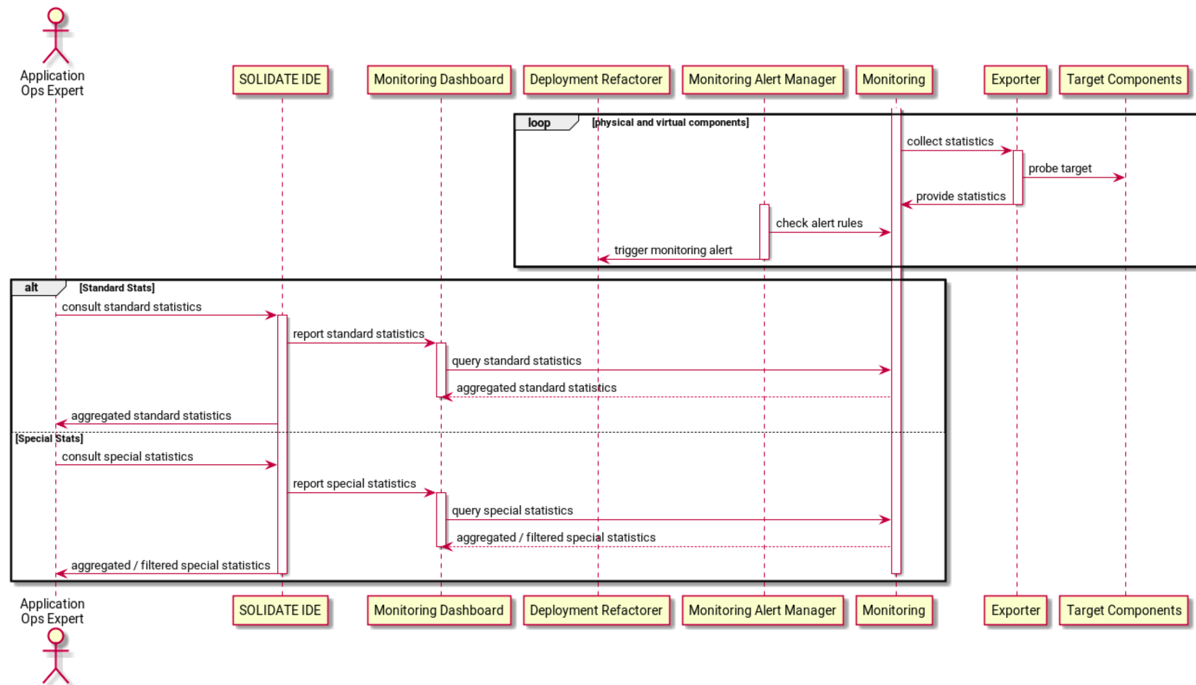

Figure 22 - Sequence diagram for UC8.

Figure 22 describes the interaction between the SODALITE components while implementing UC8 - Monitor Runtime. The Monitor component collects system statistics on an ongoing basis. On each host (whether physical or virtual) there are one or more exporters (i.e. probes) that interact with the Monitoring component and report back to it some standard (but also specialized) statistics about their target, which could be either the execution platform or the application component. Statistics are usually collected on each target by reading various counters and registers that hold updated system statistics.

Periodically, Monitoring collects the statistics from all the registered exporters and stores them into its internal database for further inspection, upon the reception of queries requested from external clients. High level reports on standard statistics are available to be consulted by AOEs in the Monitoring Dashboard that is accessible from the SODALITE IDE.

Some monitoring alerts could also be triggered by the Monitoring Alert Manager to the Deployment Refactor, for those situations where a condition (defined within a dynamic alerting rule) holds on concrete monitoring stats.

In such cases, the Deployment Refactor component could make placement decisions based on the resource usage. In other cases, it may be desirable to inspect some specific non-standard monitoring figures in order to isolate the cause of some observed anomaly. In this case, the AOE can request such a report by using the Monitoring Dashboard component, which, in turn, creates and issues to

© Copyright Beneficiaries of the SODALITE Project

the Monitoring component all the queries required to create the report. Using the results to those queries returned by Monitoring, the dashboard presents the aggregated report to the AOE.

4.3.2.4 UC9: Identify Refactoring Options

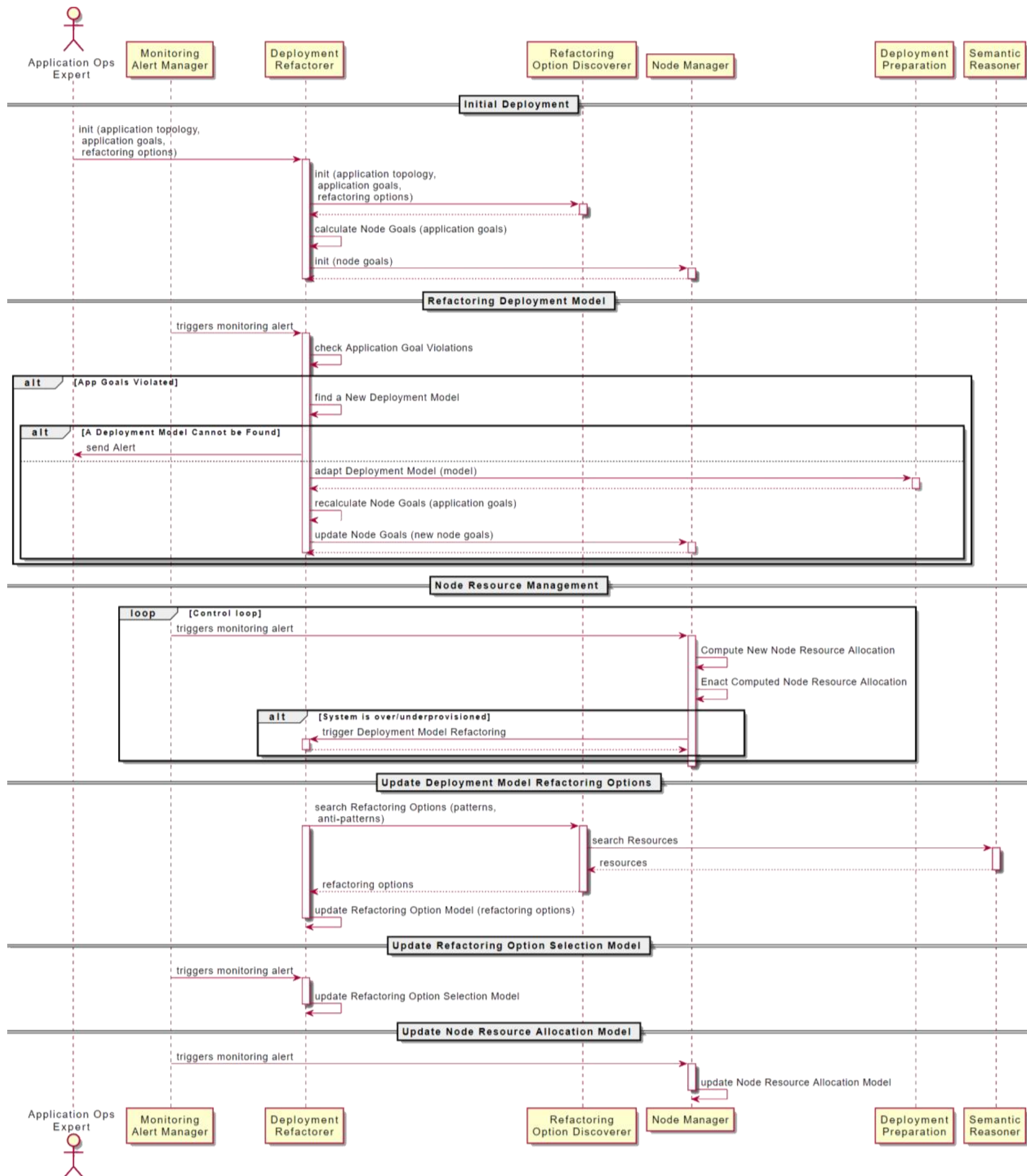

Figure 23 - Sequence diagram for UC9.

Figure 23 describes the interaction between the SODALITE components while implementing UC9 - Identify Refactoring Options. The Deployment Refactorer is initialized with the IaC models for the initial deployment, the initial set of refactoring options, and application goals. It uses the Node Managers of each of the nodes in the application topology to manage the resources in those nodes. The node resource management is based on the node level goals derived from the application goals. Via the Monitoring Alert Manager, the Deployment Refactorer is eventually notified when any of the

© Copyright Beneficiaries of the SODALITE Project

defined application goals are violated. Upon such circumstances, the Deployment Refactorer tries to find a new deployment model for the application that can resolve the detected application goal violations.

If a new deployment model cannot be found, the Application Ops Expert is alerted. The new deployment is enacted via the Deployment Preparation component. The Deployment Refactorer also may reassign node-level goals as necessary. The Refactoring Option Discoverer can find the new refactoring options as well as the changes to the existing refactoring options using patterns and anti-patterns (bugs). It uses the Semantic Reasoner for this purpose. Both the Deployment Refactorer and the Node Manager use the Monitoring Agent to collect data to determine the impacts of the refactoring decisions and to update the predictive models used for refactoring option selection and node resource allocation, respectively.

4.3.2.5 UC10: Execute Partial Redeployment

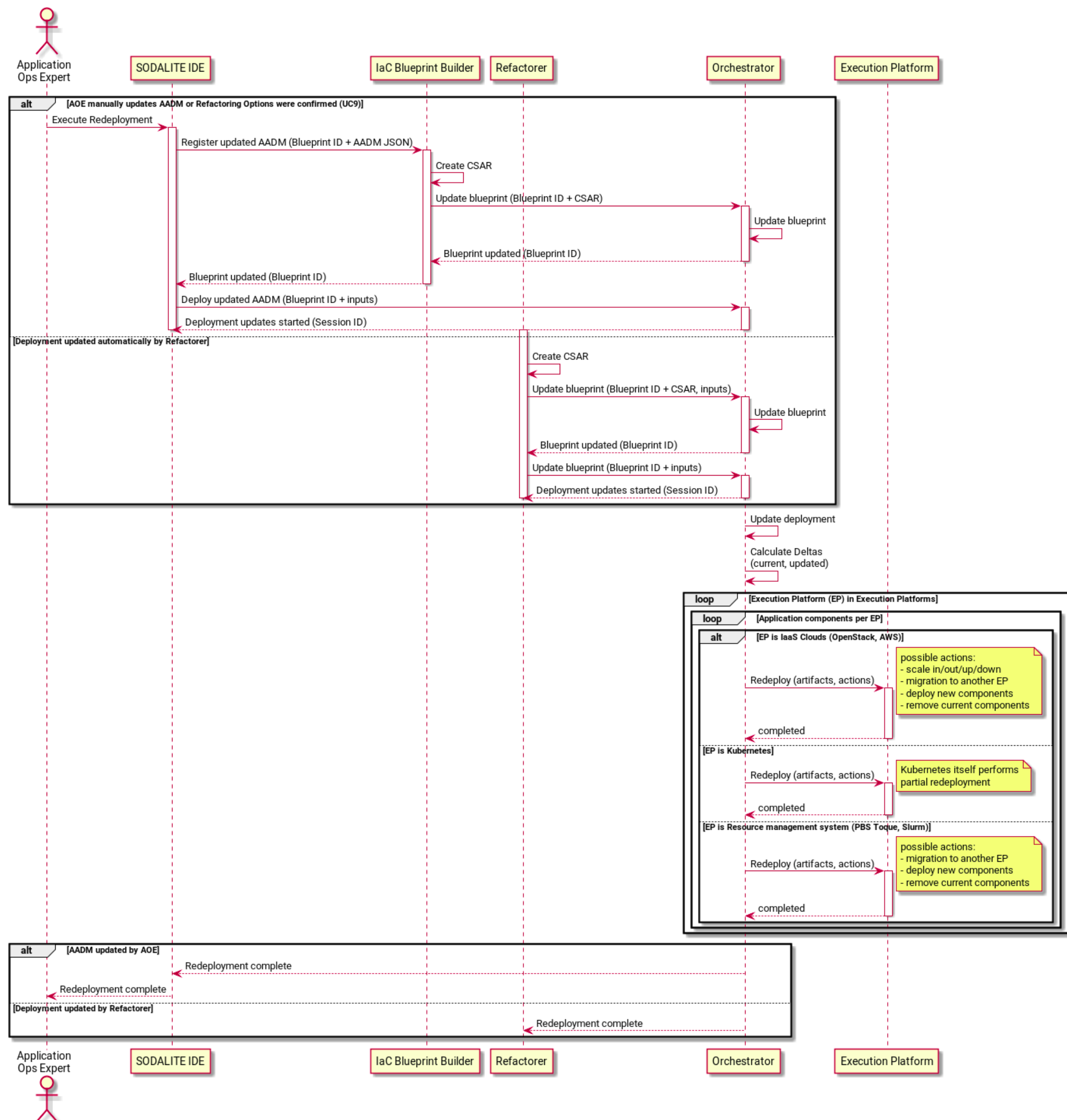

Figure 24 - Sequence diagram for UC10.

© Copyright Beneficiaries of the SODALITE Project

Figure 24 describes the interaction between the SODALITE components while implementing UC10 - Execute Partial Redeployment: SODALITE tries to apply partial redeployment through changes in TOSCA blueprints on any infrastructure. A redeployment (or deployment updates) is an act of modifying application topology or application parameters at runtime. The partial redeployment refers to updates of only affected components of application, i.e. those components that require modifications during the runtime of application as opposed to the modification of the whole application topology. As an example, a certain application component needs the updates of the container image or it requires scaling to improve the performance. Another example might be the changes in application topologies, when new components are introduced or old components need to be removed.

In SODALITE, partial redeployment can be triggered via the SODALITE IDE when the Application Ops Expert manually changes a AADM or he/she confirms one of the Refactoring Options suggested by Refactorer as described in UC9. Refactorer may also be configured to automatically request a redeployment if it detects any Refactoring Options at runtime. In both cases, a reference to a particular Blueprint ID, which is obtained after execution of UC6, should be used.

When a redeployment is triggered manually by the Application Ops Expert, the SODALITE IDE then requests IaC Blueprint Builder to register a new CSAR from the updated AADM and to redeploy the blueprint in the Orchestrator. After that, SODALITE IDE directly requests the Orchestrator to start the deployment updates and it receives a Session ID to monitor the redeployment progress.

Alternatively, when a redeployment is triggered automatically by Refactorer, it creates a CSAR, which reflects the updated deployment, and updates the blueprint in the Orchestrator. Similarly, it then directly requests the Orchestrator to start the deployment updates and receives Session ID to monitor the redeployment progress.

At this point, the deployment updates are executed by the Orchestrator. The Orchestrator derives the difference between current and updated deployments and applies adaptation actions until the current state of deployment becomes the updated state. Such adaptation actions are performed on the Execution Platforms used for the redeployment.

If the selected platform is IaaS Cloud or Kubernetes, the actions that might be performed are the following:

- any form of scaling (in/out/up/down),
- migration to another Execution Platform,
- deployment of the new application components introduced by the Application Ops Expert and removal of current components.

It should be noted that Kubernetes, being itself an orchestration platform, is enforcing partial redeployment.

For what concerns resource management systems (common in HPC), these Execution Platforms lack flexibility in scaling at runtime, hence the scaling actions are not present as possible adaptation actions; however, all the other actions can be executed as well (migration, deployment and removal of components).

4.4 Security Pillar

SODALITE provides tools and methods to authenticate and authorize actions on API endpoints using open-source Identity management and Secure Secret handling tools. While authorization is required - a single SODALITE endpoint can manage different infrastructures belonging to different domains. Apart from proper authentication and authorization of user actions, safe secret management across the whole deployment pipeline is also required and ensured by SODALITE.

Sample IAM workflow is shown in Figure 25.

© Copyright Beneficiaries of the SODALITE Project

Figure 25 - Sample IAM authentication workflow.

4.4.1 Security Pillar Toolkit

As a basis for authorization the OAuth 2.0 protocol was chosen, which is the de-facto industry standard for authorization. As for IAM provider, SODALITE uses *Keycloak*[3] - a popular and widely used open source tool which simplifies the creation of secure services with minimal coding for authentication and authorization. It allows wide customization of options exceeding the needs of

© Copyright Beneficiaries of the SODALITE Project

SODALITE. Apart from the basic authentication mechanism provided by Keycloak, SODALITE could also support such features as 2-factor authentication and seamless integration with third party identity providers like Google or GitHub.

For issuing JWT (JSON Web Token) Access Token OAuth 2.0 Resource Owner Password Credentials Grant flow is used, since the whole flow is confined inside SODALITE there is no insecure exposure of credentials to the client. A JWT Access Token, once issued by Keycloak, is then used across the whole SODALITE workflow. Keycloak allows the creation of different clients for different components, e.g., one set of rules can be applied to users logins from IDE and others for Automation components like Deployment Refactorer. Each client has a client secret assigned that is provided to Keycloak upon token creation to validate Token Endpoint API calls. Once JWT token is issued it can be validated by Keycloak using 2 standard mechanisms:

- Introspection endpoint
- JSON Web Key Sets

Introspection mechanism provides a more secure way for validating tokens as the token can be revoked before expiration. It is up to component developers to choose the exact mechanism but token introspection is encouraged to be used as the default one.

Different Resource Models, Application Deployment Models may refer to different Project Domains, meaning that only a certain group of end users should be granted access to these resources. To ensure that mechanism Client roles and Groups are used in Keycloak. Each role provides access to a certain project domain. Client roles are stored as JWT claims signed by Keycloak client private key, Groups are used to facilitate user management between different Domains. One user can belong to many Project Domains and thus have any number of client roles assigned to him.

Apart from properly authoring users actions other concerns are also addressed - properly handling infrastructure secrets, like RSA keys, tokens, passwords. This involves 2 points to be addressed:

- Security of data in use
- Security of data at rest

First one is mitigated by properly handling the secrets across the whole pipeline: not storing unencrypted information, no logging for security critical parts, proper user management on virtual containers that host SODALITE components. While SODALITE allows not storing any secrets at all and providing them in inputs, storing secrets in a vault allows to automate workflow and additionally ensure its safety. For that purpose *Hashicorp Vault*[4] was chosen, which is probably the most widely used open source tool for secret management. This approach would allow SODALITE operators to integrate it with their own Vault installations that are not part of SODALITE stack.

Hashicorp Vault configuration matches the one of Keycloak - roles have one to one correspondence, such that policies ensure access to secret management and each project domain has its own secret storage. Upon accessing a secret any SODALITE component must provide a client role and a valid access token issued by Keycloak. This token is validated by Vault using Keycloak and a short lived token is issued by Vault itself and returned. A role must be configured to have access to this project secrets. Then this token along with the secret address is used to retrieve a secret.

Both Keycloak and Hashicorp Vault are deployed as a part of SODALITE stack. Configuration of the components is done on the fly, basically these two components are ready to use after deployment. Admin credentials, roles, groups, clients, policies are created automatically, additional configuration can be done via API calls or component Web UIs.

5. KPIs and evaluation plan

This section summarises the work done on KPIs, on their evaluation, and on quality metrics used to probe and assess the artifacts developed by the project. Besides restating the original KPIs, we include the definition of the processes that will lead to the evaluation of SODALITE based on

© Copyright Beneficiaries of the SODALITE Project

foreseen measurements. The actual results obtained through this analysis will be presented in deliverable D6.3.

5.1 Technical KPIs

For the sake of completeness and simplicity, the following table lists all the technical KPIs proposed in the description of work.

Id	Description
KPI 1.1	Abstraction of application and infrastructure.
KPI 1.2	Abstraction of Infrastructure Performance Patterns.
KPI 1.3	Abstraction of execution constraints and possibilities.
KPI 2.1	Increase of abstracted application performance on abstracted infrastructure by using Infrastructure performance abstraction patterns
KPI 2.2	Increase of concretized (deployed) application performance running on targeted infrastructure through Predictive Deployment Refactoring.
KPI 3.1	Reduction in software and/or application development time and cost.
KPI 3.2	Reduction in software management (redeployment, reconfiguration) time and cost.
KPI 4.1	Component compatibility
KPI 5.1	Open source release
KPI 5.2	Extension of existing projects

5.2 Quality metrics associated with the SODALITE development process

As foreseen in the first release of this document, the consortium has adopted Sonar and sonarcloud.io[5] as our means for the continuous assessment of the quality of the code of developed assets. The tool, one of the most-widely used solutions in these years, easily integrates with the project's GitHub repositories and gives developers the ability to easily check code maintainability, reliability, and security in development branches and pull requests to allow early handling of issues in the development workflow. Sonar is installed as a plugin of Jenkins. After each git push it analyses the quality status of the updated project. Sonar was also enriched with plugins (such as Jacoco for Java and coverage.py for Python) that provide code coverage values (with respect to executed tests). More specifically, the tools highlight:

- Bugs, that is, critical errors that can lead to non-executable code;
- Vulnerabilities, that is, possible problems that may lead to unsecure code and thus security issues;
- Security hotspots, that is, code that must be checked manually to see whether there are security issues;
- Code smells, that is, possible problems in the code that indicate the need for further analyses and possibly some specific reactions to improve the overall organization/quality of the code;
- Coverage, that is, the percentage of code lines that is covered (executed) by the tests associated with the particular piece of code and run automatically in the build process through dedicated Jenkins pipelines (as described in deliverable D6.3).
- Code duplications, that is, clearly duplicated pieces of code that hamper the quality of the code and should be removed properly.

© Copyright Beneficiaries of the SODALITE Project

After the analysis, the data are pushed to sonarcloud.io that stores them and provides a public dashboard: https://sonarcloud.io/organizations/sodalite-eu/projects. The values currently measured are reported in deliverable D6.3.

5.3 Evaluation plan

This section presents the different solutions, processes, and tools we designed to assess the different KPIs presented in Section 5.1. Each table restates the original target measure and clarifies it, if needed. It identifies involved layers and used metrics. It then presents a baseline, if applicable, and the foreseen measurement process and machinery. The official deadlines, as in the description of work, and possible additional comments complete the table. The current data on the KPIs is provided as part of the evaluation in D6.3.

KPI 1.1

Original formulation	Abstraction of application and infrastructure
Target	Lower bound is 25% coverage of all application and infrastructure scenarios in the scope of SODALITE case-studies
Involved layer	SODALITE Modelling Layer
Clarifications	This KPI refers to the capability of the modelling layer to support the defined use cases in terms of abstract application and infrastructure structures
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	# resource models / # needed resource models # AADMs / # needed resource models # type definitions / # type definitions # components (templates) / # components (templates)
Baseline (A clearly defined baseline, initial numbers, and their sources)	Not applicable
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	The current numbers can be automatically derived by querying the KB for the respective constructs. The number of needed elements is given by the use case users
Official deadline (taken from DoW)	M24
Comments (any other important aspect)	

KPI 1.2

Original formulation	Abstraction of Infrastructure Performance Patterns
Target	Lower bound is 80% of all performance patterns found in HPC and Cloud infrastructures
Involved layer	SODALITE Modelling Layer

© Copyright Beneficiaries of the SODALITE Project

Clarifications	The KPI refers to performance patterns found in the demonstrating use cases. We use the use case requirements to map this and calculate the lower bound
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	% of use case performance requirements modelled in WP3
Baseline (A clearly defined baseline, initial numbers, and their sources)	None
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	No automated process. The collection is performed manually by analysing the requirements formulated by the owners of the case studies.
Official deadline (taken from DoW)	M33
Comments (any other important aspect)	

KPI 1.3

Original formulation	Abstraction of execution constraints and possibilities
Target	Lower bound is coverage of 80% of execution scenarios
Involved layer	SODALITE Modelling Layer
Clarifications	The KPI refers to execution constraints on computing, memory, network, and storage resources, and to the possibilities found in the demonstrating use cases. We use use case requirements to map this and calculate the lower bound
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	percentage of execution constraints and possibilities found in the demonstrating use cases that are modelled in WP3
Baseline (A clearly defined baseline, initial numbers, and their sources)	None
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	No automated process. The collection is performed manually by inspecting the requirements formulated by case study users and reported in D2.1 and in this document.
Official deadline (taken from DoW)	M33
Comments (any other important aspect)	

© Copyright Beneficiaries of the SODALITE Project

KPI 2.1

Original formulation	Increase of abstracted application performance on abstracted infrastructure by using Infrastructure performance abstraction patterns
Target	Application performance increased by 15%. The performance metric to be used will depend on the specific case study.
Involved layer	Case studies, SODALITE Modelling Layer, Infrastructure as Code Layer, Runtime, and Case studies
Clarifications	When AOEs exploit abstractions in their application code, we expect an increase in performance of 15%
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	Speedup measured based on application run time
Baseline (A clearly defined baseline, initial numbers, and their sources)	Baseline defined in d3.3
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	Measure the application run time with and without MODAK optimisation and calculate speedup
Official deadline (taken from DoW)	M30
Comments (any other important aspect)	Skyline extraction is evaluated on TensorFlow v1.1 using the resnet50 neural network model as a base. Resnet 50 is a convolutional neural network which is composed of 50 neural layers and incorporates skip connections that make bypassing layers possible.

KPI 2.2

Original formulation	Increase of concretized (deployed) application performance running on targeted infrastructure through Predictive Deployment Refactoring.
Target	Lower bound target is 20% improvement over the baseline
Involved layer	SODALITE Infrastructure as Code Layer, SODALITE Runtime
Clarifications	Applications that are run on a heterogeneous infrastructure require a dedicated resource management approach that takes into account the different computational power, functionalities, cost, and scaling and sharing capabilities of available executor devices (e.g., CPUs, GPUs). The two components in charge of resource management are Node Manager and

© Copyright Beneficiaries of the SODALITE Project

	Deployment Refactorer. The former optimizes existing resources through smart load balancing vertical scalability. The latter optimizes the deployment (e.g., resource migration and topology changes) and changes the goal of the Node Manager when needed.
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	- Management capabilities: list of management actions provided by the solution - SLA violations: number of times that a system violates the desired performance and security constraints, Accuracy and efficiency of predicting performance of alternative deployment variants. - Resources allocated: amount of resources allocated by the system (in cores*seconds) - Performance anomalies detected: accuracy of detecting performance anomalies, number of performance anomalies failures detected
Baseline (A clearly defined baseline, initial numbers, and their sources)	**Node Manager** - Kubernetes Horizontal Pod Autoscaler and Vertical Pod Autoscaler. - Rule-based approach (number of SLA violations) - Rule-based approach (total resources allocated) **Deployment Refactorer** The baseline is the statically optimized version of the application/case study (for a given workload).
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	**Node Manager** To evaluate the Node Manager and retrieve meaningful data for the aforementioned metrics, we deployed it in isolation on the Azure public cloud. We used a cluster of three virtual machines: one VM of type HB60rs with a CPU with 60 cores and 240GB of memory, and two VMs of type NV 6 equipped with a NVIDIA Tesla M60 GPU and a CPU with 6 cores and 56GB of memory. We also used an additional instance of type HB60rs for generating the client workload. The experiments exploited four existing ML applications: Skyline Extraction

© Copyright Beneficiaries of the SODALITE Project

	from the Snow UC, ResNet, GoogLeNet, and VGG16. We used differently-shaped, highly-varying synthetic workloads. Applications were run in different combinations concurrently on the servers. **Deployment Refactorer** Different deployment switching cases will be evaluated with SODALITE cases and benchmark applications (e.g., RUBiS and TeaStore).
Official deadline (taken from DoW)	M30
Comments (any other important aspect)	The evaluation of the Node Manager is based on the work presented in the technical report by Baresi et. al.[6].

KPI 3.1

Original formulation	Reduction in software and/or application development time and cost.
Target	Lower bound target is 10% improvement over the baseline and will be evaluated through external parties where possible. The improvement will be measured by considering the time needed to develop an application manually and then with SODALITE.
Involved layer	SODALITE Modelling layer
Clarifications	The focus will be on development of deployment descriptions, not application code.
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	Time needed to develop a complete blueprint. Corresponding cost considering the salary of involved people
Baseline (A clearly defined baseline, initial numbers, and their sources)	Manual development of blueprints for the three case studies. The exact baseline numbers are defined as part of the controlled experiment
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	We plan to run a controlled experiment to demonstrate reduction of time and cost against the baseline. The experiment will include three groups of participants: 1. non-expert developers (e.g., students), 2. use case owners 3. TOSCA experts All users will receive a tutorial concerning the SODALITE IDE. The ones who are not expert in TOSCA will also receive a tutorial about TOSCA. The request for all user groups will be to develop

© Copyright Beneficiaries of the SODALITE Project

	a blueprint both manually and using SODALITE. Groups 2 and 3 will focus on the development of the SODALITE use cases. Group 1 will be asked to focus on a simpler example. Each participant will keep track of the time dedicated to each task. We will also prepare a questionnaire to collect their subjective feedback on the experience. Moreover, we will check that the result of the work is correct, i.e., it executes correctly. Since repeating each task twice (without and with SODALITE) may introduce a bias, we will have different participants performing the task in a different order and we will compare the results we will obtain.
Official deadline (taken from DoW)	M24
Comments (any other important aspect)	Experiments described in Section 5.4

KPI 3.2

Original formulation	Reduction in software management (redeployment, reconfiguration) time and cost.
Target	Lower bound target is 30% improvement over the baseline and will be evaluated through external parties where possible. The improvement will be measured by changing the way we re-deploy the app.
Involved layer	SODALITE Runtime
Clarifications	This reduction specifically refers to resource management.
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	Time needed to redeploy and reconfigure an application. Also, time needed for configuring the management tools, e.g., monitoring. Corresponding cost considering the salary of the involved people
Baseline (A clearly defined baseline, initial numbers, and their sources)	Manual modification and redeployment of blueprints for the three case studies. The exact baseline numbers are defined as part of the controlled experiment
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	We plan to run a controlled experiment to demonstrate reduction of time and cost against the baseline. The experiment will include three groups of participants: 1. non-expert developers (e.g., students), 2. use case owners 3. TOSCA experts All users will receive a tutorial concerning the

© Copyright Beneficiaries of the SODALITE Project

	SODALITE IDE and SODALITE monitoring tool. The ones who are not expert in TOSCA will also receive a tutorial about TOSCA. The request for all user groups will be to modify a blueprint both manually and using SODALITE. The groups 2 and 3 will focus on the development of the SODALITE use cases. Group 1 will be asked to focus on a simpler example. Each participant will keep track of the time dedicated to each task. We will also prepare a questionnaire to collect their subjective feedback on the experience. Moreover, we will check that the result of the work is correct, i.e., it executes correctly. Since repeating each task twice (without and with SODALITE) may introduce a bias, we will have different participants performing the task in a different order and we will compare the results we will obtain.
Official deadline (taken from DoW)	M24
Comments (any other important aspect)	Experiments described in Section 5.4

KPI 4.1

Original formulation	Component compatibility
Target	The target is 95% of SODALITE component compatibility
Involved layer	All components
Clarifications	Integration of the SODALITE system allows for combined use of all its components.
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	Counting the number of components integrated in the SODALITE platform / total number of SODALITE components
Baseline (A clearly defined baseline, initial numbers, and their sources)	Not applicable
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	Will be assessed in terms of the # of components
Official deadline (taken from DoW)	M33
Comments (any other important aspect)	Components should provide - expected API usage/samples - requirements regarding usage - map expected callers

© Copyright Beneficiaries of the SODALITE Project

KPI 5.1

Original formulation	Open source release
Target	Minimum 80% of code released under open-source license
Involved layers	All components
Clarifications	
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	LOC released as open source / LOC produced by SODALITE to build the platform
Baseline (A clearly defined baseline, initial numbers, and their sources)	Not applicable
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	Counting the LOCs in the SODALITE repositories
Official deadline (taken from DoW)	M36
Comments (any other important aspect)	

KPI 5.2

Original formulation	Extension of existing projects
Target	Minimum 60% of code extending the existing projects, to be upstreamed
Involved layer	All components
Clarifications	The meaning is that given the subset of the SODALITE code that is built extending an existing open source project, 60% of this code is donated back to the existing project. To check the fulfilment of this KPI we need to identify the reference projects we extend.
Used metrics (A set of clearly defined metrics that will be used for checking the fulfilment of the KPI)	LOC developed by SODALITE for a component and donated to OS / LOC developed by SODALITE for the corresponding component
Baseline (A clearly defined baseline, initial numbers, and their sources)	Not applicable
Measurement process and machinery (how we plan to collect numbers and if the process is automated)	We collect information about the contributions to other projects and we compute the metric
Official deadline (taken from DoW)	M36
Comments (any other important aspect)	The list of contributed open-source projects is provided in deliverable D6.3.

© Copyright Beneficiaries of the SODALITE Project

5.4 Controlled experiments

Experiment with students and other external stakeholders

The purpose of this controlled experiment is to compare the time needed to define correct deployment code without and with SODALITE. Each experiment participant will go through the following steps:

1. (One hour) Training on the usage of the TOSCA language and on the usage of the SODALITE IDE (this is made available either as an eclipse plugin or as a dockerized component)
2. (Two hours) Development of two exercises. Half of the group will realize the two exercises (see below) in the order first A and then B, half will realize the exercises in the opposite order, first B and then A.
3. (15 mins) Questionnaire and interviews to discuss the results of the experiment. The purpose is to understand the following:
    a. How long each individual took to complete each of the tasks, including the two training activities (this could also be measured by someone observing the work).
    b. Which problems each individual encountered (did not understand the languages, could not use the tool, could not use the tool, could not save the file, could not generate the code, …)
    c. Another aspect to be checked, but this could be done by us, is whether the developed code actually works with xOpera.

Description of the Application used for the experiments

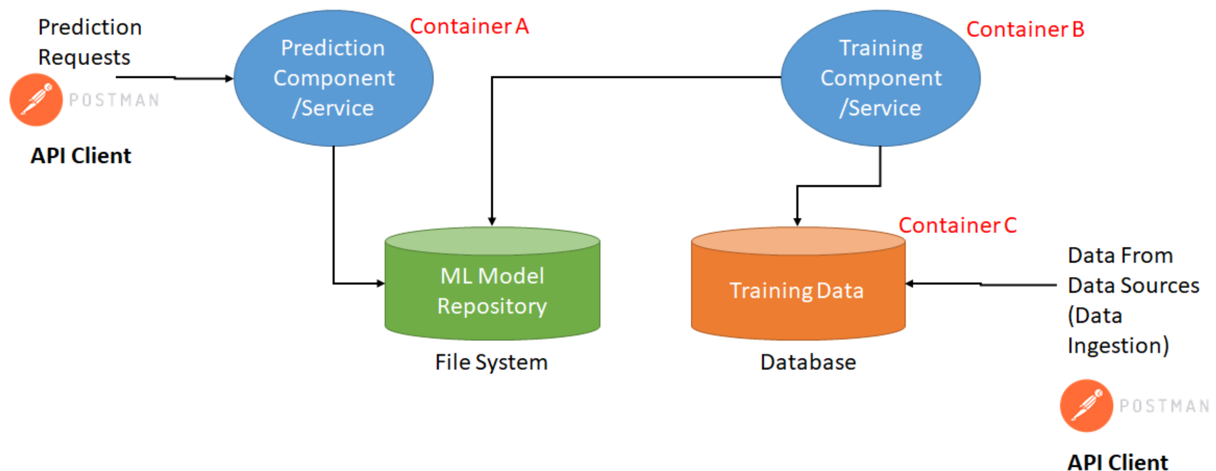

Figure 26 - Structure of ML exemplar application.

Figure 26 shows the structure of the distributed machine learning application. It consists of 1) a database that stores training data (as a RESTFul service), 2) a component that trains a machine learning model (as a RESTFul service) 3) a repository that stores trained machine learning models (File System or volume), 4) a component that makes predictions/inferences based on the trained models (a RESTFul service).

Exercise A - Development of a deployment blueprint in TOSCA

Develop the TOSCA blueprint needed for describing the above-mentioned machine learning application

© Copyright Beneficiaries of the SODALITE Project

Exercise B - Development of an AADM using the SODALITE IDE

Develop the Abstract Application Deployment Model needed for describing the above-mentioned machine learning application.

Experiment with case study owners

We give them the M18 version of the AADM for their cases and ask them to study them and extend them by adding the new aspects considered in their case study.

We could reuse the material developed for the first experiment to support training and collection of feedback.

In the second step, for the purpose of checking the achievement of KPI3.2, we will ask the case study owners to introduce a new change in their deployment and to modify the AADM accordingly.

Experiment with TOSCA experts

We give to the expert a TOSCA model and ask them to develop the corresponding AADM. They would need to take the SODALITE IDE tutorial. An ad hoc questionnaire should be defined for them. Also, we should decide with examples we should give to them as input.

In the second step, for the purpose of checking the achievement of KPI3.2, we will ask the TOSCA experts to modify the AADM to include monitoring aspects.

6. Conclusions and Future Work

This document presented an updated and self-contained version of the work done on the requirements: current status of year-one requirements, new use case, and new requirements the described architecture of the SODALITE environment identified the major changes we applied in the second phase of the project and also the new elements released since the first milestone. The architecture described here complies with Milestone MS6. The section on KPIs gives a detailed, improved description of each KPI, its scope, and of the evaluation workflow we defined and applied. The next and last version of this document will be presented at month 30.

© Copyright Beneficiaries of the SODALITE Project

7. References

1. https://docs.ansible.com/ansible/latest/collections/index_module.html
2. https://github.com/SODALITE-EU/iac-platform-stack
3. https://www.keycloak.org/ Open Source Identity and Access Management for Modern Applications and Services
4. https://www.vaultproject.io/ Secure, store and tightly control access to tokens, passwords, certificates, encryption keys for protecting secrets and other sensitive data using a UI, CLI, or HTTP API.
5. https://sonarcloud.io/
6. L. Baresi, A. Leva, G. Quattrocchi, and N. Rasi, Service Level Guarantees for Machine Learning Applications [Technical Report], 2020. Available at: http://hdl.handle.net/11311/1145281

© Copyright Beneficiaries of the SODALITE Project