Sodalite

SOftware Defined AppLication Infrastructures managemenT and Engineering

SODALITE Platform and Use Case Implementation Plan

D6.1

IBM

July 2019

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

Deliverable data			
Deliverable	SODALITE platform and use cases implementation plan		
Authors	Kalman Meth (IBM), Dimitris Liparas (USTUTT), Kamil Tokmakov (USTUTT), Ralf Schneider (USTUTT), Paul Mundt (ADPT), Román Sosa González (ATOS), Yosu Gorroñogoitia (ATOS), Javier Carnero (ATOS), Dragan Radolović (XLAB), Elisabetta Di Nitto (POLIMI), Piero Fraternali (POLIMI), Rocio Nahime Torres (POLIMI), Panos Mitzias (CERTH), Adrian Tate (CRAY), Karthee Sivalingam (CRAY)		
Reviewers	Daniel Vladušič (XLAB) Luciano Baresi (POLIMI)		
Dissemination level	Public		
History of changes	Kalman Meth, IBM Dimitris Liparas, USTUTT	Outline created	23 May 2019, outline
	All	Initial partner contributions	14 June 2019, Initial version
	All	Additional partner contributions	5 July 2019, Initial version
	All	Reactions to comments of first review	22 July 2019, reviewed
	Kalman Meth, IBM Dimitris Liparas, USTUTT	Final edits	30 July 2019, final

Acknowledgement

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

Table of Contents

Table of Figures

Table of Tables

Executive Summary

This deliverable presents the time plan for the development of the SODALITE platform, as well as the implementation plan of the project’s use cases. This document is delivered in parallel to deliverable D2.1 “Requirements, KPIs, evaluation plan and architecture - First version”, in which the architecture components and their interactions are described in detail. More specifically, D6.1 provides a description of the technologies that we expect to use for the implementation of the components constituting the SODALITE platform. Then the deliverable presents the implementation timeline of the HPC and Cloud testbeds that will be provided for the experimentation with the platform’s components. In addition, the document describes the provisioning and setup of the project’s development infrastructure, followed by a timeline, defining the foreseen iterations of the SODALITE platform. Finally, a description of the SODALITE use cases, along with their implementation plan, is provided.

We envision 3 iterations of delivery of the SODALITE platform, one in each year of the project. By the end of Year 1, we expect the initial implementation of the basic components making up the SODALITE platform. During Year 2 we expect to progress with integration of the components, more advanced features, and initial evaluation of the improvement provided by the SODALITE platform. In Year 3 we expect to iteratively measure the results produced by the SODALITE platform and to make ongoing additional improvements.

Glossary

Acronym	Explanation
ALPR	Automatic License-Plate Recognition
API	Application Program Interface
CI/CD	Continuous Integration/Continuous Delivery
CLI	Command-Line Interface
CRI	Container Runtime Interface
CT	Computer Tomography
CV	Computer Vision
DEM	Digital Elevation Model
DMI	Daily Median Image
DSL	Domain-Specific Language
EAR	Eye Aspect Ratio
ECG	Electrocardiogram
EMF	Eclipse Modelling Framework
EXIF	Exchangeable Image File Format
FOV	Field of View
GA	Grant Agreement
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit

HPC	High Performance Computing
HPVM	High Performance Virtual Machine
IaC	Infrastructure as Code
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
ITK	Insight Segmentation and Registration Toolkit
M2T	Model-to-Text
MCA	Marching Cubes Algorithm
MIGR	Mountain Image Geo-registration
ML	Machine Learning
MPI	Message Passing Interface
MTU	Maximum Transmission Unit
NIC	Network Interface Controller
OCI	Open Container Initiative
OCR	Optical Character Recognition
PERCLOS	Percentage of Eyelid Closure
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
SVC	Support Vector Classifier

SVM	Support Vector Machine
ToR	Top-of-Rack
UDJ	Universal Data Junction
UGI	User Generated Images
VIN	Vehicle Identification Number
VM	Virtual Machine
VTK	Visualization Toolkit
WP	Work Package

1 Introduction

The objective of this deliverable is to present the plan of the Consortium regarding the development of the SODALITE platform, as well as the implementation of the three SODALITE demonstrating use cases. To this end, this document provides a detailed description of the resources needed to achieve the components' functionality that will be developed within SODALITE and of the platform as a whole, as well as a report on the plans of each demonstrating use case, coupled with realistic and concise information about their practical implementation.

This document is delivered in parallel to deliverable D2.1 "Requirements, KPIs, evaluation plan and architecture - First version", in which the architecture components and their interactions are described in detail.

1.1 Structure of the Document

This deliverable is structured as follows:

- The remainder of the Introduction Section reviews the component structure of the SODALITE platform. This material is a highlight of what is presented in detail in the Architecture Section of deliverable D2.1 "Requirements, KPIs, evaluation plan and architecture - First version".
- Section 2 provides a description of the existing technologies that will be used for the development of the platform's components, as well as the setup and provisioning of the SODALITE infrastructure.
- Section 3 presents the SODALITE testbeds and development infrastructure and provides the overall timeline for the development of the SODALITE platform and the implementation of the components in Work Packages (WPs) 3 (Semantic Abstractions Design and Modelling), 4 (IaC Management) and 5 (Runtime Implementation).
- Section 4 presents the implementation plans for the three SODALITE demonstrating use cases and finally,
- Section 5 provides some concluding remarks.

It should be noted that while (based on the SODALITE GA (Grant Agreement)) this deliverable must also provide detailed specifications of the components' functionality that will compose the SODALITE platform, it was decided to include this information in deliverable D2.1 "Requirements, KPIs, evaluation plan and architecture - First version" (also due in M6 of the project, submitted together with D6.1) for a better presentation, since D2.1 provides the initial outline of the SODALITE architecture. Therefore, in D2.1, under Section 3 (Architecture), a detailed description of the SODALITE platform's envisioned components is provided, in terms of their functionality, dependencies, supporting technologies, as well as critical factors with respect to their implementation. In this document, we provide only a synopsis of the components that are more fully described in D2.1.

1.2 SODALITE Components

We reproduce here a synopsis of the SODALITE architecture that is described in D2.1. Please see the architecture Section (Section 3) in D2.1 for full details of the functional description, inputs, outputs, and dependencies of each component.

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

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

The SODALITE platform is divided into three main layers, each covered by a separate work package. These layers are the Modelling layer (WP3), the Infrastructure as Code layer (WP4), and the Runtime layer (WP5). Figure 1 below shows these layers together with their relationships.

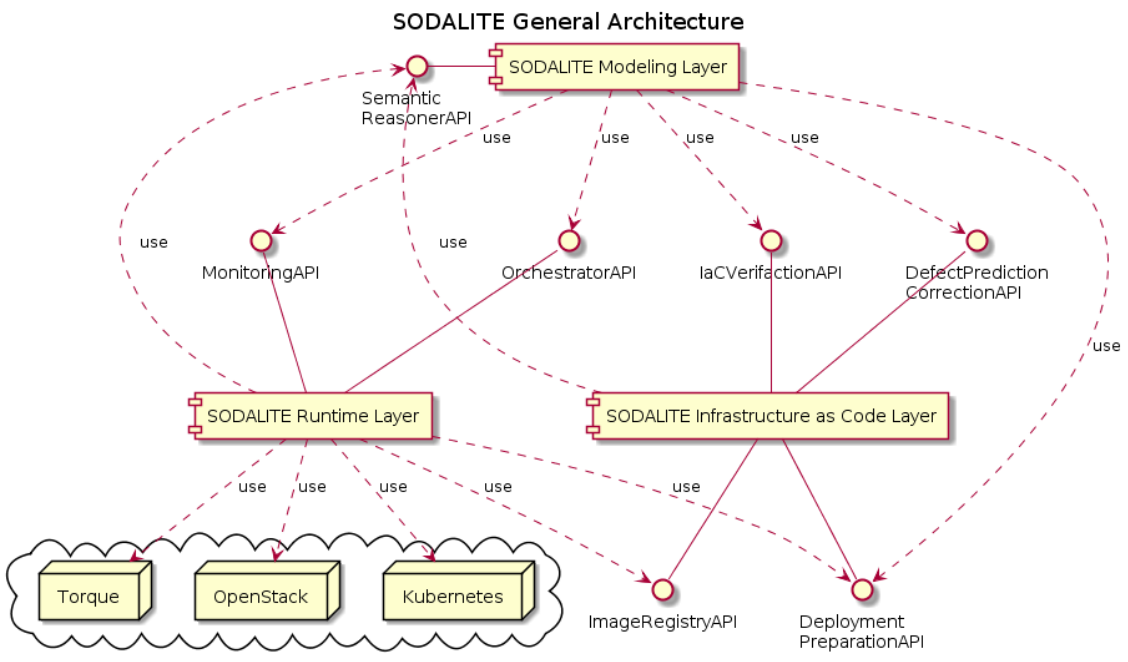


Figure 1: SODALITE overall Architecture

1.2.1 SODALITE Modelling Layer

The components of the SODALITE Modelling Layer are depicted in Figure 2.

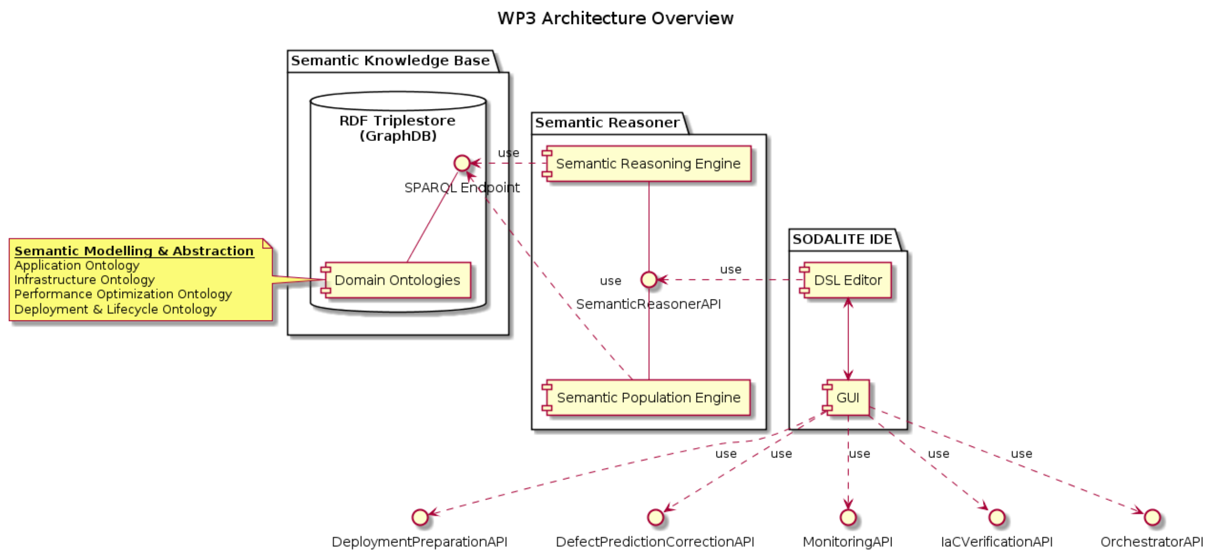


Figure 2: SODALITE modelling layer components (WP3)

The SODALITE IDE provides complete support for the authoring lifecycle of abstract application deployment models (see D2.1 for details). The Semantic Knowledge Base (KB) is SODALITE's semantic repository that hosts the models (ontologies) created in WP3. The Semantic Reasoner is a middleware facilitating the interaction with the KB. In particular, it provides an API to support the insertion and retrieval of knowledge to/from the KB, and the application of rule-based semantic reasoning over the data stored in the KB.

1.2.2 SODALITE Infrastructure as Code layer

The components of the SODALITE Infrastructure as Code (IaC) Layer are depicted in Figure 3.

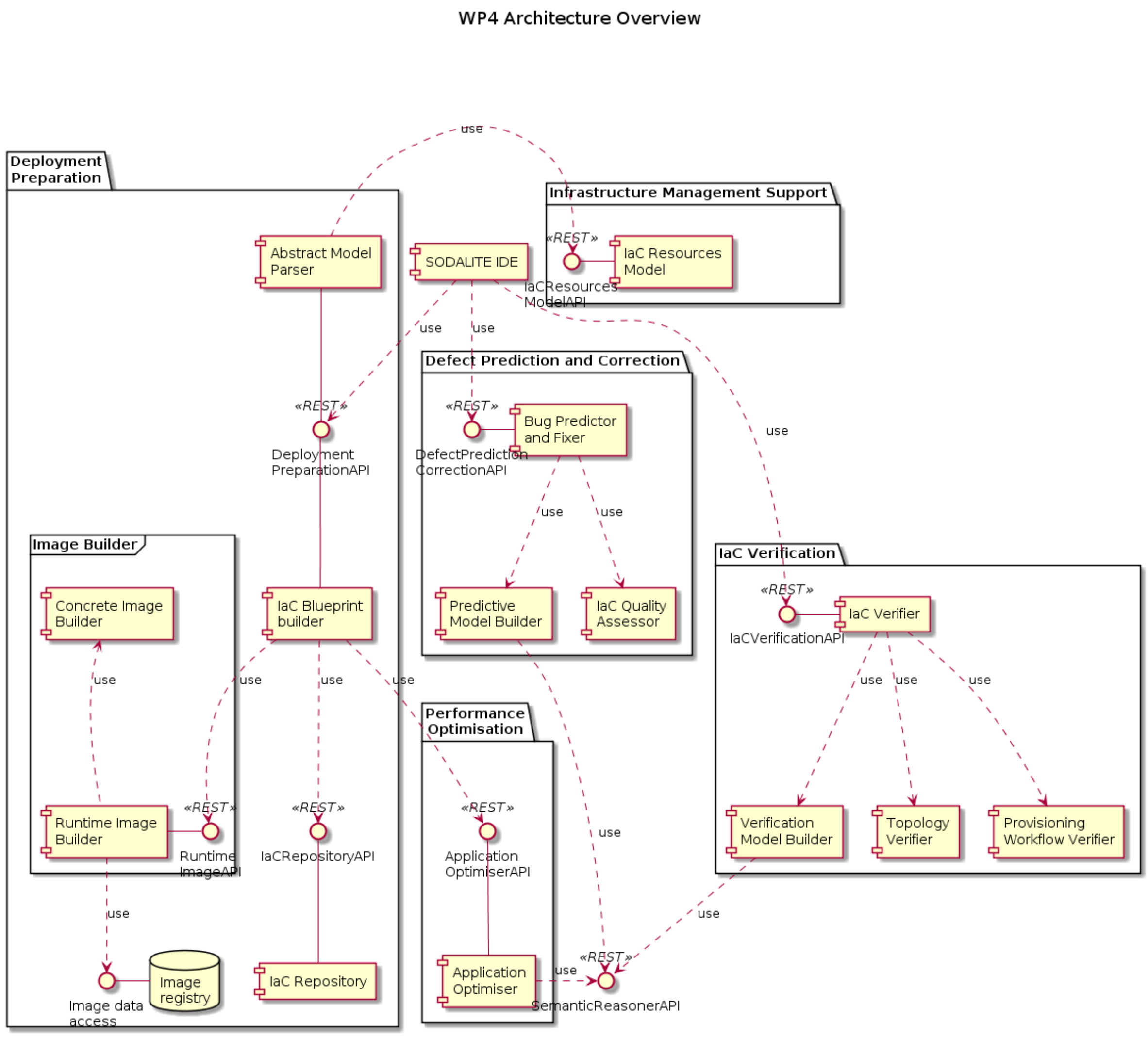


Figure 3: SODALITE infrastructure as code layer components (WP4)

The main task of the IaC layer is to take the modelling information provided by the SODALITE IDE (WP3) and produce an IaC blueprint. Deployment Preparation involves a number of operations to build an IaC blueprint. These operations are handled by sub-components depicted in Figure 3 and are detailed in deliverable D2.1. Additional components are envisioned to verify correctness of the provided model, to predict possible bugs in the provided model, and to optimise the application for a given target execution platform.

1.2.3 SODALITE Runtime layer

The components of the SODALITE Runtime Layer are depicted in Figure 4.

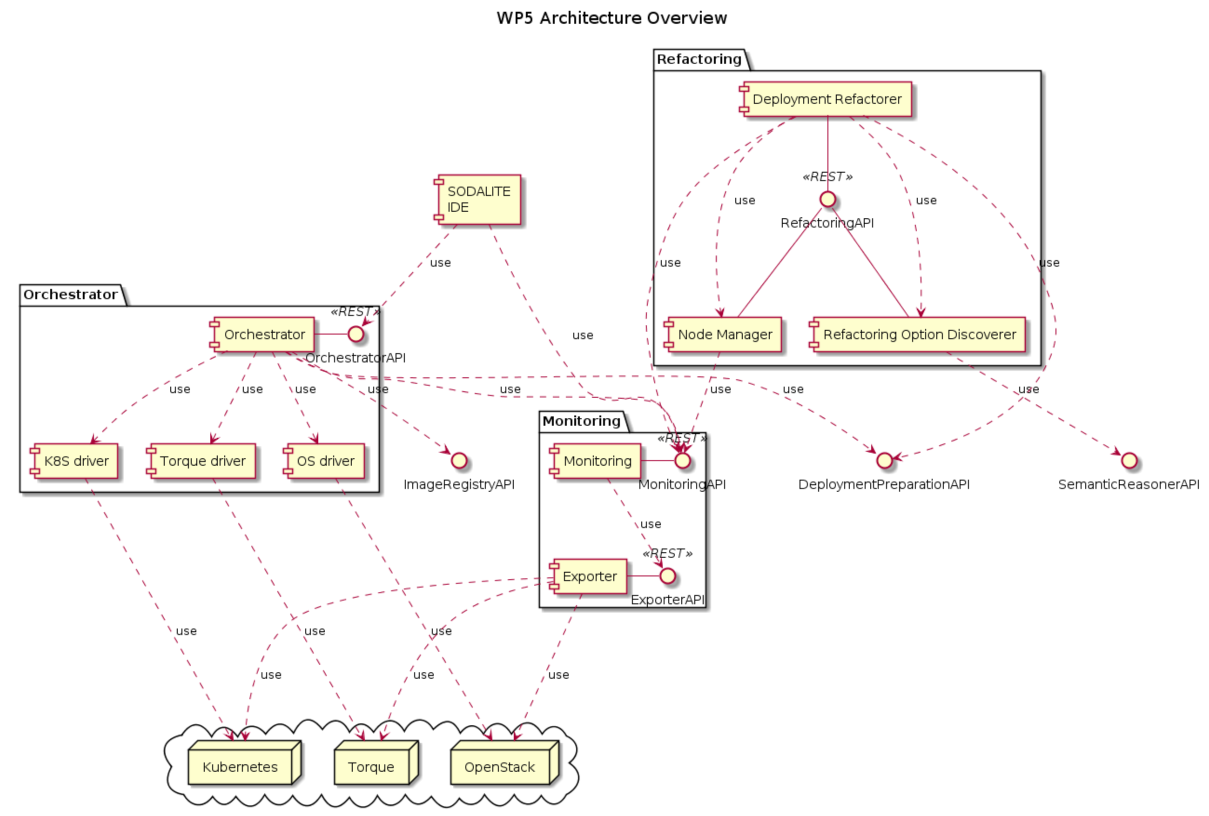


Figure 4: SODALITE runtime layer components (WP5)

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

1.3 Testing the SODALITE Stack

We plan to have 3 Demonstrating Use Cases to verify the SODALITE Platform: The POLIMI Snow use Case, The USTUTT Clinical Trial use Case, and the ADPT Vehicle IoT Use Case. Each of the use cases is expected to undergo several iterations of development and benchmarking using the SODALITE platform.

We envision 3 iterations of delivery of the SODALITE platform, one in each year of the project. By the end of Year 1, we expect the initial implementation of the basic components making up the SODALITE platform. During Year 2 we expect to progress with integration of the components, more advanced features, and initial evaluation of the improvement provided by the SODALITE platform. In Year 3 we expect to iteratively measure the results produced by the SODALITE platform and to make ongoing additional improvements.

2 Description of Technology Stack

As already explained in the Introduction Section, the currently envisaged components that make up the first iteration of the SODALITE platform are summarized above and described in detail in deliverable D2.1 under the Architecture Section. We describe in this Section technologies that are planned to be used to implement some of those components. We expect to augment these technologies with the necessary features that will be further required to implement the SODALITE solution. It should be noted that these technologies were selected based on the consortium partners' expertise, as well as the potential to further uptake the work in several tools/technologies that were developed as part of past European projects or initiatives, in which the consortium partners have been involved.

2.1 WP3 Technologies

WP3 is concerned with the semantic abstractions and the relevant design and modelling of applications and cloud infrastructures along with their performance characteristics and deployments. The main software components to support these are:

- The Semantic Knowledge Base - A semantic repository to accommodate SODALITE's knowledge in the domains of applications, infrastructure, performance optimisations, deployment and lifecycle, and more. This knowledge will be generated by multiple stakeholders (e.g. resource experts) and represented into RDF-based knowledge graphs (ontologies).
- The Semantic Reasoner- A dedicated middleware to interact with the Semantic Knowledge Base by importing/retrieving data, and applying complex, rule-based semantic reasoning. Thus, the Semantic Reasoner will expose an API to be accessible by other system components.
- The SODALITE IDE - A software component to provide complete support for the authoring of abstract application deployment models with the use of the SODALITE DSL. It will also enable the monitoring of each deployment's lifecycle, applied optimisations, etc.

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

2.1.1 Protégé

Protégé [1] is a free, open-source ontology editor that will be used for the creation of the project's domain ontologies, which will be the core of SODALITE's Semantic Knowledge Base. Protégé provides the necessary features for the definition of class hierarchies, datatype and object properties, axioms, etc., and the generation of all popular ontology file formats, like *owl* and *rdf*.

2.1.2 GraphDB

GraphDB [2] is a semantic graph database that acts as a SPARQL-served endpoint for ontologies. SODALITE's ontologies, created within Protégé, will be hosted by a GraphDB deployment, which will support the population of system data and the execution of rule-based semantic reasoning. This GraphDB deployment will act as SODALITE's Semantic Knowledge Base - repository.

2.1.3 SPARQL

SPARQL [3] is a query language for RDF (Resource Description Framework) data and ontologies. SPARQL queries will enable the insertion, update and retrieval of system data to/from the Semantic Knowledge Base. The semantic reasoning process will also be based on SPARQL queries. Thus, SPARQL will support a great part of the Semantic Reasoner functionality.

2.1.4 XText

XText [4] is an Eclipse [5] -based framework for specifying DSL (Domain-Specific Language) metamodels and textual edition of conforming model instances. It includes several components, namely a parser, linker, typechecker, compiler, as well as a textual editor for Eclipse. It is also compatible with any editor that supports the Language Server Protocol and your favourite web browser. DSL metamodels/models are EMF (Eclipse Modelling Framework)/Ecore-based. Therefore, it is compatible with EMF-based M2T (Model-to-Text) transformations tools, such as Xpand [6] or Acceleo [7] for DSL conversion (to SPARQL queries, for example).

Xtext also provides support for Web DSL edition, leveraging on existing Web editors such as Orion [8], Ace [9] or CodeMirror [10].

Around Xtext there are some related technologies. Concretely, DSLForge [11] provides an integrated Web IDE (Integrated Development Environment) Workbench for Xtext DSL editors, with a Project Explorer view and model persistence. Sirius [12] and Graphiti [13] provide a graphical DSL modelling framework for Eclipse. Using these latter technologies, users can define their DSL using graphical notation (in contrast to the textual notation available in XText). These technologies will be used by SODALITE IDE component.

2.2 WP4 Technologies

WP4 covers the aspects of IaC (Infrastructure as Code) within the SODALITE project. This includes verifying the validity of models provided as well as building and optimising of an IaC blueprint. Since TOSCA and actuation scripts are needed by the orchestrator to put to life an application deployment, we plan to build and use TOSCA IaC node repository and Ansible[16] actuation playbooks for building and preparation of the application deployment plan. As one of the most important of the WP4 goals is the optimal preparation of the deployment blueprint we plan to build components that optimise, verify and validate the IaC from the topology perspective before the execution and deployment itself. From the perspective of optimisation of the deployed application we plan to use CRAY's vast knowledge of application optimisation toolkit such as CRESTA, UDJ, Maestro etc., to fully optimise the application components before deployment. In the context of runtime environment, we decided to use the Docker virtualisation technology for deployments in the cloud execution platforms and considering a few container technology options for the deployment in the HPC environment (Singularity, CharlieCloud, Sarus). At the current stage of the project, the following technologies are being considered to be used for the WP4 developments.

2.2.1 CRESTA Autotuning framework

As part of the CRESTA [14] European project, a DSL-based autotuning framework was developed (initial implementation) by CRAY. This focuses on addressing the inherent complexity of the latest and future computer architectures. Autotuning is the process by which an application may be optimised for a target platform by making automated optimal choices of how the application is built and deployed. DSL that was developed exposes choices within an application for optimisation. This will be used as part of the Application Optimiser component.

2.2.2 Universal Data Junction

Universal Data Junction (UDJ) is a library-based transport that provides efficient communication of data between applications. It provides a capability to describe data that may be distributed and to communicate that data using put/get semantics. Distributed data (to multiple processes within an application) may be redistributed during transport. Various underlying (backend) transports are provided and may be selected at runtime. This will be used as part of the Application Optimiser component.

2.2.3 Maestro data orchestration middleware

Maestro data orchestration middleware [15] addresses ubiquitous problems of data movement in complex memory hierarchies and at many levels of the HPC (High Performance Computing) software stack. This middleware framework provides object-like data abstractions for management and reasoning about user data in applications and across workflows, with the ultimate goal of optimising data-movement across the memory-storage hierarchy. This will be used as part of the Application Optimiser component.

2.2.4 MAMBA - Managed Abstract Memory Arrays

A library-based programming model for C, C++ and Fortran based on Managed Abstract Memory Arrays, aiming to deliver simplified and efficient usage of diverse memory systems to application developers in a performance-portable way. MAMBA arrays exploit a unified memory interface to abstract memory from both traditional memory devices, accelerators and storage. This library aims to achieve good performance portability with an easy-to-use approach that requires minimal code intrusion. This will be used as part of the Application Optimiser component.

2.2.5 Ansible Actuation

XLAB provides initial node modelling through configurable Ansible [16] roles and playbooks as part of the *Infrastructure Management Support*, thus creating a repository of predefined actuation scripts used by the orchestrator to deploy, start and monitor application artefacts. A decision has to be made about whether Chef [17] is to be used as well. This technology may be used for creating the deployment artefact images by the SODALITE Deployment Preparation package and will be used by the SODALITE Orchestrator as a deployment actuation tool.

2.2.6 Runtime Container Images

The open sourcing of Docker [18] container technologies marked a new milestone in virtualisation. Simplicity of building up application environments, transportability, the ease of deployment and responsiveness are key benefits for choosing the deployment of containerized applications on private and public cloud infrastructures.

Most of the technologies and tools built around containers are well documented and open sourced with a very alive and vast community of developers and supporters, backed by industry leading IaaS (Infrastructure-as-a-Service) giants such as Amazon, Google, Microsoft and others.

As bringing orchestration to HPC and Cloud environments is one of the key goals of the SODALITE project, choosing the right container technology and tools for building up the runtime environment is an essential part of application design and deployment pipeline. At this point few HPC Container technologies are being considered:

- Singularity [19]
- CharlieCloud [20]
- SARUS [21]

The decision will be made based on benchmarking of the mentioned technologies. These technologies will be used in SODALITE Deployment preparation package and by SODALITE Orchestrator.

2.3 WP5 Technologies

WP5 deals with the SODALITE Runtime environment. The objectives of WP5 are the orchestration of deployments on heterogeneous infrastructures, the monitoring of the deployed applications and their adaptation and improvement in response to violations in the application goals. To achieve its objectives, WP5 relies on several existing technologies for deployment, orchestration and monitoring on HPC and Cloud, but extending them or adding new functionality through new components where necessary. The following technologies are being considered to be used for the WP5 developments.

2.3.1 xOpera

xOpera [22] is a lightweight orchestrator compliant with the TOSCA Simple Profile YAML v1.2[1].

It is currently available as a CLI (Command-Line Interface) tool, designed to be modular and extensible. xOpera uses Ansible playbooks as actuation scripts for TOSCA [23] node lifecycle and relationship configuration. Supports deployments to OpenStack through Ansible playbooks. The SODALITE Orchestrator may use xOpera as base orchestrator. In principle, base orchestrators should be interchangeable as long as they are TOSCA compliant.

2.3.2 Skydive

Skydive [24] is an open source real-time network topology and protocols analyser providing a comprehensive way of understanding what is happening in network infrastructure. Skydive captures all the interface metrics and stores them in a time series database. An administrator can start traffic capture allowing to monitor metrics for specific protocols between specified endpoints or according to topology specifications. All the metrics are available through a REST (Representational State Transfer) API (Application Program Interface). These metrics are to be consumed by the Monitoring component to evaluate whether an application is achieving its performance goals.

2.3.3 Prometheus

Prometheus [25] is a well-known monitoring technology that implements a time series database to store infrastructure metrics. It uses a pull model in which small servers called "exporters" are in charge of getting the raw metrics and send to the Prometheus server each time they are asked to. Grafana [26] can be used on top of it to visualize those metrics and do alerting over them. The main block of the Monitoring component is based on Prometheus.

2.3.4 Croupier / Cloudify

Cloudify [27] is a general-purpose orchestrator for the Cloud based on workflows driven by events. Its design approach enables wide flexibility, such as working on agent/agentless architectures or acting as a meta-orchestrator working with lower level schedulers/orchestrators. It provides a powerful plugin system that support working with a lot of cloud technologies like OpenStack, Kubernetes, Ansible, Puppet [28] and many more. Its DSL is TOSCA-based.

Croupier [29] is a Cloudify plugin that focuses on supporting HPC infrastructures in Cloudify, as well as the execution of batch jobs (jobs that have a concrete start and end point, as opposed to typical cloud applications like web servers). It supports HPCs based on Torque [30] and Slurm [31], and it is compatible with the other Cloudify plugins to allow execution of applications in hybrid HPC+Cloud infrastructures.

The SODALITE Orchestrator may use Cloudify as base orchestrator. In principle, base orchestrators should be interchangeable as long as they are TOSCA compliant.

[1] TOSCA Simple Profile YAML v1.2 is an OASIS standard for cloud native deployment and application orchestration

2.3.5 ALDE

ALDE [32] is a workload scheduling and application lifecycle manager for HPC applications. The objectives of ALDE are (i) compiling the source code and packetizing it for different heterogeneous architectures, and (ii) deploying the generated artefact into an HPC workload manager (only Slurm is supported at this time). It will be part of the drivers/plugins used by the Orchestrator to connect to the HPC workload managers.

2.4 WP6 Technologies

WP6 covers the testbed environment, integration of SODALITE components and implementation of the project's demonstrating use cases. We will utilize state-of-the-art technologies to achieve the envisioned outcome and results of this WP. In the following subsections, we describe the technologies that will be used in the deployment of HPC and Cloud testbeds for resource provisioning, as well as the creation of the project's development infrastructure. We will try to support these technologies with future extensions of supported platforms during the course of the project.

2.4.1 OpenStack

OpenStack [33] is an open-source software platform for provisioning compute resources as well as other resources (e.g. network, storage) following the IaaS deployment model of cloud computing. OpenStack comprises of the components that enable various services for the cloud users. For example, *Nova Compute* [34] provides virtual machines, whereas *Ironic* [35] provisions bare-metal nodes, and the services such as *Cinder* [36] and *Neutron* [37] provide block storage and tenant networking, respectively. The resources are managed by the users via CLI or REST API. In SODALITE, OpenStack brings a scalable and extendable solution regarding resource provisioning for the deployment of the project's demonstrating use cases, as well as for the experimentation with the SODALITE components that will be implemented during the course of the project.

2.4.2 Kubernetes

Kubernetes [38] is an open-source orchestrating system for deployment, management and scaling of containerized applications and services. An application container and its needed resources (e.g. network, storage) are encapsulated into a *Pod* [39], which is the basic execution and deployment unit in Kubernetes, and a *Service* [40] further groups multiple interrelated Pods together. Additionally, the pods and services can be labelled providing logical description of application deployment. As such, an application can be deployed with respect to different staging environments, e.g. development, test or production, isolated by the labels attached to each deployment.

Hence, such a high level of abstraction enables automated deployment and management of the containers done by the control plane, which schedules resources, provides an API for the pods and services and maintains the life cycle of the pods. Moreover, the control plane has declarative nature of deployment, meaning that it drives current state of deployment towards the desired state specified in the deployment description. Kubernetes provides both CLI and REST API interfaces for its management, as well as the Container Runtime Interface (CRI) that extends it with other container technologies, which are OCI-compliant, such as Docker and Singularity

SODALITE will utilize the Kubernetes features, such as automated orchestration of application deployment and labelling to deploy and distinguish the testing and production environments of the SODALITE components. Additionally, the Cloud components of the demonstrating use cases can be containerized using the container technologies presented in Section 2.2.6 and deployed in the Kubernetes cluster.

2.4.3 Torque

Torque [41] is a resource manager providing a low-level functionality to control and monitor the batch jobs and compute resources. The jobs can be parameterized in submit scripts, defining e.g. number of compute nodes/processors and execution environment, and scheduled by the workload managers, such as Moab [42] or Maui [43]. The resource manager then deploys and runs the jobs on the compute nodes with the start-up (prologue) and clean-up (epilogue) phases. A job status can be monitored, and in order to deal with high demand of compute resources, a job queue can be introduced. Torque will be used in SODALITE as a bare-metal resource manager and provisioner, providing compute resources for the execution of the jobs submitted as part of the demonstrating use cases' workflows.

2.4.4 vTorque

vTorque [44] is an open-source extension of Torque resource manager, developed by USTUTT for the purposes of the MIKELANGELO [45] H2020 European project, which introduces virtualization capabilities in the HPC infrastructure. Due to its non-invasive nature, it is independent of the version of Torque. It deploys and executes jobs in virtual machines, transparently created in the prologue phase of the job, and the jobs can be further parameterized with additional arguments related to virtual resources, e.g. number of vCPUs. In this way, vTorque enables cloud-like functionality in HPC. In SODALITE, we plan to extend vTorque in the direction of HPC job containerization, further supporting the developments that will take place in WP4 and WP5 with respect to the technologies on runtime container engines and images.

2.4.5 Jenkins

Jenkins [46] is an open-source automation server that allows one to automate the software development process with continuous integration and facilitating continuous delivery. This is the tool we plan to use for fast and automated building, testing, integration and packaging of the components that will be developed in SODALITE.

2.5 Summary

In this Section, we listed a number of existing technologies upon which to start the development of the SODALITE platform. For each technology, we identified the SODALITE component to which it is relevant. Table 1 summarizes the technologies, the supported SODALITE component, and the corresponding contributing partners.

Technology name	Use in SODALITE	Contributing / coordinating partner
Protégé	Semantic Knowledge Base (WP3)	CERTH
GraphDB	Semantic Knowledge Base (WP3)	CERTH
SPARQL	Semantic Knowledge Base (WP3)	CERTH
XText	SODALITE IDE (WP3)	ATOS
CRESTA	Application Optimiser (WP4)	CRAY
Universal Data Junction	Application Optimiser (WP4)	CRAY
Maestro	Application Optimiser (WP4)	CRAY
MAMBA	Application Optimiser (WP4)	CRAY
Ansible Actuation	Deployment Preparation (WP4)	XLAB
Runtime Container Images	Deployment Preparation (WP4)	XLAB
Prometheus	Monitoring (WP5)	ATOS
Skydive	Monitoring agent (WP5)	IBM
xOpera	Orchestrator (WP5)	XLAB
Croupier / Cloudify	Orchestrator (WP5)	ATOS
ALDE	Orchestrator plugins (WP5)	ATOS
OpenStack	Cloud Testbed (WP6)	ATOS
Kubernetes	Cloud Testbed (WP6)	ATOS
Torque	HPC Testbed (WP6)	USTUTT
vTorque	HPC Testbed (WP6)	USTUTT
Jenkins	CI/CD (WP6)	IBM

Table 1: Summary of existing technologies to use for SODALITE components

It is expected that we will need to build connectors and wrappers around some of these technologies to interact with the rest of the system. As development progresses, we expect to extend the functionality of some of these technologies and to fill in the gaps needed to implement all the components of the SODALITE platform.

3 Development Environment

In order to facilitate the development of SODALITE platform and its components, we introduce the project's development environment, which includes the SODALITE repository, CI/CD pipeline and an execution environment for running the SODALITE components. For these, we will provide Cloud and HPC testbeds that will provision virtual and bare-metal compute resources using technologies presented in Section 2.4.

In the following Sections, we describe the Cloud and HPC testbeds (Section 3.1) and their interactions with the SODALITE components (Section 3.2). Section 3.3 presents the development flow of SODALITE, whereas Section 3.4 outlines the development timeline of the overall SODALITE solution.

3.1 Cloud and HPC Testbed descriptions

Cloud and HPC testbeds, presented in Figure 5, will be deployed for the development and experimentation with the SODALITE components and will be provided and maintained by ATOS and USTUTT, respectively. Along with the testbeds, the SODALITE repository will be provided to store the source code of the components. The SODALITE testbeds and repository will be connected via the Internet in a secure way.

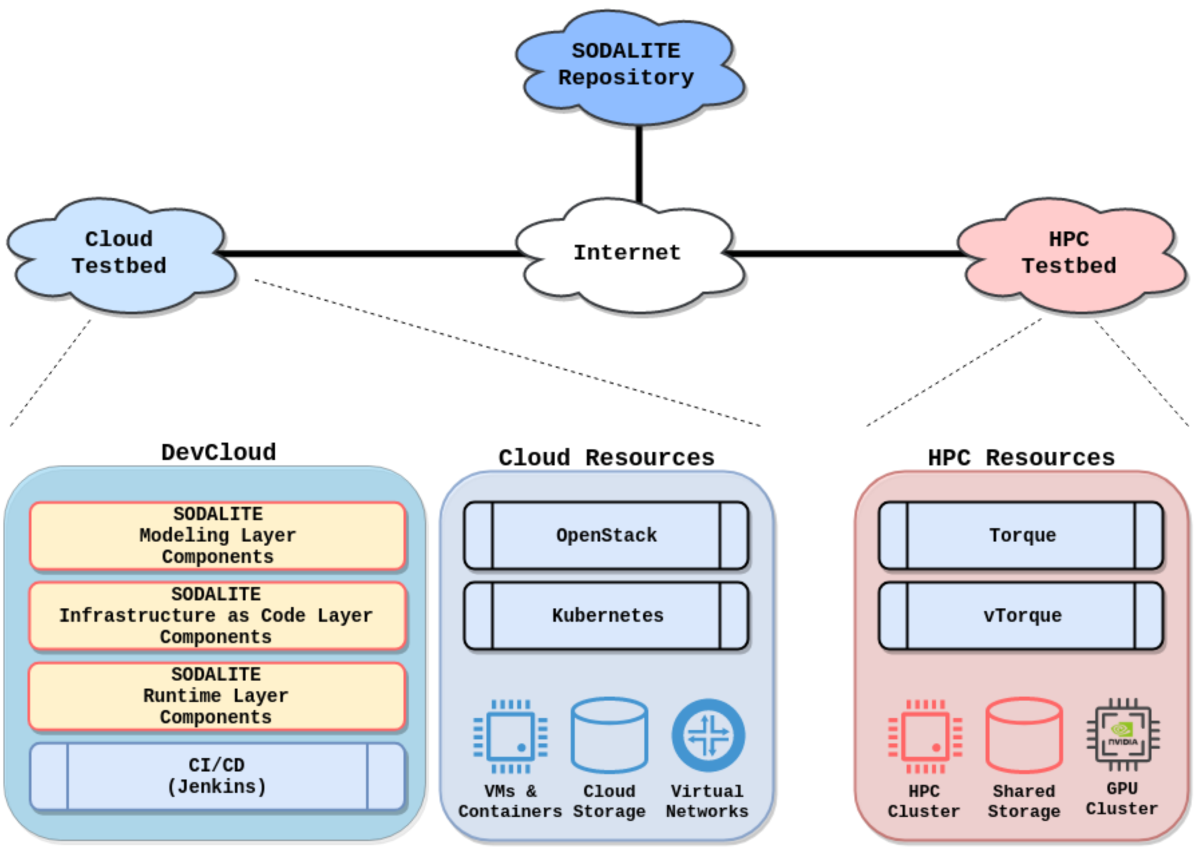


Figure 5: Cloud and HPC testbeds overview

The purpose of the Cloud testbed is to provide Cloud Resources, such as VMs (virtual machines), containers, cloud storage and virtual networks, for the application deployment of the demonstrating use cases. These resources will be managed by OpenStack and Kubernetes systems. Moreover, the Cloud testbed hosts the development environment (DevCloud), where the SODALITE components, described in Section 1.2, will reside for development and usage. In order to ensure the integrity and validity of the developed SODALITE components, a CI/CD (Continuous Integration/Continuous Delivery) approach will be adopted with the help of the Jenkins open source automation server, which will be responsible for running CI/CD tasks enabling fast and automated building, testing, integration and packaging of the SODALITE components.

The HPC testbed, on the other hand, provides a batch job system and bare-metal resources managed by a Torque resource manager extended with vTorque. At the initial stage, the resources such as bare-metal compute nodes forming an HPC cluster and sharing a storage pool, as well as GPU-enabled compute nodes, will be included in the HPC testbed.

In the following subsections, we describe in detail the specifications of both testbeds.

3.1.1 Cloud Testbed Specifications

The ATOS testbed consists of 3 nodes: 2 compute nodes and 1 storage node interconnected via a switch on each NIC (Network Interface Controller). The physical characteristics of the nodes and switches are presented below in Tables 2-4:

Compute nodes	
Number of nodes	2
CPU type	Intel Xeon E5-2670 0, 8-Core, 2.60GHz, HT (16 threads), 20 MB Cache, 8.0 GT/s QPI
Number of CPUs (Number of cores)	● Per node: 2 (2x8=16 cores) ● Total: 32 (16x2=32 cores)
Memory	● Type: DDR3 ● Amount per node: 16x4GB=64GB ● Total: 2x64GB=128GB
Internal storage	● Type: HDD (SATA) ● Size per node: 2x6TB=12TB ● RAID support: no
Network card	2x Intel® GbE I350 with PCI Express V2.1 (5 GT/s) Support

Table 2: Specifications of the computes nodes in the Cloud testbed (ATOS)

Storage nodes	
Number of nodes	1
CPU type	Intel Xeon E5-2670 0, 8-Core, 2.60GHz, HT (16 threads), 20 MB Cache, 8.0 GT/s QPI

Number of CPUs (Number of cores)	● Per node: 2 (2x8=16 cores) ● Total: 16
Memory	● Type: DDR3 ● Amount per node: 16x4GB=64GB ● Total: 1x64GB=64GB
Internal storage	● Type: HDD (SATA) ● Size per node: 3x6TB=18TB ● RAID support: no
Network card	2x Intel® GbE I350 with PCI Express V2.1 (5 GT/s) Support

Table 3: Specifications of the storage nodes in the Cloud testbed (ATOS)

Interconnect (switches)	
Number of switches	1
Switch model	BROCADE ICX6450
Ports	24x 10/100/1000 Mbps RJ-45 ports
Number of switches	1
Switch model	BROCADE VDX6710
Ports	48 x 10/100/1000 + 6 x 10 Gigabit SFP+

Table 4: Specifications of the interconnect in the Cloud testbed (ATOS)

3.1.2 HPC Testbed Specifications

The testbed hosted in USTUTT consists of 9 nodes: 8 compute nodes and 1 storage node interconnected with a ToR (Top-of-Rack) switch. The physical characteristics of the nodes and switches are presented below in Tables 5-7:

Compute nodes	
Number of nodes	8

CPU type	Intel Xeon E5-2630v4, 10-Core, 2,20 GHz, HT, 25 MB Cache, 8,0 GT/s (Broadwell EP)
Number of CPUs (Number of cores)	● Per node: 2 (2x10=20 cores) ● Total: 16 (16x10=160 cores)
Memory	● Type: DDR4 ● Amount per node: 8x16GB=128GB ● Total: 8x128GB=1TB
GPU type	MSI GeForce GTX 1080 Ti Aero 11G OC, 3584 Cores, 11GB GDDR5X Memory
Number of GPUs	● Per node: 1 ● Total: 8
Internal storage	● Type: SSD (SATA) ● Size per node: 2x1.92 TB=3.84TB ● RAID support: yes
Network card	Mellanox ConnectX-4 VP, dual-port FDR IB and 40 / 56 GbE, QSFP28

Table 5: Specifications of the computes nodes in the HPC testbed (USTUTT)

Storage Node	
Number of nodes	1
CPU type	Intel Xeon E5-2630v4, 10-Core, 2,20 GHz, HT, 25 MB Cache, 8,0 GT/s (Broadwell EP)
Number of CPUs (Number of cores)	2 (20 cores)
Memory	● Type: DDR4 ● Amount: 6x32GB=192GB
Internal storage	● Type: HDD (SATA) ● Size per node: 16x4.0TB=64TB ● RAID support: yes
Network card	Mellanox ConnectX-4 VP, dual-port FDR IB and 40/56 GbE, QSFP28

Table 6: Specifications of the storage nodes in the HPC testbed (USTUTT)

Interconnect (Top-of-rack switch)	
Number of switches	1
Switch model	Mellanox Spectrum SN2100
Ports	16xQSFP28 ports, 40 GbE
Additional features	● VxLAN Hardware VTEP ● SDN: OpenFlow 1.3 ● Integration with VMware NSX & OpenStack

Table 7: Specifications of the interconnect in the HPC testbed (USTUTT)

3.2 SODALITE Components interaction with the testbeds

The SODALITE components, mainly the components of the Runtime Layer, must interact with the resource managers of the Cloud and HPC testbeds for resource provisioning, deployment, configuration, monitoring and refactoring of the application components developed by the providers of the demonstrating use cases with the use of the SODALITE platform. Figure 6 depicts a detailed view of the testbeds' setup and their interaction with the SODALITE components.

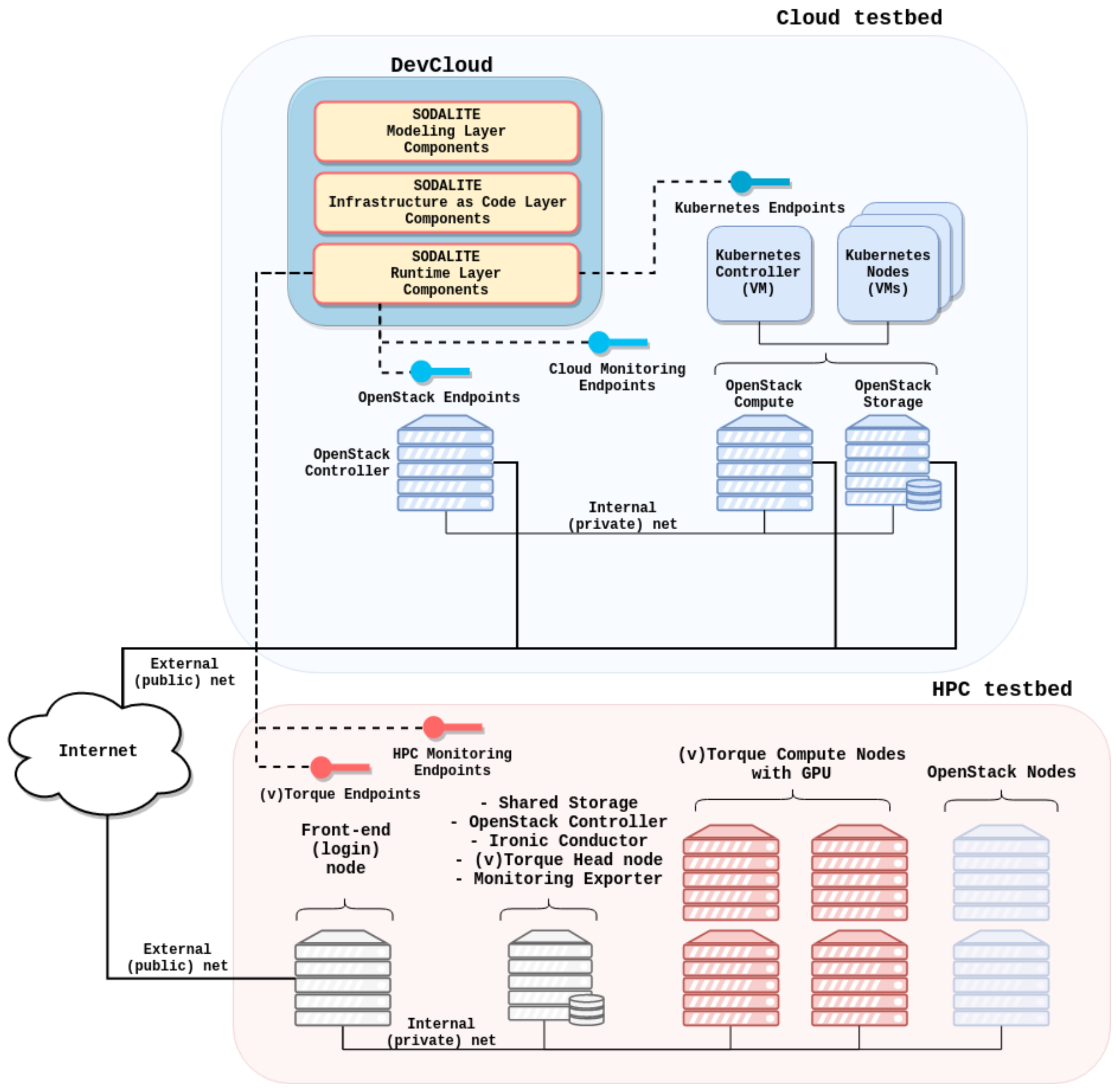


Figure 6: SODALITE HPC and Cloud testbeds

The development environment (DevCloud) for the deployment and integration of SODALITE components will reside on the Cloud testbed, which will be containerized in Kubernetes, such that the development and production versions of the developed components will be available. The DevCloud will be physically isolated from the resources available for the application deployment, due to possible interference (e.g. I/O interrupts) and contention of the resources, which will affect the performance of the applications. The CI/CD pipeline, although being part of the DevCloud, is not depicted in Figure 6 and will be presented in detail in the following Section 3.3 (Development flow description).

With respect to resource provisioning, it is planned to have OpenStack installed on the Cloud testbed, providing VMs via *Nova Compute* service, block storage via *Cinder* and networking via *Neutron* services. On top of these compute resources, the application components of the demonstrating use case providers will be deployed. It is also possible to deploy a Kubernetes cluster to further orchestrate the application deployment. As such, OpenStack will provision a set of VMs, one of which will act as a *Kubernetes Controller*, whereas the remaining VMs will be allocated as *Kubernetes Nodes*, i.e. the nodes running *Pods* and *Services*. The credentials to access both OpenStack and Kubernetes will be provided to the users of the Cloud testbed.

The HPC testbed will also be backed with OpenStack; however, *Ironic* service will be used. The rationale behind this is the flexibility and on-the-fly reconfiguration of the compute nodes on the physical infrastructure that Ironic brings, facilitating any future infrastructure modifications as the SODALITE project evolves. The Torque resource manager will be deployed providing HPC and GPU (Graphics Processing Unit) Resources for running application jobs. The credentials and workspaces will be created for the users of the HPC testbed.

The interaction between the testbeds will be established via the Internet: the Cloud testbed provides OpenStack and Kubernetes public endpoints for its management, while the Front-end node of the HPC testbed provides public ssh-based endpoints to access the Torque resource manager. Furthermore, both testbeds will provide mechanisms and endpoints to monitor various parameters from different layers of the whole SODALITE platform. The SODALITE Runtime Layer Components will communicate with the aforementioned management and monitoring endpoints in order to orchestrate the deployment, monitoring and refactoring of the application components of the demonstrating use cases.

3.3 Development flow description

The development of the SODALITE components will follow the CI/CD (Continuous Integration/Continuous Delivery) approach to allow fast and automated building, testing, integration and packaging of the components. Hence, the Jenkins open source automation server was chosen as the integration tool for SODALITE. It is introduced in the SODALITE development infrastructure, which is presented in Figure 7, and resides in the DevCloud (described in Section 3.2). As the changes of the source code of the components are submitted to the SODALITE repository (available at https://projects.hlrs.de/projects/SODALITE) by the developers, it triggers Jenkins to run the CI/CD pipeline, where automated unit, integration and functional tests of SODALITE components are scheduled. These tests validate the changes and verify that the updates did not break the build. As soon as the tests are passed, the source code changes are pushed into the repository. The SODALITE components are then ready for deployment (as a new production version of the SODALITE platform) and are subsequently available to the users, such as Application DevOps, Resource and Quality Experts, described in deliverable D2.1, Section 2.2.

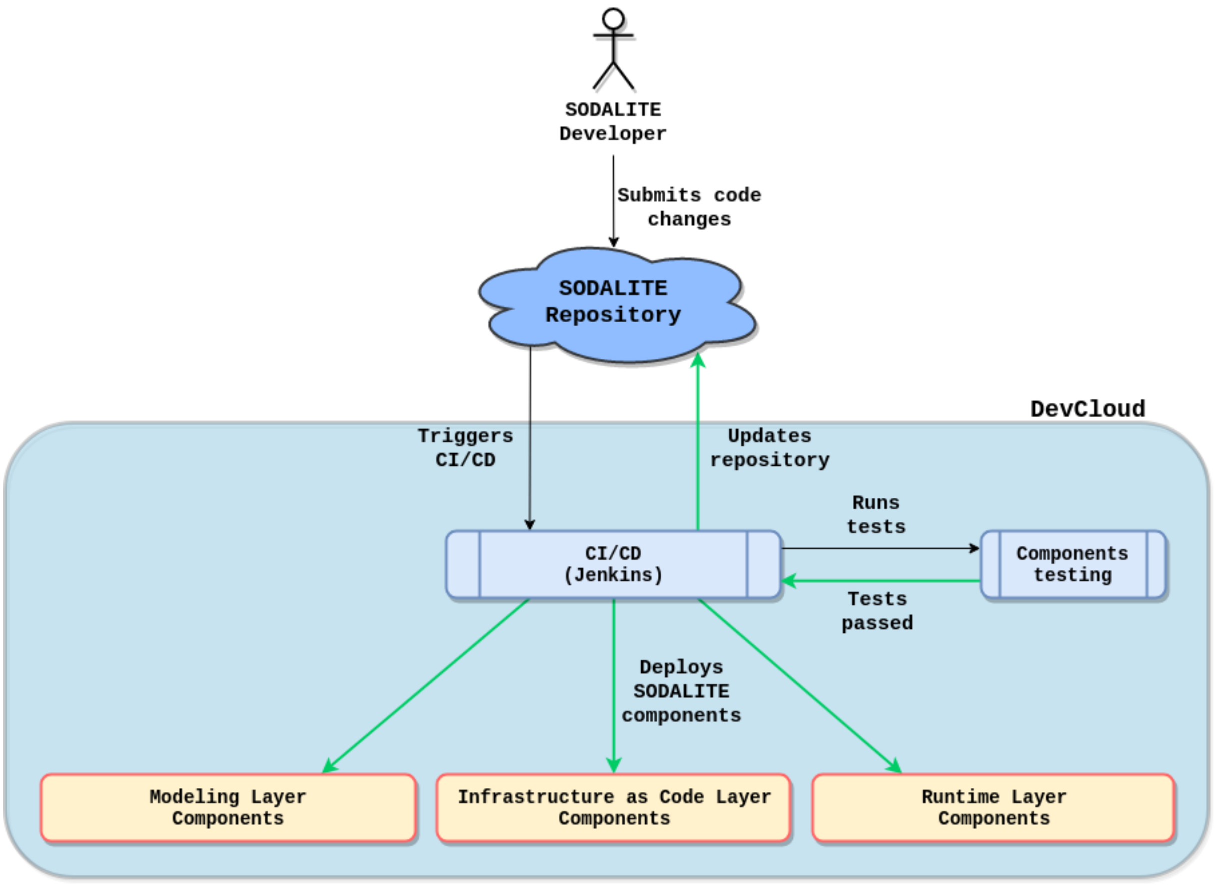
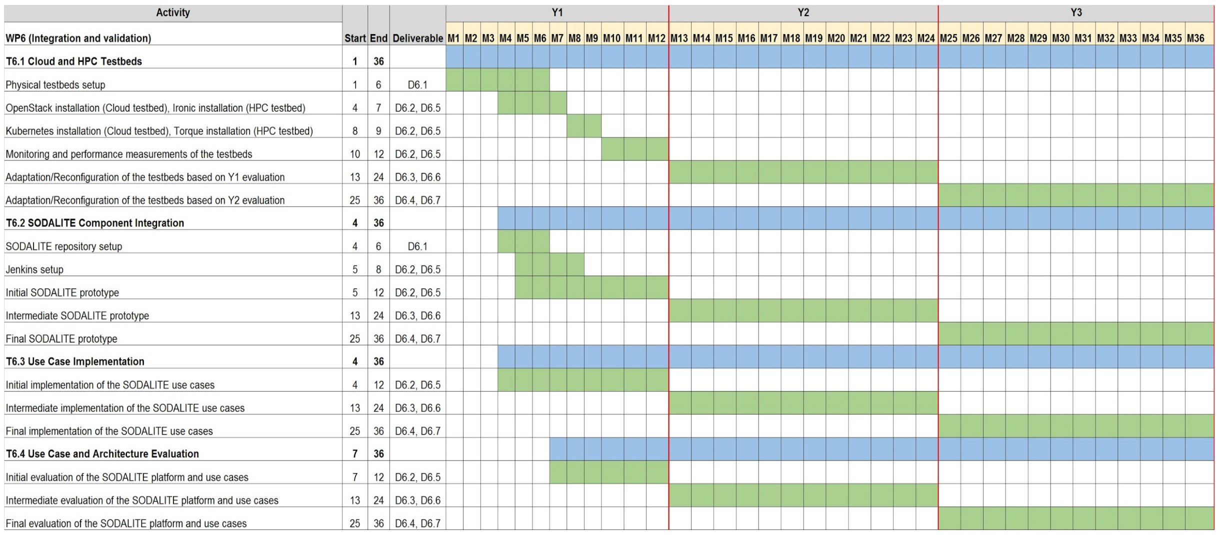


Figure 7: SODALITE development infrastructure

3.4 Timeline for Development



Activity				Y1												Y2												Y3											
WP6 (Integration and validation)	**Start**	**End**	**Deliverable**	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
T6.1 Cloud and HPC Testbeds	**1**	**36**		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Physical testbeds setup	1	6	D6.1	🟩	🟩	🟩	🟩	🟩	🟩																														
OpenStack installation (Cloud testbed), Ironic installation (HPC testbed)	4	7	D6.2, D6.5				🟩	🟩	🟩	🟩																													
Kubernetes installation (Cloud testbed), Torque installation (HPC testbed)	8	9	D6.2, D6.5								🟩	🟩																											
Monitoring and performance measurements of the testbeds	10	12	D6.2, D6.5										🟩	🟩	🟩																								
Adaptation/Reconfiguration of the testbeds based on Y1 evaluation	13	24	D6.3, D6.6													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Adaptation/Reconfiguration of the testbeds based on Y2 evaluation	25	36	D6.4, D6.7																									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
T6.2 SODALITE Component Integration	**4**	**36**					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
SODALITE repository setup	4	6	D6.1				🟩	🟩	🟩																														
Jenkins setup	5	8	D6.2, D6.5					🟩	🟩	🟩	🟩																												
Initial SODALITE prototype	5	12	D6.2, D6.5					🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩																								
Intermediate SODALITE prototype	13	24	D6.3, D6.6													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Final SODALITE prototype	25	36	D6.4, D6.7																									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
T6.3 Use Case Implementation	**4**	**36**					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Initial implementation of the SODALITE use cases	4	12	D6.2, D6.5				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩																								
Intermediate implementation of the SODALITE use cases	13	24	D6.3, D6.6													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Final implementation of the SODALITE use cases	25	36	D6.4, D6.7																									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
T6.4 Use Case and Architecture Evaluation	**7**	**36**								🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Initial evaluation of the SODALITE platform and use cases	7	12	D6.2, D6.5							🟩	🟩	🟩	🟩	🟩	🟩																								
Intermediate evaluation of the SODALITE platform and use cases	13	24	D6.3, D6.6													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Final evaluation of the SODALITE platform and use cases	25	36	D6.4, D6.7																									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 8: WP6 development timeline

The development timeline for all WP6 tasks is depicted in the Gantt chart in Figure 8. With respect to T6.1 "Cloud and HPC Testbeds", the setup of the physical testbeds (both Cloud and HPC) started in M1 of the project and will be completed by the end of M6. One month later (M7), it is expected to have OpenStack and Ironic installed on the Cloud and HPC testbeds, respectively. This will be followed by the installation of Kubernetes (Cloud testbed) and Torque (HPC testbed), which will be completed in M9, thus realizing resource provisioning for the experimentation with the SODALITE components. The remaining 3 months until the end of the first project year (M10-M12) will be allocated to monitoring and measuring the performance of the SODALITE testbeds. In the second and third years of the project, adaptations/reconfigurations of any of the underlying components in all testbeds are foreseen, based on the results of the evaluation task (T6.4 "Use Case and Architecture Evaluation") that will be performed on the overall SODALITE solution at the end of each project year.

Regarding the SODALITE component integration task (T6.2), the first activities concern the setup of the project's development infrastructure, namely the SODALITE repository (M4-M6) and the Jenkins automation server (M5-M8). The initial version of the SODALITE platform will be provided in M12 of the project and will mainly involve the integration of the components that constitute the SODALITE system, thus realizing it as a whole, but with limited functionality. The intermediate (M24) and final (M36) versions of the SODALITE platform will extend and refine the initial platform version, delivering increasingly enriched functionality and further capabilities with each updated version. The Use Case Implementation (T6.3) and Use Case and Architecture Evaluation (T6.4) tasks will follow the same incremental approach as in the case of the SODALITE platform, with the initial, intermediate and final use case implementations and evaluations of the SODALITE platform and use cases being delivered at the end of each project year.

4 Demonstrating Use Case descriptions and implementation plans

This Section describes the three demonstrating use cases of SODALITE and provides their detailed implementation plans, in terms of the envisioned functionalities of their components, as well as their development timelines. The three use cases will highlight the developed work in SODALITE and will serve as real-world demonstrators of the novel concepts brought by the project. Each one of them covers a specific professional application or industry and their expected impact to the broader community is manifold. We note here that the requirements, which are specific to the three demonstrating use cases and have been extracted by the use case owners (POLIMI, USTUTT, ADPT) during the first iteration of requirements elicitation, within the scope of WP2, are not covered in deliverable D2.1 and are instead provided in an Appendix of this deliverable, for a better connection and understandability of the use case descriptions.

4.1 POLIMI Snow UC

4.1.1 Description

The goal of this use case is to exploit the operational value of information derived from public web media content to support environmental decision making in a snow dominated context. An automatic system crawls geo-located images from heterogeneous sources at scale, checks the presence of mountains in each photo, identifies individual peaks, and extracts a snow mask from the portion of the image denoting a mountain.

Two main image sources are used: we crawl touristic webcams in the Alpine area and search Flickr for geo-tagged user-generated mountain photos in the Alpine region.

Both image types carry, explicitly or implicitly, information about the location where the image is taken, but require estimating the orientation of the camera during the shot, identifying the visible mountain peaks, and filtering out images not suitable for snow analysis (e.g., due to fog, rain etc.).

The two multimedia processing pipelines, shown in Figure 9, share common steps but also have differences: webcams produce a temporal series of images of the same view, so that only one webcam image needs to go through the relevance classification and peak identification steps, whose results apply to the entire time series. Instead, all crawled user-generated photos need pre-filtering, for discarding irrelevant content before processing them for orientation and peak detection.

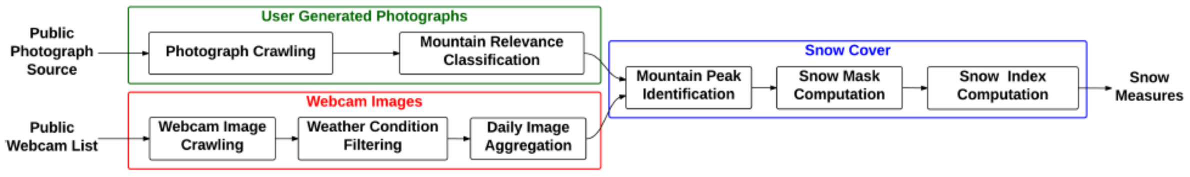


Figure 9: Schema of the SNOW use case pipelines

The project is composed of different components indicated in Figure 9, which are described in the next Sections.

4.1.2 Implementation plan: description of components

4.1.2.1 User generated image processing pipeline

The type of content that can be extracted from web social media platforms depends on the nature of the platform and usually includes one or more of the following: text, images, videos and geographical information. Photographs are taken from different locations, possibly capturing different views of the same mountain peak, but their density varies significantly depending on the location (with higher spatial density near popular tourist destinations) and time of the year (with higher temporal density during holidays).

4.1.2.1.1 User generated image crawler (UGIC)

Flickr is selected as the data source for user-generated photographs, because it contains a large number of publicly available images, many of which have an associated geotag (GPS latitude and longitude position saved in the EXIF (Exchangeable Image File Format) container of the photograph).

The Flickr API allows one to query the service using temporal and spatial filters. A user generated images (UGI) crawler algorithm is designed to query sub-regions on the area of the Alps.

Table 8 provides a summary of the user generated image crawler (UGIC) component.

Input	Coordinates of the search region bounding box Mountain-related textual keywords
Processing	● Open a connection to the query API of the user-generated image repository ● submit queries formulated with the input keywords, ● retrieve the images that match the query, ● stores the images on disk
Output	Images
Implementation technologies and languages	● Java ● PosgresSQL to save image metadata

Table 8: User generated image crawler component summary

4.1.2.1.2 Mountain relevance classifier (MRC)

Pictures tagged with a location corresponding to a certain mountainous region do not ensure the presence of mountains. For this reason, the presence of mountains in every photograph is estimated and the non-relevant photographs are discarded. The process to classify an image first computes a fixed-dimensional feature vector, which summarizes the visual content, and then provides it to a Support Vector Machine (SVM) classifier to determine whether the image should be discarded or not. A dataset of images annotated with mountain/no mountain labels is needed to train the model.

Table 9 summarizes the mountain relevance classifier (MRC) component.

Input	An image
Processing	● Calculate Image Features ● Input the features into an SVM
Output	Classification of the image (mountain, no mountain) used to decide if image should be discarded or not.
Implementation technologies and languages	● Python-TensorFlow/Matlab

Table 9: Mountain relevance classifier component summary

4.1.2.2 Public webcam processing pipeline

Outdoor webcams represent a valuable source of visual content. The images need to be filtered by the weather conditions, since these can significantly affect short- and long-range visibility. Additionally, snow cover changes slowly over time, so that one measurement per day is sufficient; for this reason, an aggregation of the images obtained during the day is desirable.

4.1.2.2.1 Webcam image crawler (WIC)

Public webcams expose a URL which returns the most recent available image. The webcam crawler:

- Loads the list of all the webcams in the dataset and starts asynchronous loops, one for each webcam.
- At each loop iteration, it checks the corresponding webcam image and adds the image to the dataset if it is changed w.r.t. the previous iteration, then idles and starts over again. Since downloading the entire image to check a webcam new data consumes bandwidth unnecessarily, the new image check is performed only on a portion of the image. Namely, only the first 5KB of the image are downloaded, hashed and compared to the previous webcam hash: if the hash is different, it is saved as the new hash and the rest of the image is downloaded. After the crawler boots, the first image acquired from every webcam is discarded, as there are no guarantees on its timestamp (some webcams, due to failures, propose the same images for days or months).

In Table 10, a summary of the webcam image crawler (WIC) component is provided.

Input	A list of webcams endpoints
Processing	For each webcam: ● Connect to the service to download the first 5KB of an image ● Generate the hash of the downloaded portion and compare with last downloaded image ● Compare hash of the two images, if the two hashes are equal, skip ● Download and save the entire image ● Wait 1’
Output	Images for each webcam temporarily saved on disk
Implementation technologies and languages	● JavaScript ● NodeJS

Table 10: Webcam image crawler component summary

4.1.2.2.2 Weather condition filter (WCF)

Due to bad weather conditions that significantly affect short- and long-range visibility (e.g., clouds, heavy rains and snowfalls), only a fraction of the images can be exploited as a reliable source of information for estimating snow cover. The weather condition filter is based on the assumption that if the visibility is sufficiently good, the skyline mountain profile is not occluded.

Table 11 describes the weather condition filter (WCF) component.

Input	A webcam image The binary mask corresponding to the webcam.
Processing	● The edge map of the input image is computed. ● The skyline visibility value is computed
Output	Boolean value indicating if it should be deleted or not.
Implementation technologies and languages	● Python/Matlab

Table 11: Weather condition filter component summary

4.1.2.2.3 Daily median image aggregation (DMIA)

Good weather images might suffer from challenging illumination conditions (such as solar glare and shadows) and moving obstacles (such as clouds and persons in front of the webcam). At the same time, snow cover changes slowly over time, so that one measurement per day is sufficient.

Therefore, the DMIA aggregates the images collected by a webcam in a day, to obtain a single representative image to be used for further analysis. A median aggregation algorithm can deal with images taken in different conditions, removing transient occlusions and glares. Given N good weather daily images I1, I2, …In the Daily Median Image (DMI) is obtained as applying the median operator along the temporal dimension.

The daily median image aggregation (DMIA) component is summarized in Table 12.

Input	A list of images obtained in one day for each single webcam
Processing	● Calculate the global offset of each image with respect to the first image of the day ● Adjust each image based on the calculated offset ● Calculate DMI
Output	For each webcam, a DMI
Implementation technologies and languages	● Python/Matlab

Table 12: Daily median image aggregation component summary

4.1.2.3 Snow cover pipeline

The distance between the shooting location and the framed mountains can be very high (tens of KMs). The photo geotag only is not sufficient for the analysis of the mountains. It is necessary to determine which portions of the image represent which mountains, identify the geographical correspondence of each pixel: estimate whether it is a terrain surface or sky, what is the corresponding geographical area, what are its GPS coordinates, altitude and distance from the observer. Once an image is geo-registered, the portion of the image that represents the mountain area can be analysed and divided into snow and non-snow areas. Mountain Image Geo-registration (MIGR) is done by finding the correct overlap between the photograph and a 360-degree cylinder with a virtual mountain panorama, i.e., a synthetic image of the visible mountain skyline generated with a projection from DEM (Digital Elevation Model) data and from the camera shooting position.

4.1.2.3.1 Skyline extraction (MIGR-SE)

To compute the alignment of the photo and the virtual panorama, the two images should have the same scale, i.e. the same pixel size. Since the photograph and the virtual panorama are taken/generated from the same location, the angular size of the mountains on the photograph and that of the mountains on the panorama are equal by definition. The horizontal FOV (Field Of View) of the photograph is calculated from the focal length and the size of the camera sensor. Then, the photograph is rescaled considering that the width of the panorama corresponds to a FOV equal to 360°. The next step is to obtain the landscape skyline of a photograph, i.e., the set of all the points that represent the boundary between the terrain slopes and the sky. For this purpose, every pixel of the input image is fed to a binary classifier, and only positive edges are retained. The training and validation of the classifier is done using a dataset of mountain images, where for each one exists an annotation containing the skyline present on it.

Table 13 presents a summary of the skyline extraction (MIGR-SE) component.

Input	Image Image FOV
Processing	● Image resize based on FOV ● Provide the image to the classifier that will output a mask indicating for each pixel whether it corresponds to the skyline or not. (The new skyline mask does not match the size of the original image)
Output	Skyline mask

Implementation technologies and languages	● Java ● OPENCV

Table 13: Skyline extraction component summary

4.1.2.3.2 360° panorama generation (MIGR-360PG)

From the coordinates of the picture, we process the 360° panoramic view of the terrain using the DEM of the terrain publicly available. The functionality is exposed as a service.

The rendering model is composed by a C++ program that initialize the context in which the OpenGL graphics API operates and exploit hardware-accelerated graphics capabilities by invoking shader programs to perform rendering operations.

This component uses the DEM files provided by NASA.

In Table 14, the 360° panorama generation (MIGR-360PG) component is summarized.

Input	Latitude and Longitude Terrain Model Precision (3" or 1") Relative altitude of the viewer (meters from the ground) Maximum visible distance
Processing	● Loading of DEM, ● Initialisation or OpenGL rendering, ● Execution of OpenGL rendering, ● Extraction and conversion of results
Output	360° panorama depthmask
Implementation technologies and languages	● C++ 14 compliant ● OPENGL, EGL ● Java, JavaScript and NodeJS (to make the panorama web-accessible)

Table 14: 360° panorama generation component summary

4.1.2.3.3 Peak alignment (MIGR-PA)

The alignment can be seen as the search for the correct overlap between two cylinders (assuming the zero tilt of the photograph): one containing the 360° panorama and the other one containing the photo, suitably scaled.

Table 15 describes the peak alignment (MIGR-PA) component.

Input	An image with its corresponding skyline annotation and the 360° panorama corresponding to its location
Processing	● Perform global alignment between skyline and panorama
Output	Image annotated with the mountain peaks M = A mask indicating pixels that correspond to the mountain surface.
Implementation technologies and languages	● Java ● OPENCV

Table 15: Peak alignment component summary

4.1.2.3.4 Snow mask computation (SMC)

A snow mask is defined as the output of a pixel-level binary classifier that, given an image and a mask M that represents the mountain area as inputs, produces a mask S that assigns each pixel of the mountain area a binary label denoting the presence of snow. Snow masks are computed using the Random Forest supervised learning classifier with spatio-temporal median smoothing of the output. To perform the supervised learning a dataset of images with an annotation at pixel level indicating if the pixel corresponds to the snow area is needed.

The snow mask computation component (SMC) is described in Table 16.

Input	An image and a mask indicating the pixels corresponding to the mountain area.
Processing	● Calculate feature vectors for the pixels in the mountain area ● Input the features into the Random Forest Classifier
Output	S = Snow mask indicating for each pixel if it represents snow or not in the original image.
Implementation technologies and languages	● Python/Matlab

Table 16: Snow mask computation component summary

4.1.2.3.5 Snow index computation (SIC)

The pipeline produces a pixel-wise snow cover estimation from images, along with a GPS position, camera orientation, and mountain peak alignment. Thanks to the image geo-registration and orthorectification (using the associated topography data) it is possible to estimate the geographical properties of every pixel, such as its corresponding terrain area and altitude. Consequently, it is possible to compute the snow line altitude (the point above which snow and ice cover the ground) expressed in meters.

The virtual snow index for an image is defined as:$\Sigma_{(x,y)\,|\,S(x,y)=1}\, vsi(x,y)$, where vsi is a virtual snow index function that transforms a pixel position into a snow relevance coefficient and can be defined as $vsi_{(x,y)} = 1$ and $S(x,y) = 1$ indicates it will be calculated for each pixel that corresponds to the snow mask obtained in previous step.

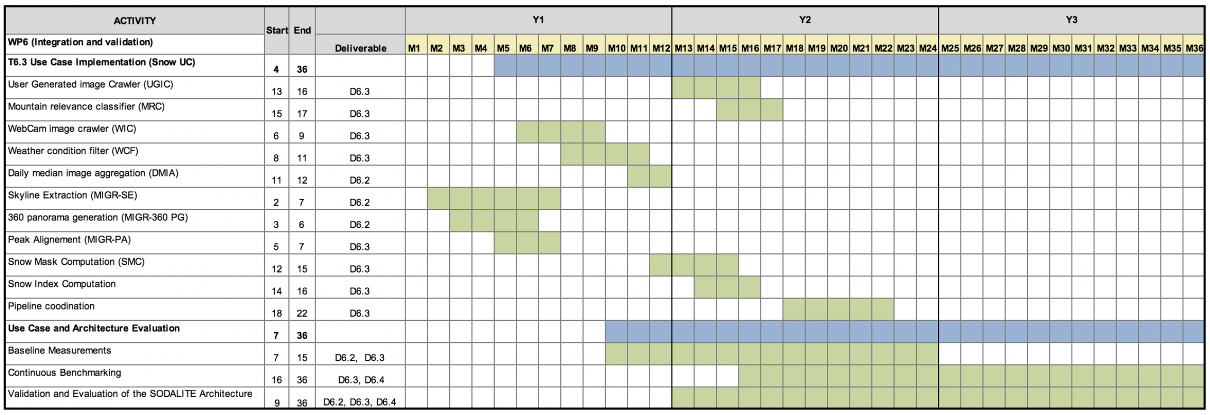
Table 17 provides a summary of the snow index computation (SIC) component.

Input	S = the snow mask M = the mountain area mask
Processing	Calculate the VSI
Output	Virtual snow index
Implementation technologies and languages	● Python/Matlab

Table 17: Snow index computation component summary

4.1.3 Implementation plan: timeline

The development of the components that constitute the SNOW Use Case pipeline is scheduled as shown by the Gantt diagram displayed in Figure 10. At the end of the components development there is another phase to ensure the flow of data among the components. There is a further phase to compare the original baseline with redeployment resulting from the output from the SODALITE Platform.



ACTIVITY	Start	End		Y1												Y2												Y3											
WP6 (Integration and validation)			**Deliverable**	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
T6.3 Use Case Implementation (Snow UC)	**4**	**36**						🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
User Generated image Crawler (UGIC)	13	16	D6.3													🟩	🟩	🟩	🟩																				
Mountain relevance classifier (MRC)	15	17	D6.3															🟩	🟩	🟩																			
WebCam image crawler (WIC)	6	9	D6.3						🟩	🟩	🟩	🟩																											
Weather condition filter (WCF)	8	11	D6.3								🟩	🟩	🟩	🟩																									
Daily median image aggregation (DMIA)	11	12	D6.2											🟩	🟩																								
Skyline Extraction (MIGR-SE)	2	7	D6.2		🟩	🟩	🟩	🟩	🟩	🟩																													
360 panorama generation (MIGR-360 PG)	3	6	D6.2			🟩	🟩	🟩	🟩																														
Peak Alignement (MIGR-PA)	5	7	D6.3					🟩	🟩	🟩																													
Snow Mask Computation (SMC)	12	15	D6.3												🟩	🟩	🟩	🟩																					
Snow Index Computation	14	16	D6.3														🟩	🟩	🟩																				
Pipeline coodination	18	22	D6.3																		🟩	🟩	🟩	🟩	🟩														
Use Case and Architecture Evaluation	**7**	**36**											🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Baseline Measurements	7	15	D6.2, D6.3										🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Continuous Benchmarking	16	36	D6.3, D6.4																🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Validation and Evaluation of the SODALITE Architecture	9	36	D6.2, D6.3, D6.4													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 10: Gantt diagram of the development of the Snow use case pipeline components

4.2 USTUTT Virtual Clinical Trial UC

4.2.1 Description

The in-silico clinical trials for spinal operations use case targets the development of a simulation process chain supporting in-silico clinical trials of bone-implant-systems in Neurosurgery, Orthopedics and Osteosynthesis. It deals with the analysis and assessment of screw-rod fixation systems for instrumented mono- and bi-segmental fusion of the lumbar spine by means of continuum mechanical simulation methods. As a novelty, we consider the uncertainty inherent in the computation by means of probabilistic programming. The simulation chain consists of a number of steps that need to be fulfilled in order and can be thought of as a pipeline. The output of each step serves as input to the next step.

The use case addresses one of the most prevalent health problems experienced by the populations of developed nations resulting in enormous losses of productivity and costs for ongoing medical care. The simulation process developed within this use case will optimise the screw-rod fixation systems based on clinical imaging data recorded during standard examinations and consequently target the lowering of the reported rates of screw loosening and revisions, enhance safety, expand the knowledge of the internal mechanics of screw-rod fixation systems applied to the lumbar spine and finally reveal optimization potential in terms of device application and design.

4.2.2 Implementation plan: description of components

The individual steps for the simulation chain can be seen in Figure 11 below. First, the extraction module component takes imaging data and extracts a geometry for the vertebral bodies. The de facto standard for doing this is the *marching cubes algorithm*, for which many implementations exist.

Next, the discretization module component generates a volume-mesh inside the surface geometry. This enables one to treat the mesh as a set of finite elements and to use the existing finite elements methodology.

The density mapping component takes the original image data and maps it onto the volume-mesh. In doing this, each element inside the mesh is assigned a density value.

This enhanced meshed geometry is then fed into the probabilistic mapping module component. Here the values for density are transformed into values for elasticity, as this is what is actually needed for the simulation. Because of the uncertainty that is inherent in this transformation, we use a probabilistic programming approach. Eventually, boundaries for the 95% highest density interval as well as the mode are computed.

These data are used in the input decks for the last step, the solver module component. Here, the finite element method is actually used in computing a solution that describes the structural mechanics inside the vertebral bodies.

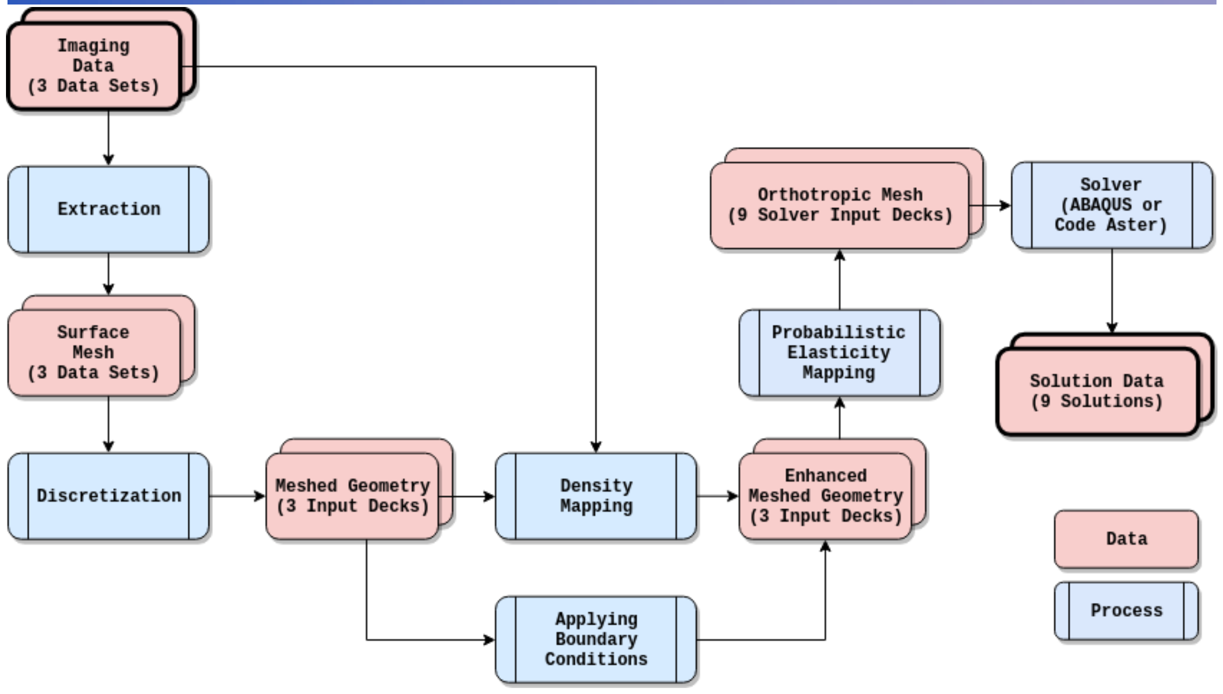

Imaging Data (3 Data Sets)
Extraction
Surface Mesh (3 Data Sets)
Discretization
Meshed Geometry (3 Input Decks)
Density Mapping
Applying Boundary Conditions
Enhanced Meshed Geometry (3 Input Decks)
Probabilistic Elasticity Mapping
Orthotropic Mesh (9 Solver Input Decks)
Solver (ABAQUS or Code Aster)
Solution Data (9 Solutions)
Data
Process

Figure 11: Schema of the Virtual Clinical Trial use case pipeline

4.2.2.1 Extraction

In this first step the geometry of the vertebral bodies is extracted from CT-data (Computer Tomographic) which are recorded pre-operatively, post-operatively and approximately six-weeks post-operatively. By means of the marching cubes algorithm three triangulated surface meshes are generated which serve as input for the next processing step. Depending on the quality of the CT-data it might be necessary to introduce additional image filtering techniques into the extraction step. These filters are available in the Visualization Toolkit (VTK) or in the Insight Segmentation and Registration Toolkit (ITK) which are both open source C++ libraries.[47] Alternatively, manual pre-processing of the datasets can be performed upfront.

Table 18 summarizes the functionality of the extraction component.

Input	Three data sets per patient containing data from CT scans, one preoperatively and two postoperatively.
Processing	The marching cubes algorithm is applied to each data set to extract a surface geometry. Successful extraction depends on the CT data quality. Because of this it might be necessary to do some manual processing upfront.
Internal concurrency	No, sequential process
Output	Three so called surface meshes
Implementation technologies and languages	Fortran/C++

Table 18: Extraction component summary

4.2.2.2 Discretization

Based on the three surface meshes volume meshing of the bone geometries is performed. Targeted libraries for 3D volume mesh generation are currently Netgen or NGSolve. Additionally the boundary conditions i.e. the supports and the loadings have to be discretized and structural model features like muscle strands, tendons and cartilage have to be attached to the modeled bone geometries. As a result of this step the completed models will be written out as so called solver input decks still with a homogeneous material distribution.

In Table 19, the functionality of the discretization component is described.

Input	Three data sets with surface meshes per patient.
Processing	A 3-dimensional mesh is generated inside each of the surface meshes.
Internal concurrency	No, sequential process
Output	So called meshed geometries in three solver input decks.
Implementation technologies and languages	Netgen/NGSolve (written in C++, Python Interface)

Table 19: Discretization component summary

4.2.2.3 Density Mapping

In this step the three generated input decks as well as the three CT data sets are taken as input. By means of direct geometrical mapping the grayscale distribution of the respective CT data set is mapped onto the volume mesh generated in the previous step. After the mapping, each element in the volume mesh holds a density value from which in the next step orthotropic material data can be generated.

This step is done by a Fortran implementation as described in Schneider, R. et al. - Inhomogeneous, orthotropic material model for the cortical structure of long bones modelled on the basis of clinical CT or density data[48].

Table 20 provides a summary of the density mapping component's functionality.

Input	Three input decks with meshed geometries per patient. CT data sets.
Processing	From the CT data a value for density is mapped onto each element inside the mesh.
Internal concurrency	No, sequential process
Output	Three modified solver input decks per patient.
Implementation technologies and languages	Fortran

Table 20: Density mapping component summary

4.2.2.4 Probabilistic elasticity mapping

In this step the volume meshes, which are augmented with the density distributions from the CT data sets, are fed into the probabilistic mapping module component. Here the values for density are transformed into orthotropic material parameters, i.e values representing the elasticity of each finite element, as this is what is actually needed for the subsequent solver step. Because of the uncertainty that is inherent in this transformation, we use a probabilistic programming approach. the targeted output are currently the boundaries for the 95% highest density interval as well as the mode. This means each of the three states of the patient, pro-operativ, post-operative and approximately six weeks post-operative is transformed into three models representing the uncertain material distributions in each state.

The probabilistic elasticity mapping component is summarized in Table 21.

Input	Three modified input decks per patient.
Processing	From the density values, a probability distribution for the elasticity for each element is computed. To keep the amount of data manageable, only the low and high bounds of a to be determined density interval (e.g. 95%) and the mode are extracted.
Internal concurrency	MPI is used for parallel computation.
Output	For each patient and for each input deck, three solver input decks (low, high, mode) are computed.
Implementation technologies and languages	Python MPI

Table 21: Probabilistic elasticity mapping component summary

4.2.2.5 Solver

To solve the nine cases per patient resulting from the previous step, the open source software package Code Aster is used. Finally, nine results are produced which show the strain and stress distribution within the simulated structures as well as the displacement field. These results are initially object to manual post-processing procedures but as soon as lessons are learned from the manual procedures, automatic or semi-automatic data analytics procedures will be set in place.

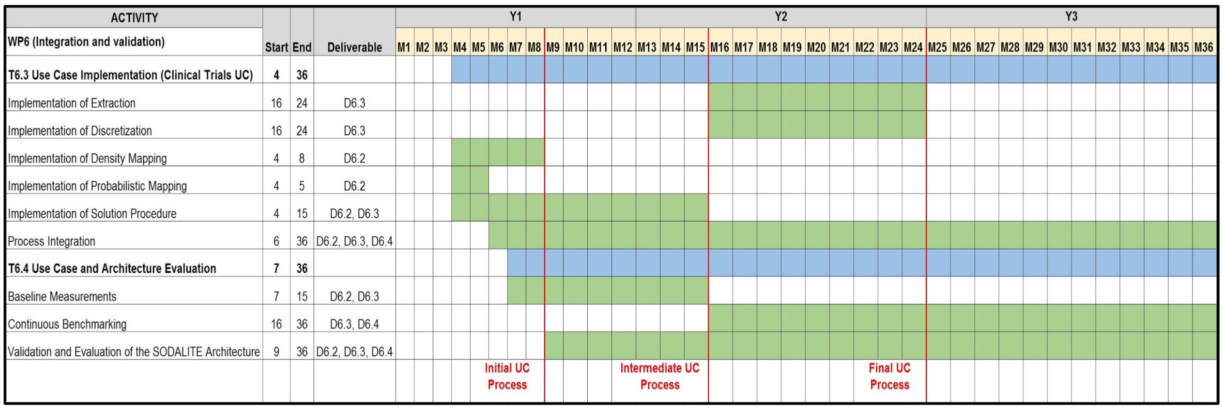
Table 22 describes the functionality of the solver component.

Input	Nine solver input decks per patient.
Processing	Using finite element methods, a solution is computed for lower and upper bound of HDI as well as for the mode. These three solutions are computed for the pre- and the two postoperatively acquired datasets.
Internal concurrency	MPI can used for parallel computation.
Output	For each input deck a solution file is computed.
Implementation technologies and languages	Code Aster (written in Fortran)

Table 22: Solver component summary

4.2.3 Implementation plan: timeline

The development of the components that constitute the Virtual Clinical Trial UC process is scheduled as shown by the Gantt diagram displayed in Figure 12 below.



ACTIVITY			
WP6 (Integration and validation)	Start	End	Deliverable
T6.3 Use Case Implementation (Clinical Trials UC)	**4**	**36**	
Implementation of Extraction	16	24	D6.3
Implementation of Discretization	16	24	D6.3
Implementation of Density Mapping	4	8	D6.2
Implementation of Probabilistic Mapping	4	5	D6.2
Implementation of Solution Procedure	4	15	D6.2, D6.3
Process Integration	6	36	D6.2, D6.3, D6.4
T6.4 Use Case and Architecture Evaluation	**7**	**36**	
Baseline Measurements	7	15	D6.2, D6.3
Continuous Benchmarking	16	36	D6.3, D6.4
Validation and Evaluation of the SODALITE Architecture	9	36	D6.2, D6.3, D6.4

Y1
Y2
Y3
M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 M16 M17 M18 M19 M20 M21 M22 M23 M24 M25 M26 M27 M28 M29 M30 M31 M32 M33 M34 M35 M36
Initial UC Process
Intermediate UC Process
Final UC Process

Figure 12: Gantt diagram of the development timeline of the Virtual Clinical Trial use case components

More specifically, the development of the probabilistic mapping and density mapping components follow the M4-M5 (finished) and M4-M8 timelines, respectively. The delivery of these two components marks the first milestone (Initial UC-Process) in the implementation of the Virtual Clinical Trial Use Case (due in M8). The second milestone (Intermediate UC-Process), due in M15 of the project, is defined by the implementation of the overall solution procedure (M4-M15). This is followed by the development of the extraction and discretization components (M16-M24), which will finalize the Use Case process implementation and will complete the related milestone (Final UC-Process) at the end of the second project year.

The overall integration of the Use Case process into the SODALITE system environment (will be done progressively following the development of the constituting components) will start after the implementation of the probabilistic mapping component (M6) and will carry on until the end of SODALITE (M36). With respect to the evaluation of the Use Case and the SODALITE Architecture, baseline measurements will be acquired throughout the implementation of the initial and intermediate Use Case process (M7-M15), followed by continuous benchmarking until the end of the project. The validation and evaluation task of the Use Case will start in M9 (after the initial baseline measurements have been acquired), following the implementation of the initial, intermediate and final Use Case processes.

4.3 ADPT Vehicle IoT UC

4.3.1 Description

Through the combination of vehicle telemetry, instrumentation, and behavioural data, insurance companies are able to shape a more holistic view of an individual driver's overall risk profile based on empirical analysis of driving data (referred to as usage based-insurance, or UBI) - areas that have traditionally relied upon static data points over which the individual has little control, and which have been more focused on risk probability than empirical analysis (these factors include, e.g. age, gender, marital status, make/model of vehicle, etc.). While UBI models have been successfully engaged in markets with a more relaxed and homogeneous regulatory environment, European industry (and citizens) have been hesitant to pursue this model without adequate safeguards for personal data protection and privacy rights - a situation remediated in part by the coming into force of the GDPR (General Data Protection Regulation).

The growth of Connected Car data and concerns over data usage are further compounded by: (1) Individual expectation of contextualised service offerings that respect personal preferences and privacy expectations; (2) Service providers aiming to deploy service offerings across an increasingly dynamic environment; and (3) growing trend of drivers seeking to analyse and benefit from their own driving data directly.

These growing expectations, both from individuals and businesses, lead to an enormous increase in the volume and rate of the sensor data, its aggregation, and its analysis, at various hierarchical levels. This data, in turn, must be processed in line with the relevant privacy constraints and regulatory restrictions it is subject to - aspects subject to dynamic change, while also being highly latency-sensitive.

This leads to two key architectural demands for SODALITE: (1) an increasing amount of in-vehicle data processing and intelligence at the network edge, and (2) increased computational capacity to process large amounts of data in a timely manner - at varying levels of granularity (e.g. device-local, vehicle-local, fleet-wide) - including both fleet-wide big data analytics, as well as periodic online retraining of machine learning models that support the deployment.

This will be achieved by the use of SODALITE tools ensuring privacy-preserving distributed processing on one hand and large-scale data processing on the other. Through the use of modelling, it will be possible to enrich the processing workflows with information about data- and latency-sensitive phases (services), steering the overall placement strategies of the orchestration engines. On the fly predictive deployment refactoring will allow for optimal use of available resources by reconfiguring the whole system and distributing workloads between the heterogeneous edge (IoT/Vehicle, Edge Gateways, Fleet Gateways) and backend compute resources (Cloud or HPC) as the application evolves towards a hierarchical deployment throughout the duration of the project (as shown in Figure 13 below).

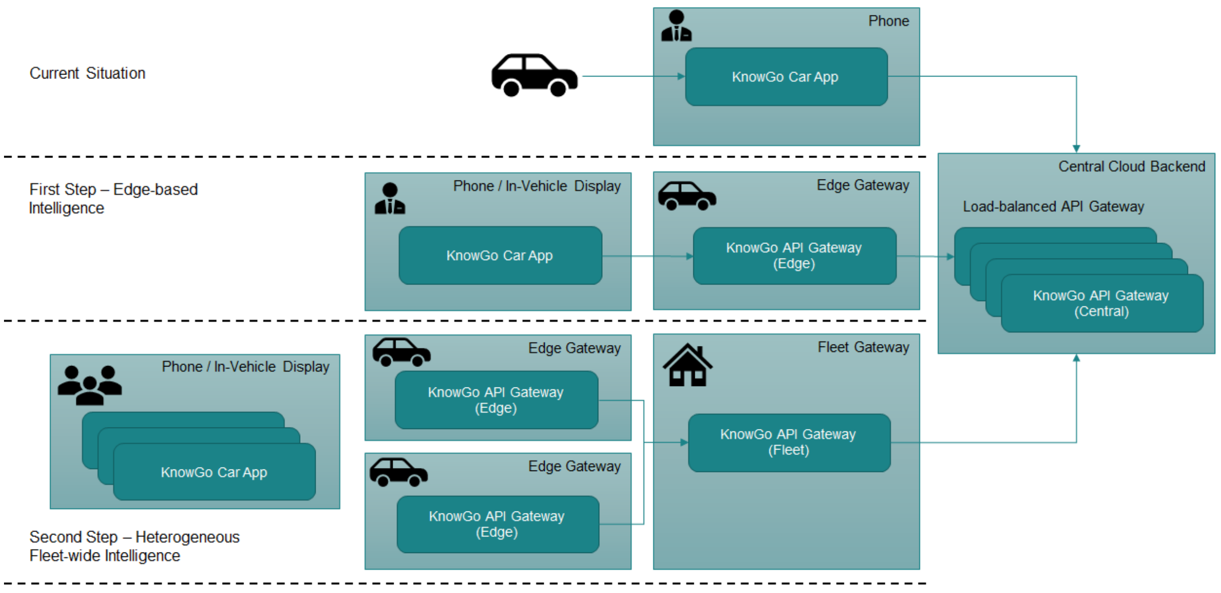

Current Situation
Phone
KnowGo Car App
First Step – Edge-based Intelligence
Phone / In-Vehicle Display
KnowGo Car App
Edge Gateway
KnowGo API Gateway (Edge)
Central Cloud Backend
Load-balanced API Gateway
KnowGo API Gateway (Central)
Phone / In-Vehicle Display
KnowGo Car App
Edge Gateway
KnowGo API Gateway (Edge)
Edge Gateway
KnowGo API Gateway (Edge)
Fleet Gateway
KnowGo API Gateway (Fleet)
Second Step – Heterogeneous Fleet-wide Intelligence

Figure 13: Schema of the Vehicle IoT use case deployment phases

The current architecture (shown in Figure 14, below), furthermore, makes extensive use of deployed microservices in the Cloud to provide many of the value-added features upon which the use case relies. These include, but are not limited to:

- License Plate Recognition
- VIN Decoding
- Reverse Geocoding
- Drowsiness Detection
- Theft and Intrusion Detection

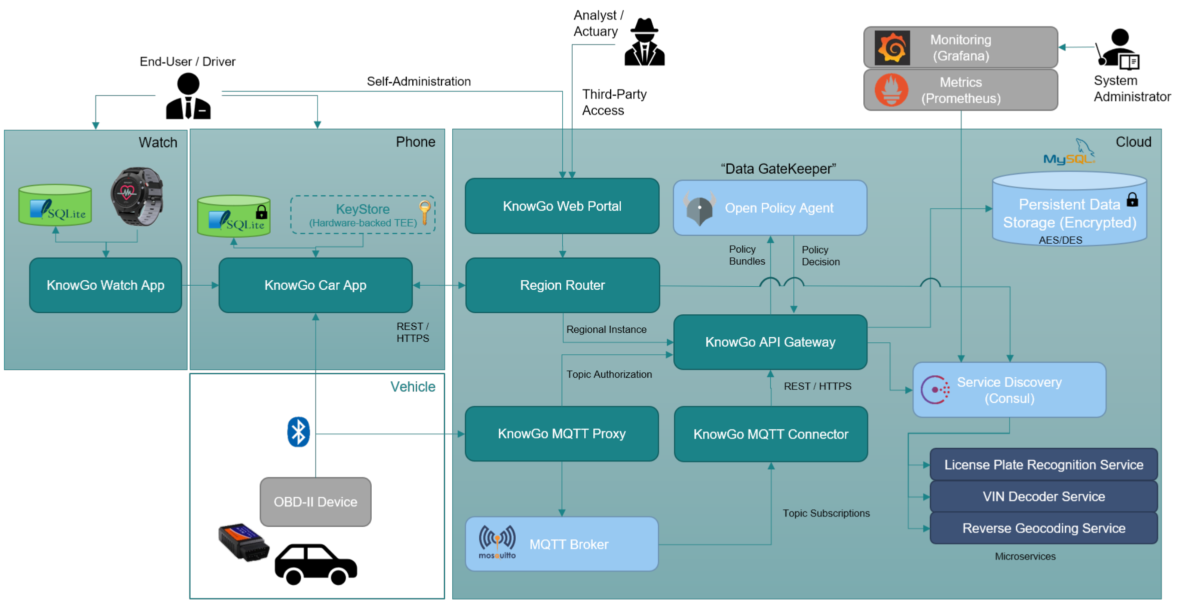


Figure 14: Currently deployed system architecture (Vehicle IoT use case)

Many of these microservices, in turn, leverage trained machine learning models, and are able to quickly provide results with minimal computational overhead, providing the opportunity to re-deploy and run these services at different hierarchical levels (backend, in-vehicle edge gateway, smartphone, etc.).

While these models can be improved through subsequent training phases, the computational overhead (and costs, in the case of public cloud deployment) involved in this often means that models are only periodically updated - delaying incremental benefits that could be deployed to the existing user base at various stages of data availability. The addition of an optional HPC or GPU compute resources at the backend, made available through the SODALITE tooling, would allow for online re-training and continuous deployment of the machine learning models, making model training and deployment a first-class citizen of the application's production CI/CD pipeline, enabling benefits to be brought to the end-user both rapidly and directly.

Other services, such as the Intrusion and Theft Detection Service introduced in Section 4.3.2.2.2 below, further require the generation of personalized SVC classifiers, changing the role of model training as an infrequent event to periodically carry out offline, to a more frequent and online one - creating further challenges in resource identification and utilization both in the Cloud and at the Edge.

4.3.2 Implementation plan: description of components

4.3.2.1 License Plate Detection Pipeline

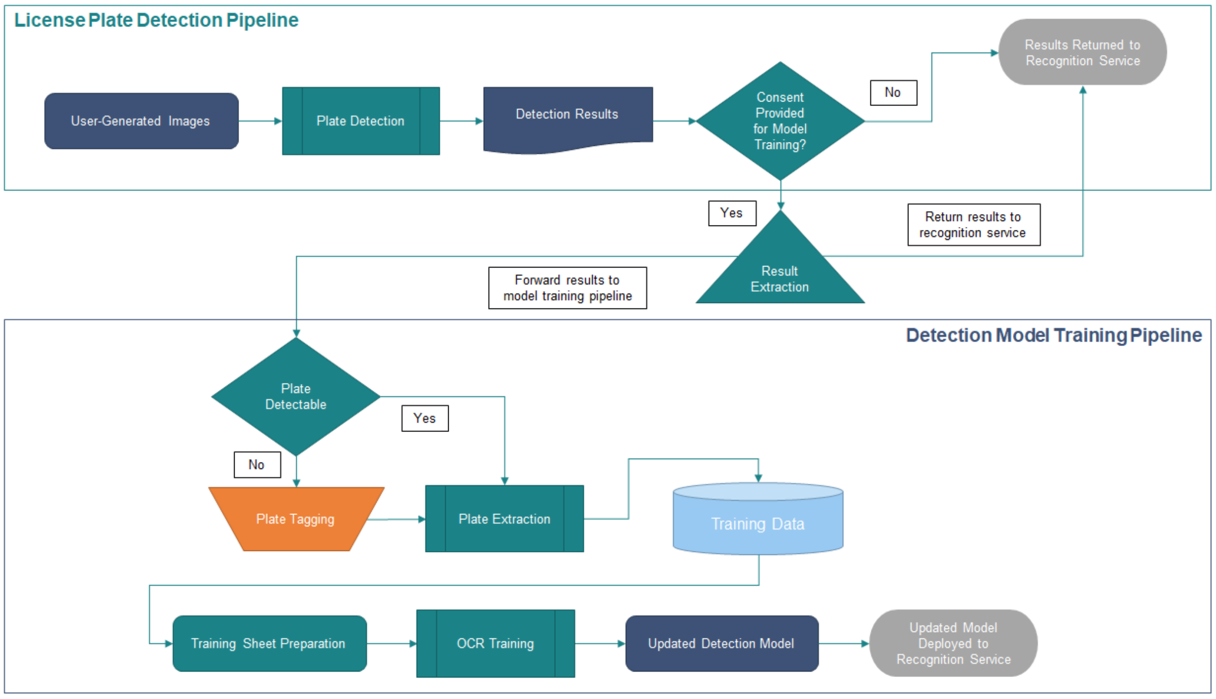

License Plate Detection Pipeline
User-Generated Images
Plate Detection
Detection Results
Consent Provided for Model Training?
No
Results Returned to Recognition Service
Yes
Return results to recognition service
Result Extraction
Forward results to model training pipeline
Detection Model Training Pipeline
Plate Detectable
Yes
No
Plate Tagging
Plate Extraction
Training Data
Training Sheet Preparation
OCR Training
Updated Detection Model
Updated Model Deployed to Recognition Service

Figure 15: Schema of the license plate detection and detection model training pipelines (Vehicle IoT use case)

Within the Vehicle IoT Use Case, individuals may, at various times, submit license plate images for recognition. These purposes include the initial registration of the vehicle with the mobile app (as one possible registration mechanism - of particular interest in countries which provide open access to their vehicle registration databases), evidence to support claims preparation (in the case of a collision), etc. In order to benefit from improved plate recognition, the use case will be expanded to include user-generated image crowd-sourcing and dynamic updating of the detection model by leveraging appropriate resources (Cloud or HPC). This is envisioned across a number of steps:

1. Inclusion of user-generated images in the training data set
2. Plate extraction from uncropped training data (Bulk processing)
3. Re-training model on suitable backend resource (e.g. GPU cluster)
4. Validating control set against the new model (regression detection)
5. Re-deployment / update of plate recognition microservice backed by the new model

This is further exemplified by the license plate detection and detection model training pipelines in Figure 15 above.

4.3.2.1.1 Training

User-generated images are crowd-sourced from the front-end application (and limited to vehicle registration, such that consent can be obtained from the end-user) - these augment the existing data set and are used for periodic retraining of the detection algorithm in order to enable more precise (and increasingly contextualized) license plate recognition.

4.3.2.1.2 Plate Extraction

While the license plate recognition service operates directly on uncropped images based on the trained model, plate images that are preserved for the purpose of model training are first cropped and extracted, discarding any other identifying characteristics or background content. Examples of user-submitted images and extracted plate images are seen in Figure 16 below:

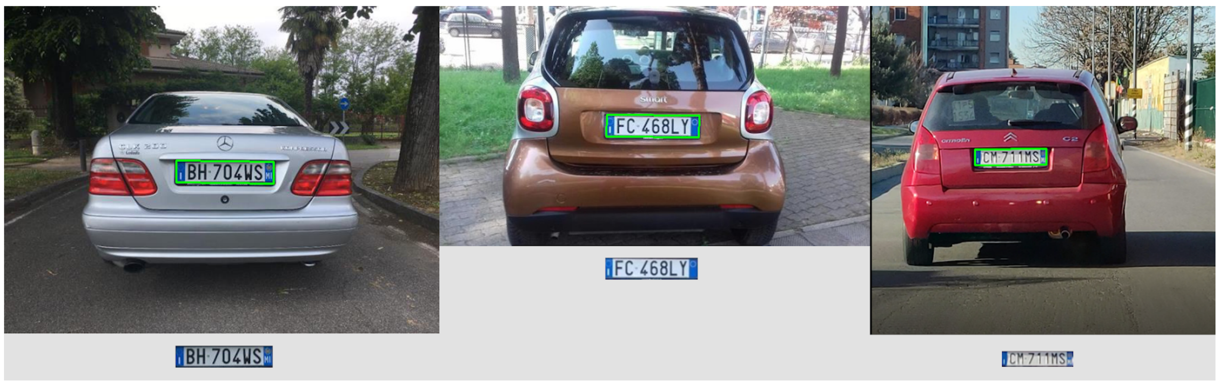


Figure 16: Schema of the license plate detection and detection model training pipelines (Vehicle IoT use case)

As training of the ML (Machine Learning) model is presently an infrequent occurrence that happens offline (and independent of the deployed application stack), it is sufficient to batch input images for periodic extraction. As the input images may contain sensitive information (e.g. the house number of the individual) unrelated to the purpose of data collection, source images must be kept securely until such a time that they are processed and promptly discarded after successful plate extraction.

A summary of the plate extraction component is provided in Table 23.

Input	Uncropped user-generated images (various sizes)
Processing	Tesseract OCR / OpenCV
Internal concurrency	No, sequential process
Output	Cropped images of license plates
Implementation technologies and languages	Technologies: Tesseract OCR, OpenCV; Languages: C++

Table 23: Plate extraction component summary

4.3.2.1.3 Plate Detection

Plate detection is carried out on the user-submitted image as-is, with failure to detect notifying the user and prompting the user if they'd like to try again, submit the image to improve the underlying ML model, or use an alternative method of vehicle registration.

Table 24 summarizes the functionality of the plate detection component.

Input	Uncropped user-generated images (as form data)
Processing	Dedicated OpenALPR-backed plate recognition microservice
Internal concurrency	No, sequential process
Output	JSON-encoded detection results
Implementation technologies and languages	Technologies: OpenALPR, Tesseract OCR, OpenCV; Languages: C++, Go

Table 24: Plate detection component summary

4.3.2.2 Advanced Video Analytics for Driver Monitoring and Alerting

A number of further application scenarios are supported through video monitoring and analysis - these include both a case where real-time analysis of a video stream is necessary (drowsiness detection), and a less latency-sensitive case where custom trained and contextualized classifiers must be provided in order for the service to provide meaningful results (intrusion and theft detection). These are elaborated in the Sections below.

4.3.2.2.1 Drowsiness Detection (Face Detection)

Drowsiness Detection aims to determine when a Driver is at risk of falling asleep at the wheel and taking evasive actions (e.g. playing a loud noise, triggering a vibration, etc.) in order to alert the driver to the problem before a more serious incident occurs.

Drowsiness detection is typically carried out using a couple of different methods, with differing levels of accuracy and invasiveness. While the gold standard (and most accurate method) for drowsiness detection remains ECG monitoring, ECG measurement itself is invasive and requires active participation by the individual under monitoring, making it a poor fit for passive observation of a driver. The most common non-invasive methods, on the other hand, are PERCLOS (Percentage of eyelid closure) - measuring the proportion of time that the eyelids are between 80-100% closed, and blink detection (Blink detection methods further being split between blink frequency and duration detection). While the PERCLOS method is fairly well established, it has also been found to generate false positives in scenarios where:

1. subjects under monitoring periodically look down (as in typing on a keyboard) in relation to the camera; and
2. in cases where the camera is oriented in such a way that it does not have head-on visibility of the driver's eyes.

As we can expect drivers to be periodically checking their dashboard readings, and cameras to frequently be mounted on an angle relative to the driver's position, blink methods are considered to be a more appropriate fit and are what is explored within this use case. An example of the EAR (Eye Aspect Ratio) method of blink detection (as is currently used in this service) can be seen in Figure 17 below:

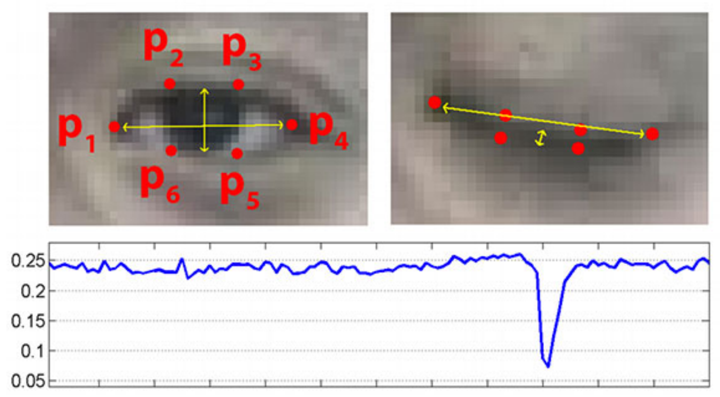


Figure 17: Blink detection using Eye Aspect Ratio (drowsiness validation service)

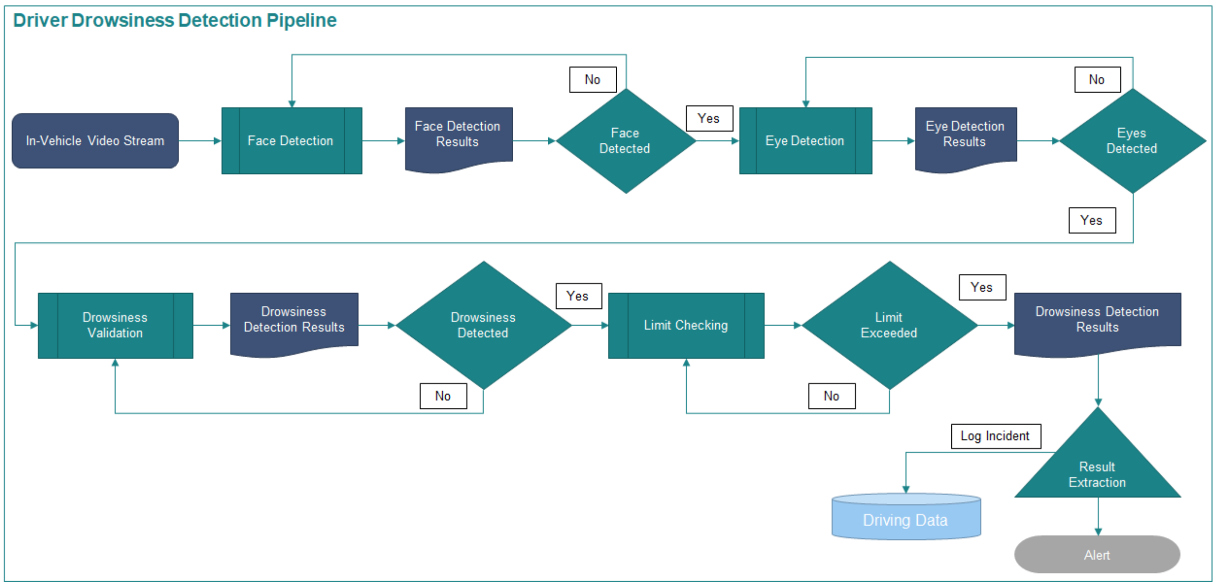


Figure 18: Schema of the driver drowsiness detection pipeline (Vehicle IoT UC)

Drowsiness detection (Figure 18) is highly latency-sensitive and must be done in real-time in order to be as accurate as possible and to alert the driver at the time they need to be alerted. Blink duration can be summarized as *awake* (< 400ms blink duration), *drowsy* (400-800ms blink duration) and *sleepy* (blink duration > 800ms). With current wireless technologies demonstrating round-trip latencies near 50-200ms (for 4G) and 500ms (for 3G) with good connectivity, a backend-deployed monitoring service cannot be expected to reliably identify and respond to drowsiness events in time - necessitating a push-down of the service delivery to the Vehicle itself.

Table 25 provides a summary of the drowsiness detection component’s functionality.

Input	Video stream from input camera
Processing	● Real-time eye and face detection with Haar Cascades ● Eye aspect ratio calculation and contour fitting (Blink detection) ● Blink duration sampling across frames, classification and alerting

Internal concurrency	Multi-threaded non-blocking I/O - dedicated thread extracts frames from the video stream, while worker thread(s) handle the actual frame analysis.
Output	JSON-encoded detection results
Implementation technologies and languages	Technologies: OpenCV, Dlib, Kafka; Languages: Python, C++

Table 25: Drowsiness detection component summary

4.3.2.2.2 Intrusion and Theft Detection (Face Recognition)

Intrusion detection builds on the face detection model developed in the drowsiness detector and defines a face recognition model capable of identifying the authorized driver's face. In the case where someone other than the designated driver is found to be driving the vehicle, further actions can be taken by the system (this may include aspects such as notifying the authorized driver and seeking confirmation of a driver switch, notifying a fleet manager, streaming vehicle telemetry to a third party, etc.). A general overview of this process is highlighted in the pipeline schema in Figure 19 below.

In contrast with drowsiness detection, intrusion and theft detection is not directly latency-sensitive, and as it does not require real-time access to the driver's video stream, is suitable for backend deployment as a long-lived microservice (notably, the infrequent nature of the invocation also makes this an ideal candidate for serverless deployments). While the authorized driver may indeed wish to know if someone is stealing their vehicle as quickly as possible, the added round-trip latency associated with mobile communications is unlikely to have a measurable impact on any asynchronous notifications that may result from the analysis.

A unique characteristic of this service is that custom classifiers must be modelled and trained in order to provide value for the Driver (that is, SVC models capable of identifying the authorized Driver's face - which the driver may take with them). This may involve dynamic training of vehicle-restricted classification models or may be open for sharing across a fleet of vehicles, or any other vehicle the end-user may use, dependent upon their individual privacy preferences and sharing settings.

The functionality of the intrusion and theft detection component is described in Table 26.

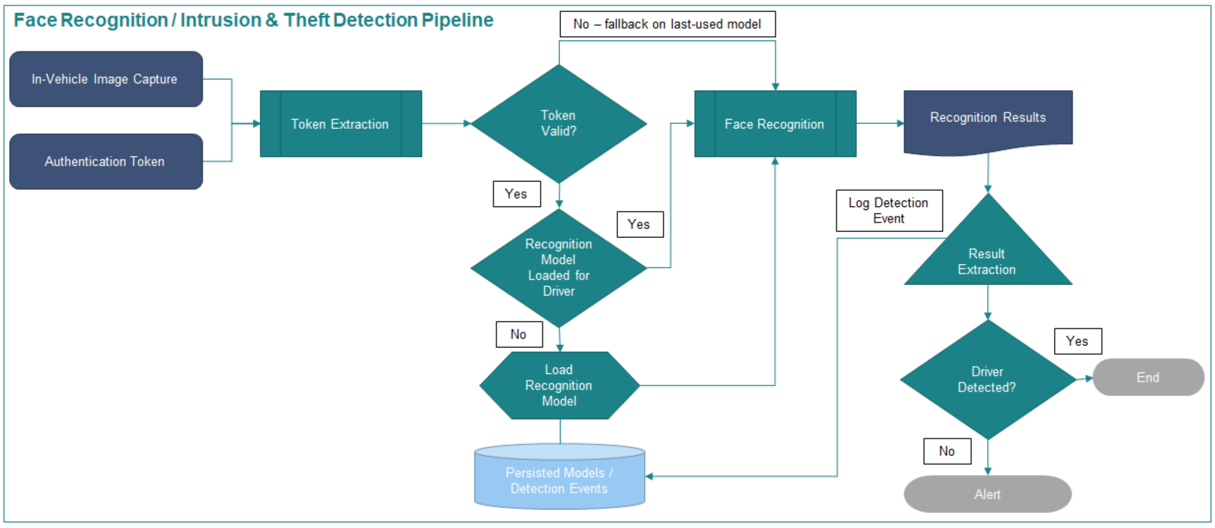


Figure 19: Schema of the intrusion and theft detection pipeline (Vehicle IoT UC)

Input	Image extracted from video stream / camera, authentication token
Processing	● JWT token validation / claims extraction ● Dynamic loading of trained SVC classifier ● Facial feature comparison of source image with loaded SVC classifier
Internal concurrency	No, sequential process
Output	JSON-encoded detection results
Implementation technologies and languages	Technologies: OpenCV; Languages: Python

Table 26: Intrusion and theft detection component summary

4.3.2.3 API Gateway

The API gateway, as shown in Figure 14 above, remains the primary entry-point for applications interacting with the platform. Multiple instances of the API Gateway can be created (currently in different geographical areas, depending on regulatory restrictions based on the type of processing) and dynamically routed to through a region-aware router. Rather than using GeoIP and DNS as a basis for routing (as popularized by e.g. Amazon's Route 53 DNS Service), we rely on the client-side to determine its own location based off of available vehicle location data. The region router, then, accepts a request header with an encoded ISO 3166-1 country code (optionally derived from a latitude/longitude pair handed off to a reverse geocoding service by the client), which is used as a basis for server discovery via a region-aware service discovery mechanism.

While this current approach is adequate for load balancing, ensuring regulatory compliance, and providing improved QoS for the end-user, the current system is still a centralized deployment model in which the bulk of the underlying business logic and processing is carried out in the Cloud instance. In order to support the phased deployment evolution identified in Figure 13, and to better support the kinds of services being developed for this use case, the architecture must support hierarchical deployments and clustering over which dynamic orchestration decisions and sufficiently granular data analyses can be made.

4.3.2.4 Edge Gateway

Following the planned deployment evolution from Figure 13, a self-contained instance of the Gateway will be created at the network Edge (integrated within the vehicle itself) in order to support further development and experimentation, which will be communicated with by the mobile app on the end-user's smartphone (or later, directly through the infotainment head unit).

Owing to the ML and CV requirements for some of the UC services, the Edge Gateway itself will be evaluated on two different hardware configurations, as noted in Table 27 below:

	Raspberry Pi 3B+	**NVIDIA Jetson Nano**
CPU	1.4 GHz 64-bit Quad-Core ARM Cortex-A53	1.4 GHz 64-bit Quad-Core ARM Cortex-A57 MPCore
GPU	Broadcom VideoCore IV	128-Core NVIDIA Maxwell
RAM	1GB LPDDR2	4GB LPDDR4

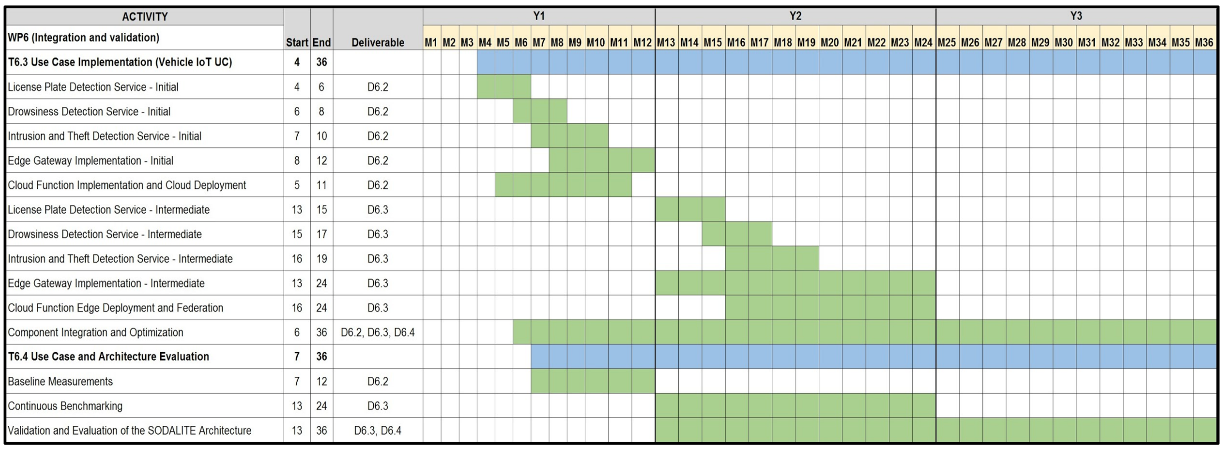
Performance	21.4 GFLOPS	472 GFLOPS
Cameras	Raspberry Pi Camera Module V2 (Sony IMX219 8-megapixel sensor) Raspberry Pi Pi NoIR Camera V2 - infrared version of the V2 Camera Module	
Networking	WiFi: 802.11ac, Bluetooth: 4.2	

Table 27: Planned hardware configurations for Edge Gateway (Vehicle IoT UC)

Precise requirements for the Edge Gateway will vary based on the capabilities of the vehicle and the individual preferences of the driver, services that will be delivered, and the way in which they are provisioned will be continually subject to change. Furthermore, local instances will need to be periodically updated when e.g. ML model updates are made available by a training pipeline. Rather than using a fixed configuration for each Edge Gateway instance, the aim of the Vehicle IoT UC is to maintain a minimal level of state within the Edge Gateway while shipping services from the Cloud as locally runnable functions.

During Y1, the Edge Gateway will be instantiated with a Cloud Function controller (the precise technology of which is still to be evaluated - candidate technologies include OpenFaaS and OpenWhisk), providing a mechanism for remote function creation and delivery to the Edge. Existing microservices with less stringent latency requirements will be adapted and deployed as Cloud functions during this time, further simplifying deployment complexity in preparation for Edge deployment and orchestration. During Y2, this will be expanded to include federation between the Edge Gateway and its immediate hierarchical parents (e.g. at the Fleet or Cloud level).

4.3.3 Implementation plan: timeline



ACTIVITY				Y1												Y2												Y3											
WP6 (Integration and validation)	Start	End	Deliverable	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
T6.3 Use Case Implementation (Vehicle IoT UC)	**4**	**36**					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
License Plate Detection Service - Initial	4	6	D6.2				🟩	🟩	🟩																														
Drowsiness Detection Service - Initial	6	8	D6.2						🟩	🟩	🟩																												
Intrusion and Theft Detection Service - Initial	7	10	D6.2							🟩	🟩	🟩	🟩																										
Edge Gateway Implementation - Initial	8	12	D6.2								🟩	🟩	🟩	🟩	🟩																								
Cloud Function Implementation and Cloud Deployment	5	11	D6.2					🟩	🟩	🟩	🟩	🟩	🟩	🟩																									
License Plate Detection Service - Intermediate	13	15	D6.3													🟩	🟩	🟩																					
Drowsiness Detection Service - Intermediate	15	17	D6.3															🟩	🟩	🟩																			
Intrusion and Theft Detection Service - Intermediate	16	19	D6.3																🟩	🟩	🟩	🟩																	
Edge Gateway Implementation - Intermediate	13	24	D6.3													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Cloud Function Edge Deployment and Federation	16	24	D6.3																🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Component Integration and Optimization	6	36	D6.2, D6.3, D6.4						🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
T6.4 Use Case and Architecture Evaluation	**7**	**36**								🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Baseline Measurements	7	12	D6.2							🟩	🟩	🟩	🟩	🟩	🟩																								
Continuous Benchmarking	13	24	D6.3													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
Validation and Evaluation of the SODALITE Architecture	13	36	D6.3, D6.4													🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 20: Gantt diagram of the development timeline of the Vehicle IoT use case components

The Vehicle IoT UC timeline highlighted in Figure 20 above is structured into 3 distinct phases to align with the project year. Y1 focuses on the development of basic functionality, primarily focused on the use of a centralized Cloud backend with some basic functionality pushed down to the Edge providing a baseline both in terms of measurements and of functionality to further build upon in Y2 and Y3. Y2 builds on the components developed during Y1, allowing these to be iteratively enhanced and shifted to the Edge, while laying the groundwork for tighter coupling with SODALITE components in Y3. Use case-specific component development is expected to wrap up at the end of Y2, allowing for Y3 to focus primarily on the integration and continued optimization of the use case using SODALITE technologies developed in the technical work packages.

Based on model re-use between some of the Vehicle IoT UC components, the implementation of components is carried out sequentially, with the basic support services (License Plate Detection, Drowsiness Detection, and Intrusion and Theft Detection) kicking off in M4 and wrapping up in M10. Experimentation with Cloud Functions, including the adaptation and deployment of developed components, will continue in parallel - beginning in M5 and concluding in M11. The results of the Cloud Function experimentation and initial component definition will form the basis of the initial Edge Gateway implementation at the end of M12.

Y2 follows a similar pattern as Y1 - allowing for each use case component to be iterated through and functionally enhanced, beginning in M13 and wrapping up in M19. The main focus during this period will be in the development of the Edge Gateway, the integration of Cloud Functions, and the ability to federate Cloud function controllers, thereby allowing for Cloud-to-Edge/Edge-to-Cloud orchestration to impact deployment at the function level and to support the kind of adaptation flexibility required by the use case.

Y3 will be focused primarily on integrating SODALITE-developed technologies with the use case components. In parallel continuous measurement and optimization of components will be carried out, providing the basis for practical validation and evaluation of the SODALITE architecture and technologies within the use case application context.

4.4 SODALITE Platform Coverage

In Section 2 (Requirements) of D2.1 "Requirements, KPIs, evaluation plan and architecture - First version", we introduced a number of SODALITE UML use cases and provided sequence diagrams for the implementation of each case. These UML use cases will be run as part of the evaluation of the SODALITE platform. Table 28 (reproduced from D2.1) shows the planned coverage of the UML use cases by the SODALITE demonstrating use cases (defined in this document), and when the intermediate and final versions of the UML cases will be ready to be tested by the demonstrating use case owners. The table also includes a column concerning the testbed providers. In fact, they will act as Resource Experts and as Quality Experts and will therefore experiment with the UML use cases associated to these two roles. Demonstrating use case owners, instead, will act mostly as Application Ops Experts and will all test the core UML cases. The UML use cases concerning bug prediction, selection of specific resources and optimization will be tested by the respective demonstrating use case that has specific concerns in the corresponding area.

UML Use Case	Virtual clinical trial	SNOW	Vehicle IoT	Testbed providers	Released at
UC1 Define Application Deployment Model (WP3)	X	X	X		M12, M18, M24
UC2 Select Resources (WP3)	X				M12, M18, M24
UC3 Generate IaC code (WP4)	X	X	X		M12, M18, M24
UC4 Verify IaC (WP4)	X	X	X		M12, M18, M24
UC5 Predict and Correct Bugs (WP4)	X				M12, M18, M24
UC6 Execute Provisioning, Deployment and Configuration (WP5)	X	X	X		M12, M18, M24, M33
UC7 Start Application (WP5)	X	X	X		M12, M18, M24, M33
UC8 Monitor Runtime (WP5)	X	X	X		M12, M18, M24, M33
UC9 Identify Refactoring Options (WP5)	X	X	X		M18, M24, M30, M33
UC10 Execute Partial Redeployment (WP5)	X	X	X		M18, M24, M33
UC11 Define IaC Bugs Taxonomy (WP4)				X	M12, M18
UC12 Map Resources and Optimisations (WP3)	X	X	X	X	M12, M24
UC13 Model Resources (WP3)		X		X	M12, M22, M30
UC14 Estimate Quality Characteristics of Applications and Workload (WP3)			X	X	M18, M24, M33
UC15 Statically Optimize Application and Deployment (WP4)	X	X	X		M18, M30
UC16 Build Runtime images (WP4)	X	X	X		M12, M18, M24

Table 28: Planned coverage of the SODALITE UML use cases by the project's demonstrating use cases

5 Conclusions

In this deliverable, the implementation plan of the SODALITE platform and use cases was presented. The document's goal is to guide, together with deliverable D2.1, the project developments in order to transform the SODALITE scientific and technological results into a unified platform with running services and tools. Three iterations of the SODALITE platform are envisioned, one for each year of the project. By the end of the first year, the initial implementation of the basic components forming the SODALITE platform will be provided. In parallel, the first iteration of the development of the Demonstrating Use Cases will be performed. During the second project year, the focus will be on component integration, delivery of more advanced features, as well as the initial evaluation of the improvement provided by the SODALITE platform for the Demonstrating Use Cases. Finally, during the third year of the project, iterative measurements of the results produced by the SODALITE platform will be taken and based on these measurements, additional improvements will be applied to the SODALITE system.

It easily follows that the requirements stemming from WP2 will evolve over time and as a result, the SODALITE requirement catalogue will be updated based on the main iterations within the project. Consequently, it is expected that small adaptations/updates on the initial SODALITE platform and use cases implementation plan will be required in order to meet the final project requirements. Any such adaptations/updates will be reported in future deliverables D6.2, D6.3, and D6.4, "Implementation and evaluation of the SODALITE platform and use cases".

Appendix A

As already described in Section 4 of the document, the Appendix provides the specific requirements for the three demonstrating use cases of SODALITE, extracted by the use case owners during the first iteration of requirements elicitation within WP2. Please note that the “Rationale” field has not been filled out for every requirement, as it was optional for the use case owners to provide information in this regard.

A1. POLIMI Snow UC

Id.	**Title**	**Description**	
SNOW.R1	UGIC throughput	The user generated image crawler should acquire images respecting the query limit of the API	
Rationale			**Scope**
The Snow UC starts with either the acquisition of user generated content or of images from fixed-position web cams. Maximum throughput is limited by the admitted frequency acquisition of the sources.			Runtime
Id.	**Title**	**Description**	
SNOW.R2	UGIC parallelism	The user generated image crawler should query sub-regions in parallel	
Rationale			**Scope** Runtime
Parallel acquisition increases input throughput.			
Id.	**Title**	**Description**	
SNOW.R3	UGIC API key	The user generated image crawler of each sub-region should use a different API key	
Rationale			**Scope** Runtime
This technique is used to increase parallelism by using multiple clients.			
Id.	**Title**	**Description**	

SNOW.R4	UGIC locality	Despite distribution, the images of a sub-region should be stored in the same logical space	
Rationale			**Scope** Runtime
Parallelism may increase the overall throughput, but the data set should be accessed uniformly w.r.t. the region of persinence of images.			
Id.	**Title**	**Description**	
SNOW.R5	UGIC scheduling	The user generated image crawler should execute at a predefined interval (e.g., once per week) at scheduled time	
Rationale			**Scope** Runtime
In this way, images are sampled with known frequency.			
Id.	**Title**	**Description**	
SNOW.R6	MRC throughput2	The mountain relevance classifier should classify 100 image per second	
Rationale			**Scope** Runtime
MRC should execute fast to discard irrelevant data quickly.			
Id.	**Title**	**Description**	
SNOW.R7	MRC HW1	The SIFT feature extraction could be accelerated exploiting the GPU	
Rationale			**Scope** Runtime
The present implementation does not exploit GPU computation, which is a desirable improvement.			
Id.	**Title**	**Description**	
SNOW.R8	MRC HW2	The SVM classification could be accelerated exploiting the GPU	

Rationale	Scope
The present implementation does not exploit GPU computation, which is a desirable improvement.	Runtime

Id.	Title	Description
SNOW.R9	MRC parallelism1	The mountain relevance classifier should batch images choosing the batch size based on the conditions of the computation infrastructure: size of the batch, available transfer bandwidth, number of available GPU nodes.

Rationale	Scope
Batching may improve the throughput, by making better use of classifier instances.	Runtime

Id.	Title	Description
SNOW.R10	MRC parallelism2	The mountain relevance classifier should extract image features in parallel to SVM classification

Rationale	Scope
The two processes are independent, and their parallelization could improve end-to-end performance.	Runtime

Id.	Title	Description
SNOW.R11	MRC availability	The mountain relevance classificator should be triggered by the availability of (a batch of) images to classify

Rationale	Scope
This solicits the capability of handling application-defined events in the deployed architecture.	Runtime

Id.	Title	Description
SNOW.R12	WIC throughput	The webcam crawler should crawl up to 1 image per minute per webcam

Rationale	Scope

Web cams have variable frame update frequency, which should be normalized at 1 minute for regularizing acquisition.	Runtime

Id.	Title	Description
SNOW.R13	WIC hw	The webcam crawler should have access to distributed storage system with guarantees of data replication and high availability.

Rationale	Scope
Loss of images would harm the temporal continuity of the data set, so high availability is important.	Runtime

Id.	Title	Description
SNOW.R14	WIC parallelism1	The crawler of each webcam should be executed in parallel

Rationale	Scope
Parallel acquisition improves input throughput.	Runtime

Id.	Title	Description
SNOW.R15	WIC parallelism2	The crawler of each webcam could be executed on different machines but should save all the daily images of a webcam in the same logical space

Rationale	Scope
Images should be accessible per web cam / location easily.	Runtime

Id.	Title	Description
SNOW.R16	WIC availability1	The webcam crawler should run in day time only (6am - 19pm)

Rationale	Scope
Night images cannot be processed.	Runtime

Id.	Title	Description

SNOW.R17	WIC availability2	The webcam crawler should issue an alert if webcam is down
Rationale		**Scope**
For temporal continuity of the data set, webcam downtime should be minimized.		Runtime
Id.	**Title**	**Description**
SNOW.R18	WIC availability3	The failure of a webcam crawler node should be signalled and an equivalent process re-instantiated to avoid losing unrecoverable data
Rationale		**Scope**
For temporal continuity of the data set, webcam crawling downtime should be minimized.		Runtime
Id.	**Title**	**Description**
SNOW.R19	WIC availability4	The crawled images should be replicated against disk failures
Rationale		**Scope**
No way to re-acquire them, if lost.		Runtime
Id.	**Title**	**Description**
SNOW.R20	WFC throughput	The weather filter should process up to 100 images per second
Rationale		**Scope**
Discarding irrelevant data as quickly as possible is important.		Runtime
Id.	**Title**	**Description**
SNOW.R21	WFC HW	Edge extraction could be accelerated exploiting the GPU
Rationale		**Scope**

The present implementation does not exploit the GPU, which is a relevant improvement.		Runtime
Id.	**Title**	**Description**
SNOW.R22	WFC parallel1	The weather filter should filter images of different webcams in parallel
Rationale		**Scope**
This would improve the throughput for downstream processing.		Runtime
Id.	**Title**	**Description**
SNOW.R23	WFC parallel2	The weather filter of each webcam could be executed on different machines but should save images in the same logical space where the corresponding webcam images reside
Rationale		**Scope**
Images should be easily accessible per webcam and location.		Runtime
Id.	**Title**	**Description**
SNOW.R24	WFC availability	The weather filter should be triggered by the availability of (a batch of) webcam images
Rationale		**Scope**
This solicits the capability of the deployed architecture to provide application based triggers.		Runtime
Id.	**Title**	**Description**
SNOW.R25	DMIA throughput	The DMI aggregator should process up to 100 images per second
Rationale		**Scope**
To increase the throughput for downstream processing.		Runtime

Id.	Title	Description
SNOW.R26	DMI - webcams	The DMI of different days for different webcams should be calculated in parallel

Rationale	Scope
As computation is independent, this improves throughput.	Runtime

Id.	Title	Description
SNOW.R27	DMI - aggregators	The DMI aggregator should batch images to aggregate, choosing a batch size based on the conditions of the computing infrastructure

Rationale	Scope
Batching can better load the available DMI aggregator instances.	Runtime

Id.	Title	Description
SNOW.R28	DMI - storage	Despite distribution, the DMI aggregator should store the median image in the same logical space as the daily data series it comes from

Rationale	Scope
DMI should be easily accessible by webcam and location.	Runtime

Id.	Title	Description
SNOW.R29	DMI - trigger	The DMI aggregator should be triggered by the availability of a batch of images

Rationale	Scope
This solicits the capability of the deployed architecture to handle application defined triggers.	Runtime

Id.	Title	Description
SNOW.R30	MIGR-SE throughput	The skyline extractor module should process up to 50 images per second

Rationale	Scope
This ensures the possibility of scaling the number of web cams.	Runtime

Id.	Title	Description
SNOW.R31	MIGR-SE hardware	The execution of the DL model of the skyline extraction could be accelerated exploiting the GPU

Rationale	Scope
The present implementation does not exploit the GPU, which is a relevant improvement.	Runtime

Id.	Title	Description
SNOW.R32	MIGR-SE parallelism	The execution of the DL model of the skyline extraction should be applied one image at a time, but different GPUs could run more than one instance

Rationale	Scope
The present implementation does not exploit allocation to multiple GPUs, which is a relevant improvement.	Runtime

Id.	Title	Description
SNOW.R33	MIGR-SE scheduling	The skyline extraction should be computed on demand

Rationale	Scope
This solicits the deployed architecture to support user-defined triggers.	Runtime

Id.	Title	Description
SNOW.R34	MIGR-SE accessibility	The skyline extraction should be exposed as a service callable on the internet

Rationale	Scope
Skyline extraction could be exploited for alternative purposes, also by third party applications.	Runtime

Id.	Title	Description
SNOW.R35	MIGR-SE throughput	The skyline extraction should be computed in real time (e.g., 100ms)

Rationale	Scope
This would enable to use skyline extraction also in alternative applications, e.g, based on a mobile phone interface.	Runtime

Id.	Title	Description
SNOW.R36	MIGR-SE availability	The skyline extraction must have high availability

Rationale	Scope
Skyline extraction determines the computation of snow indexes, which should be maximally continuous in time and space	Runtime

Id.	Title	Description
SNOW.R37	MIGR-360PG throughput	The panorama generator should compute 10 panoramas per second

Rationale	Scope
Panorama computation influences the alignment time, which is essential for image geo-registration.	Runtime

Id.	Title	Description
SNOW.R38	MIGR-360PG hardware1	The panorama generator should be accelerated exploiting the GPU (for the DEM render generation).

Rationale	Scope
The present implementation does not exploit the GPU, which is a relevant improvement.	Runtime

Id.	Title	Description

SNOW.R39	MIGR-360PG Availability1	The panorama generator should be computed on demand	
Rationale			**Scope**
This solicits the deployed architecture to support user-defined triggers.			Runtime
Id.	**Title**	**Description**	
SNOW.R40	MIGR-360PG Availability2	The panorama generator should be exposed as a service callable on the internet	
Rationale			**Scope**
For use by third party applications.			Runtime
Id.	**Title**	**Description**	
SNOW.R41	MIGR-360PG Availability3	The panorama generator should be computed in real time (e.g., <50ms)	
Rationale			**Scope**
Panorama generation impacts image geo-registration, necessary for downstream processing.			Runtime
Id.	**Title**	**Description**	
SNOW.R42	MIGR-360PG Availability4	The panorama generator must have high availability	
Rationale			**Scope**
Failures block image geo-registration and downstream processing.			Runtime
Id.	**Title**	**Description**	
SNOW.R43	MIGR-PA throughput	The peak aligner should process 20 images per seconds	
Rationale			**Scope**

Alignment conditions image geo-registration, necessary for snow index contextualization.		Runtime
Id.	**Title**	**Description**
SNOW.R44	MIGR-PA Parallelism	The peak aligner should process images in parallel
Rationale		**Scope**
Parallelism would improve throughput for downstream processing.		Runtime
Id.	**Title**	**Description**
SNOW.R45	MIGR-PA throughput	The peak aligner should be computed in real time (e.g. <20ms)
Rationale		**Scope**
Fast alignment is required for scaling the number of webcam images.		Runtime
Id.	**Title**	**Description**
SNOW.R46	MIGR-PA Availability1	The peak aligner should be computed on demand
Rationale		**Scope**
This solicits the deployed architecture to support user-defined triggers.		Runtime
Id.	**Title**	**Description**
SNOW.R47	MIGR-PA Availability2	The peak aligner should be exposed as a service callable on the internet
Rationale		**Scope**
For use by third party applications.		Runtime
Id.	**Title**	**Description**
SNOW.R48	MIGR-PA Availability4	The peak aligner must have high availability

Rationale	Scope
Alignment conditions snow index calculations.	Runtime

Id.	Title	Description
SNOW.R49	SMC throughput	The snow mask calculator should process 20 images per second

Rationale	Scope
SMC fast computation is needed to scale the number of webcam images.	Runtime

Id.	Title	Description
SNOW.R50	SMC throughput	The snow mask calculator could be accelerated exploiting the GPU to run the SVM classifier

Rationale	Scope
The present implementation does not exploit the GPU, which is a relevant improvement.	Runtime

Id.	Title	Description
SNOW.R51	SMC throughput	The snow mask calculator could be accelerated exploiting the GPU (requires re-implementing the classifier using CNN)

Rationale	Scope
The present implementation does not exploit the GPU, which is a relevant improvement.	Runtime

Id.	Title	Description
SNOW.R52	SMC parallel	The snow mask calculator should process images in parallel

Rationale	Scope
This increases output throughput.	Runtime

Id.	Title	Description

SNOW.R53	SMC availability	The snow mask calculator should be triggered offline for a batch of image+mountain mask pairs	
Rationale			**Scope**
This solicits the deployed architecture to support application defined triggers.			Runtime
Id.	**Title**	**Description**	
SNOW.R54	SIC throughput	The snow index calculator should process 20 images processed by second	
Rationale			**Scope**
This is needed for scaling the number of webcam images.			Runtime
Id.	**Title**	**Description**	
SNOW.R55	SIC hw	The snow index calculator could be accelerated by exploiting the GPU	
Rationale			**Scope**
The present implementation does not exploit the GPU, which is a relevant improvement.			Runtime
Id.	**Title**	**Description**	
SNOW.R56	SIC parallel	Different images can be processed in parallel	
Rationale			**Scope**
This would support high output throughput.			Runtime
Id.	**Title**	**Description**	
SNOW.R57	SIC availability	The snow mask calculator should be triggered offline for a batch of snow+mountain mask pairs	
Rationale			**Scope**

This solicits the capability of the deployed architecture to support user-defined triggers.	Runtime

A1.1 POLIMI Snow UC - Domain assumptions

Id.	**Title**	**Description**
SNOW.D1	MIGR-360PG hardware2	If the GPUs are NVIDIA, the drivers installed should be Open-Source Nouveau.
Rationale		**Scope**
		Application Container
Id.	**Title**	**Description**
SNOW.D2	MIGR-360PG parallelism	The panorama generator should execute in parallel for different images
Rationale		**Scope**
		Runtime
Id.	**Title**	**Description**
SNOW.D3	MIGR-360PG storage	The panorama generator should allocate DEM data as follows: SPACE: 71GB (DEM3 World coverage), 12GB (DEM 1 Alps coverage), 638GB (DEM1 World coverage | recently released)
Rationale		**Scope**
		Application Container

A2. USTUTT Virtual Clinical Trial UC

Id.	**Title**	**Description**

VCT.R1	Deployment of a storage system	SODALITE should support the deployment of a storage system able to fulfil the requirements of the virtual clinical trials case study in terms of I/O speed	
Rationale			**Scope**
The simulation chain starts with input data and subsequently data is produced at every step in the chain which serves as input to the next step. This data needs to be stored and accessed. The patient data may be located in external data repositories.			Runtime
Id.	**Title**	**Description**	
VCT.R2	MPI	SODALITE must support DevOps team in deploying their applications components on an MPI (Message Passing Interface)-compliant HPC system	
Rationale			**Scope**
Some parts of the simulation chain like the Code Aster solver rely on MPI to achieve parallelism. So MPI must be supported by the SODALITE framework.			Runtime
Id.	**Title**	**Description**	
VCT.R3	Data Parallelism	Start multiple instances of a program each with a different portion of the input data.	
Rationale			**Scope**
Independent input data can be handled independently by multiple instances of a process in the simulation chain, such that data parallelism can be employed as an optimization step.			Application Optimiser
Id.	**Title**	**Description**	
VCT.R4	IDE Data Parallelism	Specify which parts of an application use data parallelism.	
Rationale			**Scope**
The IDE should enable data parallelism modelling (see requirement VCT.R3).			Application Developer Editor
Id.	**Title**	**Description**	

VCT.R5	Different Runtime Environments	Support different runtime environments.	
Rationale			**Scope**
The different components of the application are written in different programming languages and need different runtime environments. So far. mostly C/C++, Fortran and Python are used. In addition, some implementation of MPI (preferably OpenMPI) must be available for some components. It should also be possible to add more if needed (e.g. JVM).			Runtime
Id.	**Title**	**Description**	
VCT.R6	Automatic resource allocation	Automatically allocate, configure and start storage and compute nodes for components of the simulation chain.	
Rationale			**Scope**
Automation in resource provisioning will decrease deployment time, as well as enable a reusable configuration.			Runtime
Id.	**Title**	**Description**	
VCT.R7	Extraction process	SODALITE must support the deployment and configuration of the Extraction component on a vCPU	
Rationale			**Scope**
Extraction process is a single core problem.			Runtime
Id.	**Title**	**Description**	
VCT.R8	Discretization process	Discretizing and generating a mesh requires 1-2 vCPUs and 3-4 GB RAM	
Rationale			**Scope**
Discretization process is more or less a single core problem.			Runtime
Id.	**Title**	**Description**	

VCT.R9	Parallel Material Mapper	Support parallel execution within parts of the simulation chain. This will probably require 1-2 HPC nodes or 1-2 HPVMs (High Performance Virtual Machines). This requirement is yet to be determined.	
Rationale			**Scope**
In the probabilistic approach to mapping density to elasticity, the computations for the cells within the geometry can be executed independently of each other. A parallel workflow is needed to speed up the overall computation time.			Runtime
Id.	**Title**	**Description**	
VCT.R10	Solver Process	Solver must be run on 2 HPC nodes with the characteristics defined in the testbed description	
Rationale			**Scope**
To the best of our knowledge, solving using finite element methods is computationally intensive and requires at least 2 HPC nodes to be run on.			Runtime
Id.	**Title**	**Description**	
VCT.R11	Fast interconnect	Support fast networking performance	
Rationale			**Scope**
The intercommunication between processes deals with large dataset (messages), therefore the communication between the processes must not bottleneck the overall performance.			Runtime
Id.	**Title**	**Description**	
VCT.R12	Fast storage	Support fast I/O performance	
Rationale			**Scope**
The read and write operations on the storage devices must perform well with large dataset.			Runtime
Id.	**Title**	**Description**	
VCT.R13	MTU (Maximum Transmission Unit) size	Support for jumbo frames	

Rationale	Scope
To boost communication performance, jumbo frames must be supported on the underlying network infrastructure to transmit larger messages, e.g. switches/routers.	Runtime

Id.	Title	Description
VCT.R14	Data Model Persistence property	The IDE must provide the persistence property of the data model, such that the data will be stored permanently or temporarily

Rationale	Scope
The lifetimes of the data that is produced at various stages of the simulation differs. Some needs to be stored only temporarily, some needs to be stored for the whole simulation process, some needs to even outlive the simulation.	Application Developer Editor

Id.	Title	Description
VCT.R15	Establish the relationship between data models and process models	IDE must provide a way to establish a relationship between data model and a process model

Rationale	Scope
It is important for the simulation chain to map the input and output data that are consumed and produced by the respective processes in the chain.	Application Developer Editor

Id.	Title	Description
VCT.R16	Incorporating third party components	IDE must provide a way to specify third party components (programs/algorithms) that act on data. This can either take the form of the application model itself or it can be a service model/container to be specified.

Rationale	Scope
Due to the complexity of describing certain computational tasks in terms of models (modelling functional requirements of application) it must be possible to use existing software packages/libraries.	Application Developer Editor

Id.	Title	Description
VCT.R17	Geometry Surface Data Model	IDE must provide a data model that represents the geometry surface of the vertebral bodies. It must have a "temporary" persistence property.

Rationale	Scope
See requirement VCT.R14.	Application Developer Editor

Id.	Title	Description
VCT.R18	Send produced data to temporal storage	IDE must provide a way to specify that the output data from this step needs to be stored until the discretization component (yet to be specified) has finished.

Rationale	Scope
For each step in the chain, the data that is produced needs to live at least until the next step has finished. This is because all data produced in one step are consumed in the next step.	Application Developer Editor

A2.1 USTUTT Virtual Clinical Trial UC - Domain assumptions

Id.	Title	Description
VCT.D1	MCA Model	Within the extraction component some implementation of MCA (Marching Cubes Algorithm) needs to be available as a third-party model/service.

Rationale	Scope
See requirement VCT.R16.	Application Developer Editor

A3. ADPT Vehicle IoT UC

Id.	Title	Description
VIoT.R1	Standards compliance	Platform must be compliant with ISO 20078

Rationale		Scope
ISO 20078 (the Extended Vehicle) provides clear requirements for platforms managing Connected Car data, resources access, and service deployment which the use case must ultimately be capable of satisfying.		Runtime, Use Case Implementation
Id.	**Title**	**Description**
VIoT.R2	Multi-arch Container Deployment & Orchestration	Platform must be able to support deployment and orchestration across multi-arch container images.
Rationale		**Scope**
The Vehicle IoT UC involves deployment of containerized components onto systems with different CPU architectures (specifically - x86_64, armhf, arm64). This scenario is prepared for by providing multi-arch Docker images through the Docker manifest - SODALITE orchestration should be able to match the appropriate image to the deployment target.		Runtime
Id.	**Title**	**Description**
VIoT.R3	GPU Acceleration for Online ML model training	Platform should make use of (dynamically available) GPU resources for accelerated training of ML models by the use case, this may include resources both in the Cloud and at the Edge.
Rationale		**Scope**
A number of ML pipelines used by the use case depend on CV application, which can be accelerated with GPUs. GPU resources may exist at various times at various locations in the deployment (e.g. in the Cloud backend, or in the Edge Gateway) which may be used.		Runtime
Id.	**Title**	**Description**
VIoT.R4	Cloud Function Deployment & Orchestration from Cloud-to-Edge	Platform must support deployment and orchestration of cloud functions from Cloud-to-Edge
Rationale		**Scope**
Services that support the use case will be deployed as dedicated cloud functions, which may at various times need to be deployed and activated at different hierarchical levels (Cloud backend, Edge Gateway, etc.).		Runtime
Id.	**Title**	**Description**

VIoT.R5	Encrypted Data Storage & Analytics	Platform must be able to store and operate on large encrypted data sets based on sensitive data, including personal data, vehicle telemetry, etc.	
Rationale			**Scope**
GDPR requirements and DPA guidelines necessitate encryption at rest and in-processing			Runtime
Id.	**Title**	**Description**	
VIoT.R6	Encrypted / Sensitive Data Storage Classification	The IDE must provide a mechanism by which a developer is able to define data as sensitive/non-sensitive for subsequent data-at-rest/data-in-processing encryption.	
Rationale			**Scope**
GDPR requirements and DPA guidelines necessitate encryption at rest and in-processing (further provides support for the implementation of VIoT.R5)			Application Developer Editor

A3.1 ADPT Vehicle IoT UC - Domain assumptions

Id.	**Title**	**Description**
VIoT.D1	Cloud Backend<->Edge Gateway Connectivity	The Cloud backend and Edge gateways must be able to communicate with each other.
Rationale		**Scope**
The Edge gateway will implement a subset of functionality available from the Cloud backend, and will further defer to the Cloud backend for certain operations (as well as to synchronize state across instances). The Cloud backend, in turn, must be able to access the Edge gateways in order to reconfigure and deploy Edge-based services, while also providing a central point for e.g. fleet-wide analytics across multiple Edge instances in later stages of the project.		Runtime
Id.	**Title**	**Description**
VIoT.D2	Multi-arch Container Deployment & Orchestration	Container orchestration and deployment is able to handle x86_64, armhf, and arm64 target architectures.
Rationale		**Scope**

See VIoT.R2	Runtime

References

[1] https://protege.stanford.edu/
[2] http://graphdb.net/
[3] https://www.w3.org/TR/sparql11-query/
[4] https://www.eclipse.org/Xtext/
[5] https://www.eclipse.org/
[6] https://wiki.eclipse.org/Xpand
[7] https://www.eclipse.org/acceleo/
[8] https://wiki.eclipse.org/Orion
[9] https://ace.c9.io/
[10] https://codemirror.net/
[11] https://dslforge.org/
[12] https://www.eclipse.org/sirius/
[13] https://www.eclipse.org/graphiti/
[14] https://www.cresta-project.eu/
[15] https://www.maestro-data.eu/
[16] https://www.ansible.com/
[17] https://www.chef.io/ansible/
[18] https://www.docker.com/
[19] https://sylabs.io/singularity/
[20] https://hpc.github.io/charliecloud/index.html
[21] https://github.com/eth-cscs/sarus
[22] https://github.com/xlab-si/xopera-opera
[23] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[24] http://skydive.network/
[25] https://prometheus.io/
[26] https://grafana.com/
[27] https://wp.cloudify.co/
[28] https://puppet.com/
[29] https://github.com/ari-apc-lab/croupier
[30] http://www.adaptivecomputing.com/products/torque/
[31] https://slurm.schedmd.com/
[32] https://github.com/TANGO-Project/alde
[33] https://www.openstack.org/
[34] https://docs.openstack.org/nova/latest/
[35] https://wiki.openstack.org/wiki/Ironic
[36] https://wiki.openstack.org/wiki/Cinder
[37] https://wiki.openstack.org/wiki/Neutron
[38] https://kubernetes.io/
[39] https://kubernetes.io/docs/concepts/workloads/pods/pod/
[40] https://kubernetes.io/docs/concepts/services-networking/service/
[41] http://www.adaptivecomputing.com/products/torque/
[42] http://www.adaptivecomputing.com/moab-hpc-basic-edition/
[43] http://docs.adaptivecomputing.com/maui/
[44] https://www.mikelangelo-project.eu/technology/vtorque-virtualization-support-for-torque/
[45] https://www.mikelangelo-project.eu/
[46] https://jenkins.io/
[47] See www.vtk.org and www.itk.org
[48] https://www.sciencedirect.com/science/article/pii/S004578250900084X