Sodalite

SOftware Defined AppLication Infrastructures managemenT and Engineering

Guidelines for Contributors to the SODALITE Framework

D2.4

POLIMI

31/01/2020

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

Deliverable data			
Deliverable	D2.4 - Guidelines for Contributors to the SODALITE Framework		
Authors	Luciano Baresi (POLIMI), Elisabetta Di Nitto (POLIMI), Kamil Tokmakov (USTUTT), Anastasios Karakostas (CERTH), Stefanos Vrochidis (CERTH), Dragan Radolović (XLAB)		
Reviewers	Georgios Meditskos (CERTH), Indika Kumara (UVT)		
Dissemination level	Public		
History of changes	Name	Change	Date
	First complete version	-	15/01/2020
	Final version	Document modified and improved based on internal review	31/01/2020
	New final version	New final version after project review	31/07/2020

Acknowledgement

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

Table of Contents

List Of Images

List Of Tables

Executive Summary

This deliverable presents the structure of the SODALITE organization on GitHub (https://github.com/SODALITE-EU) and the open source repositories adopted for the development of the SODALITE components. Moreover, it introduces external contributors to the rules and steps to be followed to participate in the SODALITE effort. As highlighted in the deliverable, the project is seeking contributions at various levels, ranging from using the offered tools to highlighting bugs and extension possibilities to contributing to the code of a specific component. This deliverable is an important element of the SODALITE documentation and will be evolved based on the development of the project.

1. Introduction

SODALITE exploits an open-source approach to let developers meet users and allow the latter to contribute to the solutions carried out by the project. To this end, this document sets some rules and guidelines on how contributors can support the development of the SODALITE platform, how the SODALITE software is organized in various related open source repositories, and the practical procedures to be used to propose changes and to see them accepted by the SODALITE team.
Consistently to these goals, this document provides a centralized and detailed presentation of how SODALITE aims to create, maintain, and manage the open-source communities behind the project. Version and configuration management in SODALITE is heavily based on GitHub. The project plans to adopt all the tools it offers besides version management; GitHub will also be used for tracking issues and requests.
The document provides the inspiring motivations for the choices presented in the document, along with the organization and rules adopted in the project. Section 2 provides some high-level guidelines to frame open-source communities and possible contributions. Section 3 summarises the architecture of the SODALITE framework, presents its organization in three layers and provides a short description of each SODALITE component together with the adopted technologies and the link to the corresponding GitHub repository. Section 4 clarifies how the repositories are structured. Section 5 presents the adopted license scheme. Section 6 defines the possible roles within the SODALITE contributing team. Section 7 defines the SODALITE internal development workflow, Section 8 provides some high-level guidelines for external contributors, Section 9 defines the high-level rules for software and release management within the consortium. Finally, Section 10 concludes this deliverable.

2. The Bazaar Approach as Driving Guidelines for Contribution

When one thinks of organizing an open source community, first of all, s/he must define a framework. To this end, Raymond [1] confronts two different approaches:

- The Cathedral model provides a more hierarchical solution where the source code is available at each software release. While the code developed within releases is only available to a limited group of developers (GNU Emacs and GCC are used as examples)
- The Bazaar model supports a more cooperative approach where the code is developed all together over the Internet. Linux Torvalds is presented as the inventor of this approach.

SODALITE aims to exploit the latter model. This approach is a bottom-up solution that privileges developers and users over managers and owners. The former is a more typical top-down approach, which emphasizes the role of management, and it is often the solution used by corporations. In contrast, the power of open-source projects comes from the contributors (the bases), who are in charge of deciding, shaping, implementing, and selecting. The key essence of open-source projects originates from the "collective intelligence" of developers and users, that is, from the community behind the project. Roles are not usually predefined, as with standard projects, but come from the community: key developers are also users, and leaders are often the developers who contribute most of the ideas and the code.

The creation of a proper community is thus key for the success of SODALITE as an open-source project. The first step towards it is that potential contributors become involved. As suggested by Peters and Ruff [2], a potential user must start skimming through the documentation and communication channels (e.g., mailing lists, forums, slack channels, repositories, and others) to know more, understand, and become familiar with the project and its participation modes. This is functional to **joining the community**. The next step is that the new member is supposed to spend time **reading and studying** how the project is organized and how the community works. The obvious consideration is that one must **learn norms and expectations** before being able to contribute. The more members understand, the more appropriate, and probably well-received, their contributions will be. Before any concrete contribution, however, it is key that the members become familiar with the governance behind the community: who makes decisions about the different types of contributions and how those decisions are made.

The last suggestion is to **start small**. If one starts fixing a small bug or document, can easily become familiar with processes and rules and correct mistakes as needed. The first contributions should not be on critical paths to avoid creating bottlenecks or additional problems. The higher the expertise becomes, the more critical and complex proposed contributions can be.

As said above, the actual contribution process varies from project to project. Some projects have rules and guidelines for everything: from coding style to formatting, and from bug/ticket numbering conventions to release dates. Some projects only accept contributions after signing a contributor agreement, others only require that patches are posted on dedicated mailing lists or simply as pull requests.

Submitting a contribution is only one side of the coin. The contributor must always remain available for clarifications, changes, and answers to feedback. The aim is to clarify how things work and why a specific solution was adopted, but also to improve or modify the contribution. These requests can be tough, and they should always come in the spirit of making the contribution better and of improving the overall quality of the project.

In a genuine open-source community, any comment or criticism must always be positive and there is no need of being defensive. A contribution may require different rounds of resubmission and feedback before being accepted, and in some cases the outcome could be negative. It is nothing personal, there could be many reasons why the contribution gets rejected. It is always a way to learn and understand more about the project, to help improve it, and to conceive better possible contributions in the future. It is also true that if a contribution is accepted, then it must be maintained for a long period.

3. SODALITE Architecture and the geography of the SODALITE open source repositories

The SODALITE platform is roughly organized in three layers (see Figure 1): Modeling Layer, Infrastructure as Code Layer, and Runtime Layer. Each layer is further decomposed into a number of different elements. Such decomposition has been initially defined in Deliverable D2.1 [3] and is evolving based on the current understanding gathered by all project partners.
The SODALITE software and team is organized around these three layers and is made available to external contributors through GitHub (github.com). In particular, we have created the SODALITE-EU organization (https://github.com/SODALITE-EU) which, at the time of writing, features three teams of committers, one for each layer of the architecture, and various repositories, one for each subcomponent or group of strictly interrelated subcomponents. The organization also maintains an additional repository which includes the overall documentation of the project. In the following subsections we provide a short overview of the various layers of the SODALITE architecture, we identify the components that are part of each layer and the corresponding repositories.

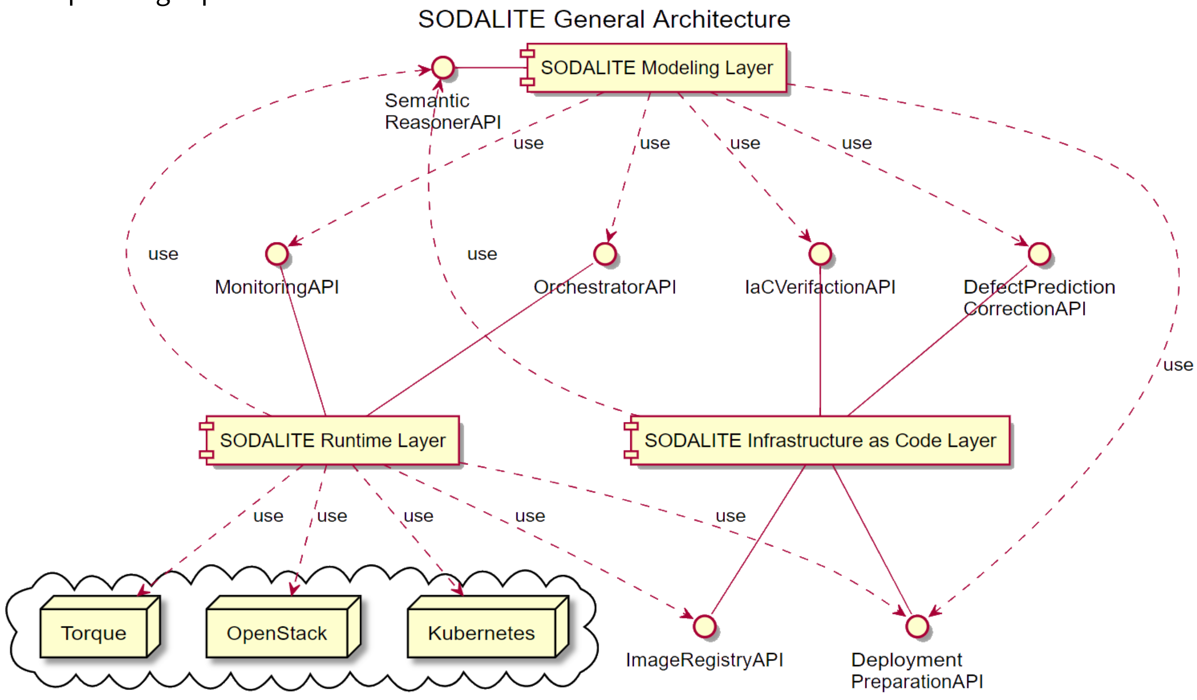

Figure 1. SODALITE high level architecture (from D2.1)

3.1. The SODALITE Modeling Layer

The Modeling Layer offers the tools to support all modeling activities by the SODALITE users. Its elements are the IDE offering smart editing features enhanced with suggestions that are generated, in a context-based fashion, by the Semantic Reasoner. This last component is reasoning on an extensible ontology, the Semantic Knowledge Base, that includes the main concepts needed to model a deployment configuration for a complex application. The following table lists these three components together with the main technologies they are based on and the GitHub repositories that include their code.

Table 1. Components of the Modeling Layer and corresponding repositories

Component	Main used technologies	GitHub repository

SODALITE IDE: The development environment offered to users. It supports modeling using the SODALITE Domain Specific Language (DSL) **License**: Apache2	**Programming language(s):** Java, Xtend **DBMS technology:** Eclipse workspace (filesystem) **Middleware:** Spring IO **UI/UX technology:** Eclipse, REST API	https://github.com/SODALITE-EU/ide
Semantic Reasoner (Knowledge Base Service - KBS): The component supporting semantic reasoning over the knowledge base **License**: Apache2	**Programming language(s):** Java **DBMS technology:** RDF triple store (GraphDB) **Middleware:** Apache Tomcat **UI/UX technology:** N/A	https://github.com/SODALITE-EU/semantic-reasoner
Semantic Knowledge Base (KB): The ontology that describes the main concepts needed to model an application deployment **License**: Apache2	**Programming language(s):** RDF/OWL 2 **DBMS technology:** RDF triple store (GraphDB) **Middleware:** N/A **UI/UX technology:** N/A	https://github.com/SODALITE-EU/semantic-models

3.2. The SODALITE Infrastructure as a Code Layer

The Infrastructure as a Code Layer is in charge of transforming an Abstract Deployment Model built using the modeling layer into an executable blueprint and a set of related artifacts (configuration scripts and execution container images). It also includes tools that identify and detect anti-patterns to be avoided as well as mechanisms to optimize the deployment of an application based on its characteristics. More specifically, this layer includes the subcomponents described in the following table.

Table 2. Components of the Infrastructure as a Code Layer and corresponding repositories

Component	**Main used technologies**	**GitHub repository**
Abstract Model Parser: It parses an abstract deployment model and generates the corresponding abstract syntax tree **License**: Apache2	**Programming language(s):** Java, Python **DBMS technology:** N/A **Middleware:** N/A **UI/UX technology:** REST API	https://github.com/SODALITE-EU/iac-blueprint-builder
IaC Blueprint Builder: Starting from the output provided by the Abstract Model Parser, this generates a TOSCA blueprint **License**: Apache2	**Programming language(s):** Java, Python **DBMS technology:** N/A **Middleware:** N/A **UI/UX technology:** REST API	https://github.com/SODALITE-EU/iac-blueprint-builder
Runtime Image Builder: It generates application component images ready to be executed **License**: Apache2	**Programming language(s):** Python, Ansible **DBMS technology:** SQLite **Middleware:** xOpera, Docker, Singularity	https://github.com/SODALITE-EU/image-builder

	UI/UX technology: REST API	
Concrete Image Builder: Concrete Image Builder builds the image adjusting it to the execution platform and handles specific implementation regarding configuration, deployment or monitoring **License**: Apache2	**Programming language(s):** Python, Ansible **DBMS technology:** N/A **Middleware:** xOpera, Docker, Singularity **UI/UX technology:** API	https://github.com/SODALITE-EU/image-builder
Application Optimiser: Tries to build a performance-wise improved version of an application for a given target platform based on the optimisation options selected **License**: Apache2	**Programming language(s):** Python **DBMS technology:** N/A **Middleware:** Dockerhost engine, Singularity engine **UI/UX technology:** REST API	https://github.com/SODALITE-EU/application-optimisation
IaC Verifier: Acts as a facade to the Topology Verifier and Provisioning Workflow Verifier, and coordinates the processes of verification of the application deployment topology and provisioning workflow **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** N/A **Middleware:** Web Server **UI/UX technology:** Swagger REST API	https://github.com/SODALITE-EU/verification
Verification Model Builder: This component builds the models required to verify the deployment model and its provisioning workflow **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** RDF triple store (GraphDB) **Middleware:** RDF triple store (GraphDB) **UI/UX technology:** N/A	https://github.com/SODALITE-EU/verification
Topology Verifier: This component verifies the constraints over the structures of the TOSCA blueprints and Ansible playbooks **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** N/A **Middleware:** N/A **UI/UX technology:** N/A	https://github.com/SODALITE-EU/verification
Provisioning Workflow Verifier: It verifies the constraints over the deployment (provisioning) workflow of the application using one of the widely used techniques for verifying workflows such as Petri Net **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** N/A **Middleware:** CPN Tools **UI/UX technology:** N/A	https://github.com/SODALITE-EU/verification

Bug Predictor and Fixer: It detects smells in TOSCA and Ansible playbooks and suggests corrections or fixes for each smell **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** N/A **Middleware:** Web Server **UI/UX technology:** Swagger REST API	https://github.com/SODALITE-EU/defect-prediction
Predictive Model Builder: This component uses a rule-based model for detecting implementation and security smells in Ansible playbooks and TOSCA blueprints **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** RDF triple store (GraphDB) **Middleware:** RDF triple store (GraphDB) **UI/UX technology:** N/A	https://github.com/SODALITE-EU/defect-prediction
IaC Quality Assessor: it includes the tool to calculate the software quality metrics for TOSCA and Ansible artifacts **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** N/A **Middleware:** N/A **UI/UX technology:** N/A	https://github.com/SODALITE-EU/iac-quality-framework
Image Registry: It stores the images after their generation by the Runtime and Concrete Image Builder **License**: Apache2	**Programming language(s):** Python, Ansible **DBMS technology:** N/A **Middleware:** Dockerhost engine **UI/UX technology:** N/A	https://github.com/SODALITE-EU/iac-management
Examples of IaC code and images: This is not a software component, but rather a set of IaC code examples that are developed to provide examples to the users of the SODALITE platform **License**: Apache2	**Programming language(s):** TOSCA, Ansible **DBMS technology:** N/A **Middleware:** Dockerhost engine **UI/UX technology:** N/A	https://github.com/SODALITE-EU/iac-management

3.3. The SODALITE Runtime Layer

The Runtime Layer is in charge of orchestrating, monitoring and reconfiguring the execution of a complex application even when it exploits multiple execution environments (Cloud, HPC, GPUs). The following table describes each sub-component of this layer and links to the corresponding GitHub repository. Most of the sub-components highlighted here extend and/or integrate pre-existing open source components. These are included as submodules in the corresponding repositories.

Table 3. Components of the Runtime Layer and corresponding repositories

Component	**Main used technologies**	**GitHub repository**
Orchestrator -> xOpera: This is a pre-existing lightweight	**Programming language(s):**	https://github.com/SODALITE

TOSCA compliant orchestrator that executes infrastructure provisioning and deployment of applications and blueprints produced in SODALITE. **License**: Apache2	TOSCA, Ansible, Python **DBMS technology:** N/A **Middleware:** OpenStack **UI/UX technology:** REST API	-EU/orchestrator includes the following repository as submodule https://github.com/xlab-si/xopera-opera
Orchestrator -> ALDE: This component includes the drivers that enable the usage of xOpera within SODALITE. **License**: Apache2	**Programming language(s):** Python **DBMS technology:** SQLite **Middleware:** Flask **UI/UX technology:** REST API	https://github.com/SODALITE-EU/orchestrator
xOpera REST API - includes xOpera REST API interface with persistence, session management, status of deployment, history of deployment, documented with swagger	**Programming language(s): Python** **DBMS technology: Postgres** **Middleware: Flask** **UI/UX technology: REST API**	https://github.com/SODALITE-EU/xopera-rest-api
Monitoring -> IPMI Exporter: This includes custom exporter files that enable to get the power consumption of physical nodes **License**: Apache2	**Programming language(s):** Go **DBMS technology:** ElasticSearch, OrientDB **Middleware:** Prometheus, Grafana, Skydive **UI/UX technology:** REST API, WebUI	https://github.com/SODALITE-EU/ipmi-exporter
Monitoring -> Skydive Exporter: The Skydive Flow Exporter is a pre-existing tool that provides a framework for building pipelines which extract flows from the Skydive Analyzer (via it WebSocket API), process them and send the results upstream. **License**: Apache2	**Programming language(s):** Go **DBMS technology:** ElasticSearch, OrientDB **Middleware:** Prometheus, Grafana, Skydive **UI/UX technology:** REST API, WebUI	https://github.com/SODALITE-EU/monitoring-system includes the following repository as submodule: https://github.com/skydive-project/skydive-flow-exporter
Monitoring configuration files: This includes configuration files used for the current Prometheus deployment **License**: Apache2	**Programming language(s):** TOSCA, Ansible, Prometheus configuration files (YAML) **Middleware:** Prometheus, Grafana, Skydive	https://github.com/SODALITE-EU/monitoring-system
LRE Exporter: It is a Prometheus exporter that provides monitoring metrics at	**Programming language(s):** Go **Middleware:** Prometheus **UI/UX technology:** REST API	https://github.com/SODALITE-EU/monitoring-lre-agent

the level of the Light-weight Runtime Environment (LRE). **License**: Apache2		
HPC Exporter: It is a Prometheus that provides monitoring metrics of the execution of applications on an HPC environment. **License**: Apache2	**Programming language(s):** Go **Middleware:** Prometheus **UI/UX technology:** REST API	https://github.com/SODALITE-EU/hpc-exporter
Deployment Refactorer: Includes rule-based and machine-learning based approaches to refactoring the deployment model of an application at runtime **License**: Apache2	**Programming language(s):** Java**,** MySQL, Redis, Python **Middleware:** Web Server, Rule Engine, Redis, Varnish **UI/UX technology:** Swagger REST API	https://github.com/SODALITE-EU/refactoring-ml
Node Manager: Includes control-theory based approaches to managing the resources in the nodes in a deployment model **License**: Apache2	**Programming language(s):** Python **DBMS technology:** N/A **Middleware:** TensorFlow, Spark, Kubernetes **UI/UX technology:** N/A	https://github.com/SODALITE-EU/refactoring-ct
Refactoring Option Discoverer: Includes semantic-matching capabilities for locating the new deployment options and resources **License**: Apache2	**Programming language(s):** Java and Python **DBMS technology:** RDF triple store (GraphDB) **Middleware:** RDF triple store (GraphDB) **UI/UX technology:** N/A	https://github.com/SODALITE-EU/refactoring-option-discoverer

4. General organization of repositories

Each repository provides the source code associated with the corresponding component, any infrastructural code and configuration files needed to compile it, deploy it, and make it work, the test suites currently available and executed on the software, known open issues and bugs, and its public APIs, as openAPI [4] specification. Thus, we envision the organization of the SODALITE repositories as follows:

```
|- Repository X
|-- documentation
|---- public APIs (openAPI)
|-- source code
|---- unit tests
|-- infrastructural code (any script needed to compile, deploy, execute the code)
|-- integration tests (tests to check the integration of component X)
|-- open issues/bugs: link to an issue tracking system or to a document
```

The only exception to this rule is the semantic-model repository which does not include source code but only ontology definitions (text files). As such, it shows the following simplified structure:

```
|- semantic-models
|-- documentation
|-- ontology definitions
|-- open issues/bugs: link to an issue tracking system or to a document
```

The project-wide documentation is made available on a dedicated repository (https://github.com/SODALITE-EU/project-wide-documentation) that features the following structure:

```
Sodalite ---project-wide documentation
|- general rules and roles
|- docker images (through a link to Docker Hub)
|- integration tests
|- infrastructural code
|--any script needed to compile the whole SODALITE framework
|--any script needed to compile partial solutions (design and runtime frameworks)
|-- uml
```

Within this structure, we will include the open-source **LICENSE** [5] associated with each repository (at the time of writing, all components under development feature an Apache 2 license), the **README** that represents the instruction manual that welcomes new community members to the project, the **CONTRIBUTING** document that helps people *contribute* to the project (essentially this deliverable and defines what one can do), the **CODE_OF_CONDUCT** that sets ground rules for participants' behavior and helps facilitate a friendly environment (i.e., how to contribute). The project will also have additional documentation, such as tutorials or walkthroughs.

5. Licenses

All the components of the SODALITE framework have been currently released by using the Apache 2 license scheme. Additional license models might be used and integrated in the project in the next phases.

6. Roles and Responsibilities

For each repository, SODALITE identifies five main roles:

- A **project leader** is in charge of the final decisions and is supposed to mitigate and manage inconsistencies and different views.
- An **artefact leader** is responsible for making final decisions about features, releases, and any other activity related to the specific artefact. This role is usually played by a representative of the partners that contributed/developed the artefact but could also be played by two/three people, mainly from different partners, if the size and importance of the artefact call for a small committee.
- **Committers** are those who have contributed to the repository and are considered reliable and responsible enough to be allowed to commit directly to all or some parts of the project, rather than having to submit to an artefact leader for review. Contributions from committers are still subject to review by project leaders and may be reverted if there are concerns.
- **Contributors** are those who contribute with code, documentation and other enhancements. These contributions are usually subject to a review from an experienced committer and the artefact leader before they are included.
- **Users** give the project a purpose and help it evolve. These valuable members of the community can provide feedback about features, bug reports and more.

A strong, vibrant, and diverse community is important to the success of open source communities. All of the people in the roles listed above are key to the SODALITE community, in general, and to the different sub-communities, organized around the different artefacts in particular.

7. SODALITE development flow

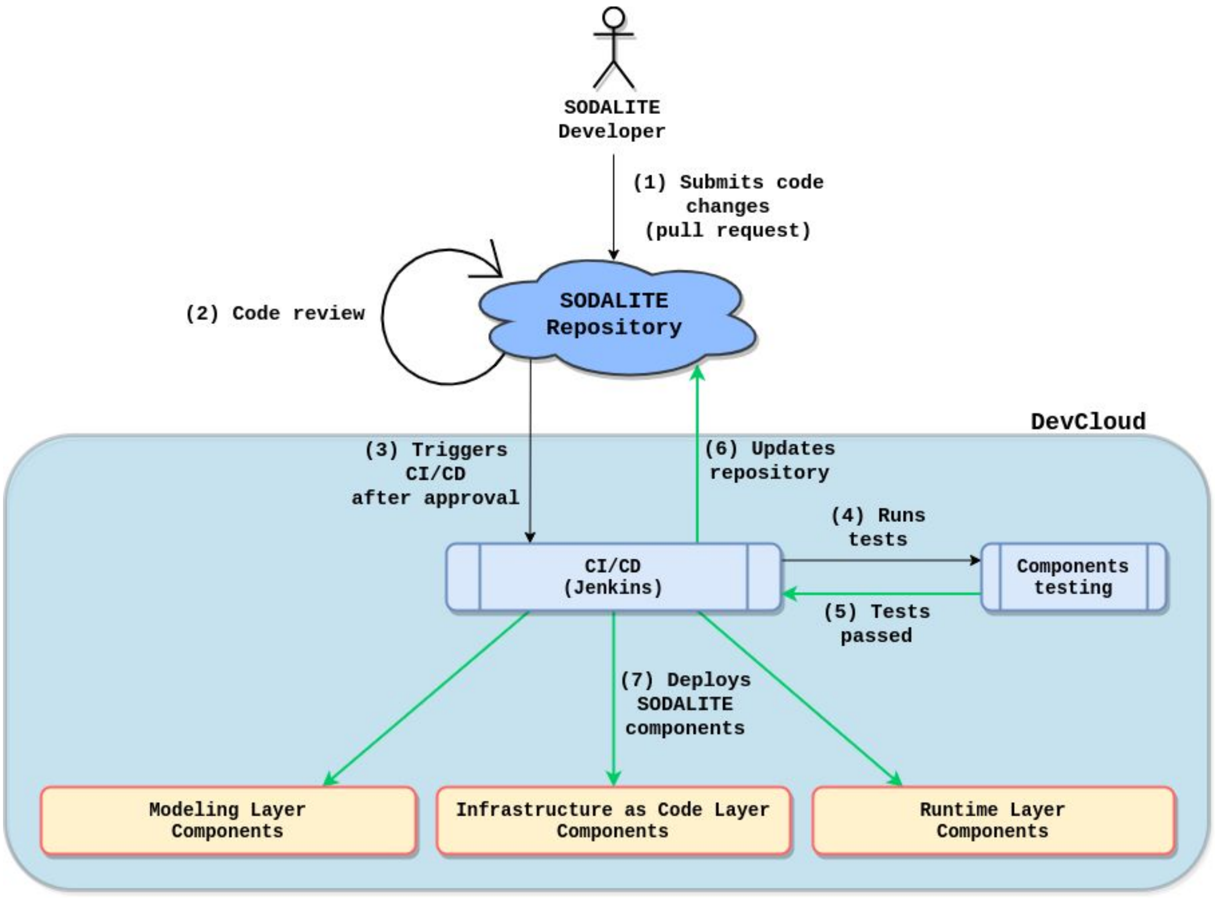

Figure 2. The development flow

SODALITE follows the CI/CD (Continuous Integration/Continuous Delivery) approach and its development flow is depicted in Figure 2. Firstly, a SODALITE developer, either a committer or a contributor (see the Roles section), submits a pull request after locally changing the source code of one of the SODALITE repositories. Upon this, a respective technical project leader reviews the changes and either approves or rejects the pull request, preliminarily discussing with the developer about the decision the leader made or in case something is unclear.

When the code changes are approved, the CI/CD pipeline is triggered by Jenkins, an automation server. It schedules any unit, integration and functional tests of SODALITE components. These tests validate the changes and prove that the updates did not break the project. In case of the test failures, the technical project leader and the developer are notified about the failures and certain actions are performed collectively to mitigate them. As soon as the tests are passed, the source code changes are pushed into the repository. The SODALITE components are then ready for the manual or automated deployment by artifact leaders (e.g., as a new production release of the SODALITE platform or a bug fix update) and are subsequently available to the users.

8. Guidelines for external contributors

The guide "How to Contribute to Open Source" [6] suggests that before doing anything, new contributors should always carry out a quick check to make sure their ideas have not been addressed already. Potential contributors must always skim through the project documentation (e.g., README files, open and closed issues). If they cannot find their ideas elsewhere, then they are ready to contribute.

8.1. Types of contributions

The many facets of a big project like SODALITE can envision different types of contributions:

- Those who like designing can restructure layouts to improve the project's usability, but they can also create a style guide to help the project have a consistent visual design.
- Those who like writing can help improve the project's documentation, curate examples to show how the different artefacts work, write tutorials, or even translate the documentation in another language.
- Those who like organizing can help keep things organized, identify duplicates, manage (and close) open issues, ask clarifying questions to move the discussion forward.
- Those who like coding can address open issues, propose a new feature, improve automated project setup, create and run new tests.
- Those who like working on the code written by others can review it, test and debug it, and create and run new tests.
- Those who like helping people can answer questions about the project and open issues and help moderate the discussion boards or conversation channels.
- Those who like extending the set of computational resources supported by SODALITE can define new resource models for SODALITE to widen its possible scope.
- Those who are interested in resource management and quality can define ways (patterns) for using resources.
- Those who like focusing on application operation will develop new deployment models and new ways of using the framework.

These are the possible types of contributions we envision for the different SODALITE repositories. These will be provided by the people who are already part of the consortium, but we envision that people outside of the consortium can become involved and contribute new and interesting artifacts to help improve the SODALITE framework.

8.2. How to contribute

As mentioned in the previous sections, there could be multiple ways of contributing (summarized in Figure 3):

- **Contributing some new resource models**: This is a very valuable contribution for us as it would extend the ecosystem of computational resources SODALITE is able to offer. Such a contribution will concern the semantic model repository (https://github.com/SODALITE-EU/semantic-models). In this case, we suggest contributors to issue a pull request on that repository. The pull request should include a clear description of what the new model describes and how it can be used by others.
- **Contributing a description of a new use case for the SODALITE platform**: This contribution is precious as it shows that SODALITE can be exploited by external users for modeling specific application examples. In this case a document describing the application along with the corresponding models could be produced. Such contribution will concern the project-wide documentation (https://github.com/SODALITE-EU/project-wide-documentation). In this case, we suggest contributors to issue a pull request on that repository. The pull request should include a

clear description of what the new application case is about, of the corresponding application architecture, and a description of the associated models.

- **Signaling a misbehaviour of the system**: SODALITE users may encounter unexpected faults and misbehavior while using the platform. In this case, we suggest users to open an issue in the repository associated with the component they think has shown the bug. In case it is not possible to identify a specific repository, then we ask users to open the issue on https://github.com/SODALITE-EU/project-wide-documentation. Each opened issue will be assigned to a SODALITE contributor that will follow up on it.
- **Contributing a bug fix**: external contributors as well as internal ones can propose to assign to themselves an issue request - this is done replying to the issue request and waiting for feedback by one of the committers - and then produce a corresponding pull request in the relevant repository.
- **Contributing a proposal for a new feature**: external contributors willing to propose the development of a new feature can open an issue request of type “enhancement”. This will be discussed with the rest of the team and, if approved, developed.
- **Contributing a solution implementing a new feature**: As soon as a proposal for a new feature is approved (see previous point), an external contributor, as well as an internal one, can offer to take care of it. In this case, the contributor proposes to assign to himself/herself the development of the specific feature and, when ready, will submit a pull request that will be reviewed by the SODALITE team. Another possibility is that, without going through the step of proposing a new feature, the contributor proposes the solution directly through a pull request. This second option, even if possible, is not encouraged as it may lead to work duplication and misalignment. To mitigate this problem, it is advisable to open the pull request as soon as possible and to mark it as “in progress” in order to let the others know, to allow them to watch and monitor any progress, and to provide any feedback. Dedicated commits must then correspond to the different milestones.

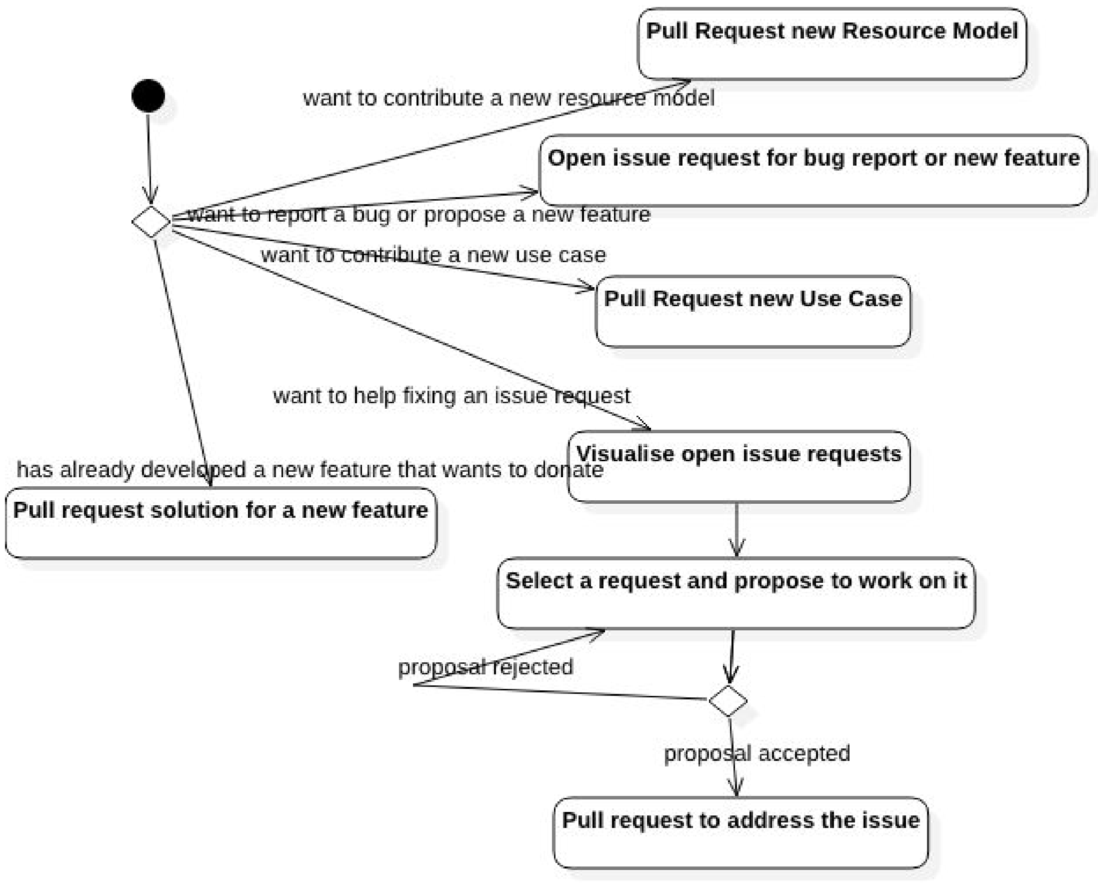
Pull Request new Resource Model
want to contribute a new resource model
Open issue request for bug report or new feature
want to report a bug or propose a new feature
want to contribute a new use case
Pull Request new Use Case
want to help fixing an issue request
has already developed a new feature that wants to donate
Visualise open issue requests
Pull request solution for a new feature
Select a request and propose to work on it
proposal rejected
proposal accepted
Pull request to address the issue

Figure 3. Possible types of contributions to the SODALITE project

The contributor must follow a simple process to submit a pull request. The contributor must fork the repository and clone it locally. The local and original "upstream" repositories must be connected by adding the latter as remote to the former. The contributor must then pull changes from the remote repository to keep the local version aligned and avoid conflicts. A dedicated branch must then be created for carrying out any relevant change, modification, or addition. Every possible contribution must always respect the styles, rules, and conventions adopted by the project. The contributor must consider what the project, and the community, is used to, and not his/her common habits. The goal is to ease any possible merge, and help the others understand. The contributor must also add references to any appropriate issue, document, or artefact in the pull request to help the others scope it properly. Screenshots of the before and after, if appropriate, could also be added to further clarify the scope of the change.
Finally, every change must always be tested properly: regression testing should be carried out if possible, and additional tests be developed to assess the quality and impact of the proposed change. Nothing should break the existing project. Figure 4 summarizes the steps to be followed when working at a pull request. After submission, the workflow in Figure 2 is entered.

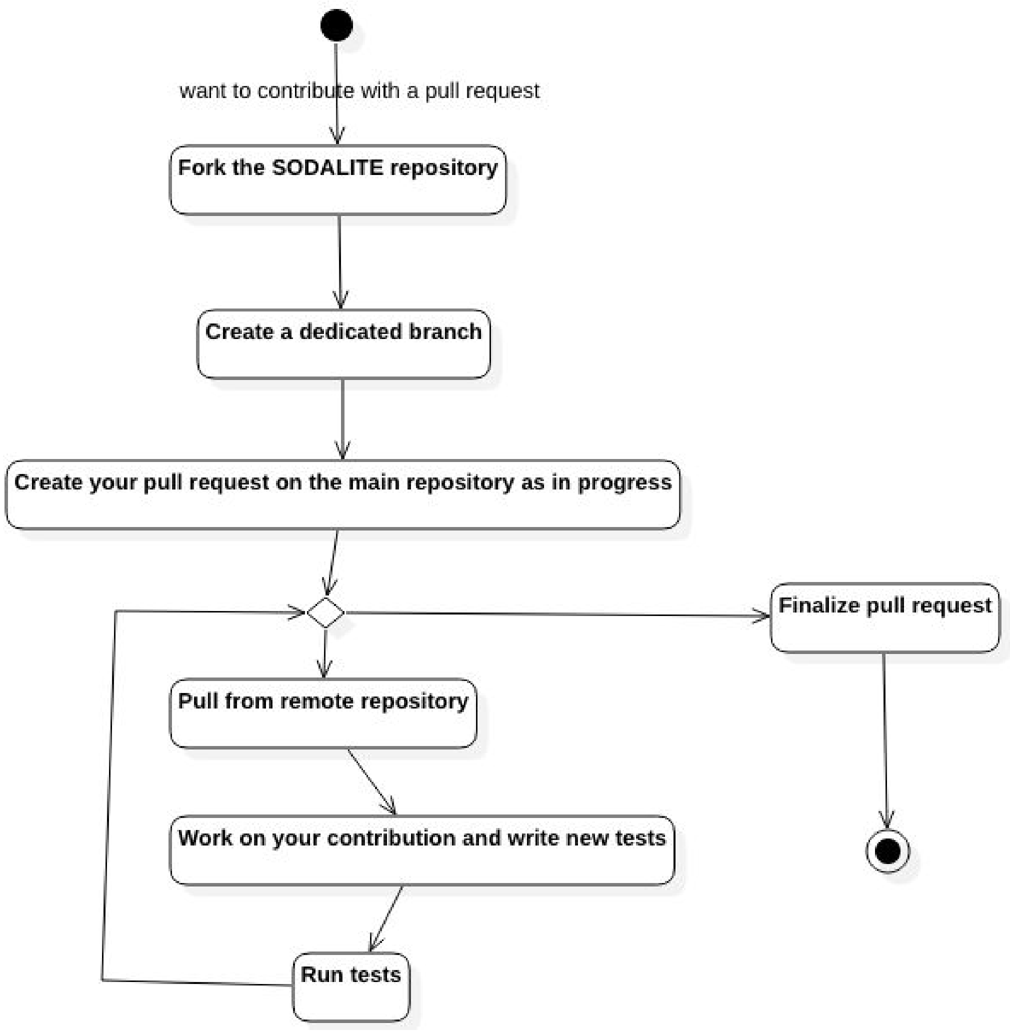
want to contribute with a pull request
Fork the SODALITE repository
Create a dedicated branch
Create your pull request on the main repository as in progress
Finalize pull request
Pull from remote repository
Work on your contribution and write new tests
Run tests

Figure 4. Pull request workflow

In general, the SODALITE team would like to enter in touch informally with potential contributors as soon as possible and discuss with them any problem or new idea. To this purpose, the team will open a slack channel and will provide on GitHub instructions to join it.

9. Guidelines for software and release management

The project adopts the best practice guidelines to govern the management and quality of released software and versions [7, 8, 9]. These guidelines are to be followed by all SODALITE contributors, either internal to the project or external.

According to what the Apache Software Foundation (ASF) says, a release in SODALITE is *"anything that is published beyond the group that owns it, that is, any publication outside the development community"*[1]. In the case of SODALITE, the unit of release is the service [8], that is, a component offering a proper REST API. The only exception to this principle is the IDE that acts as the frontend client of the SODALITE platform. Each service must be documented at least through a usage example that is used as a basis for integration testing.

The development process adopted for the different artifacts is based on the following assumptions:

- Each component resides in a GitHub repository;
- Each repository must have unit tests defined;
- Each repository has a Jenkins process defined to perform integration;
- Anything in the master branch is always deployable. This is guaranteed by the automated processes defined by Jenkins;
- Each newly released component must be accompanied by an example of usage defined in terms of corresponding REST calls. This facilitates integration of components.

The procedure to contribute new code to a repository works as follows:

- A pull request is issued to integrate the code into the master branch. If the user performing the pull request does not have write access to the repository, then the pull request will be manually evaluated by the repository owner who may delegate other partners in the request review. If the user performing the pull request has write access to the repository, then the request activates the Jenkins job automatically, builds the code, runs the defined unit tests;
- If the unit tests fail, then the code is not integrated in the master; - For this to happen, the Jenkins status check (**continuous-integration/jenkins/pr-merge)** needs to be defined as a *required status check*;
- If unit tests are successful, the code is integrated into the master (this is currently manual, but can be automated by Jenkins, if desired) and a new ready to use image is created and uploaded on DockerHub.

In addition, we also follow the following good practices, as suggested by GitHub flow:

- New ideas and evolutions are managed through branches, and their names should be descriptive;
- Each commit has an associated commit message, a description explaining why a particular change was made (each commit is considered a separate unit of change);
- Pull Requests contain an explanation to inform repository owners and other partners about the changes one would like to perform;
- Each pull request must always be associated with an issue in the management board to relate a change to an actual need;
- Quality metrics will be periodically collected and their trends will be analyzed over time to ensure the code of each component will gradually improve its quality through the project. Based on the outcome of the first measurements, we will decide whether to add automated checks in the CI/CD pipeline associating them to some quality gates. As for the tool to be used for quality metrics collection, the consortium is evaluating different solutions that both provide sufficient guarantees and cover the polyglot context behind SODALITE. At the moment, SonarQube appears to be the most likely choice given that it

[1] https://www.apache.org/legal/release-policy.html

supports multiple languages, offers a large degree of flexibility and it is well known by the SODALITE CI/CD master. Given the need for a dedicated VM to SonarQube scanners, the feasibility of deploying it on our testbed is under investigation. Other possible options are CodeFactor, Codacy or Code Climate. As for the specific metrics to collect, we will keep under control at least the following ones: cyclomatic complexity; duplicated blocks, files, lines; typical code smells; vulnerabilities; coverage of tests. As mentioned above, we will make sure that at every new release of each component the metrics show an improvement compared to the previous value.

A SODALITE platform release is composed of the IDE and the set of microservices made available as images on DockerHub. The deployment and execution of the whole platform is planned to be automated through TOSCA and Ansible blueprints created and maintained using the same SODALITE tools. These are available here https://github.com/SODALITE-EU/iac-platform-stack. A user willing to deploy and execute the platform must go through the following steps:

- Have the xOpera orchestrator installed either locally or on a VM in the preferred cloud provider
- Install the IDE locally to his/her machine. The installation can either start from downloading the source code or can rely on the corresponding Docker image available on DockerHub
- Edit the blueprints to include the specific information concerning the resources that will be used to run the platform. Specific instructions for this will be released together with the blueprints (now under development) and maintained through the development of the project.
- Run the blueprints through xOpera.

During the development of the project we plan to create SODALITE platform releases according to the following timeline:

- **M12:** Laboratory prototype that is “up-and running”. This initial version of the SODALITE platform is released in terms of source code and demonstrators, outside the CI/CD and automated deployment pipeline we have been defining in parallel.
- **M18**: First consolidated prototype, Use-Cases can all be executed on it.
- **M24:** First advanced features, more integrated prototype running. Use-Cases are clearly improved. Second public release of the complete stack.
- **M30:** Prototype validated by Use-Cases. Planning for the last features and their integration complete. Third public release of the complete stack.
- **M36:** Evaluated and integrated prototype. Use Cases used to validate the Use-Cases. Final public release of the complete stack.

All releases, except the first one, will be accompanied by the corresponding automatically generated Docker images and the TOSCA/Ansible blueprints that automate the deployment and execution of the whole infrastructure.

10. Conclusions

This document discusses the guidelines for creating open-source communities behind the different artefacts developed by SODALITE. It also identifies rules, roles, and hints to let external people contribute to the project and help ameliorate it.

This document will then serve as reference for the SODALITE communities and will be updated properly while the project evolves, and new needs emerge.

References

1. Eric S. Raymond, The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly Media, 258 pages
2. Stormy Peters and Nithya Ruff, Participating in open source communities (https://todogroup.org/guides/participating/ and https://www.linuxfoundation.org/resources/open-source-guides/participating-open-source-communities/)
3. SODALITE Consortium, Requirements, KPIs, evaluation plan and architecture - First version, Technical deliverable 2.1, 2019.
4. Choose an open source license (https://choosealicense.com)
5. The OpenAPI Specification (https://www.openapis.org)
6. How to Contribute to Open Source (https://opensource.guide/how-to-contribute/)
7. Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy Editors, Site Reliability Engineering - how Google runs production systems, O'Really 2016.
8. A. Fox, D. Patterson, Engineering Software as a Service, an Agile Approach Using Cloud Computing, Strawberry Canyon LLC; 2nd ed, 2013.
9. Pablo Orviz Fernández, Mário David, Doina Cristina Duma, Elisabetta Ronchieri, Jorge Gomes, and Davide Salomoni, Software Quality Assurance in INDIGO-DataCloud Project: a Converging Evolution of Software Engineering Practices to Support European Research e-Infrastructures, Journal of Grid Computing volume 18, pages 81–98(2020).