

Deliverable D1.4

The SmartCLIDE Concept

WP 1

Project Acronym & Number:	SmartCLIDE – GA 871177
Project Title:	Smart Cloud Integrated Development Environment supporting the full-stack implementation, composition and deployment of data-centered services and applications in the cloud
Status:	Final
Dissemination Level:	Public
Authors:	ATB
Contributors:	All Partners
Document Identifier:	D1.4 The SmartCLIDE Concept v1.0
Date:	31.10.2020
Revision:	1.0
Project website address:	https://smartclide.eu

Every effort has been made to ensure that all statements and information contained herein are accurate, however the SmartCLIDE Project Partners accept no liability for any error or omission in the same.

© 2020 Copyright in this document remains vested in the SmartCLIDE Project Partners.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871177

Partner Contacts

Institut für angewandte Systemtechnik Bremen GmbH (ATB), Germany

Intrasoft International SA (INTRA), Luxembourg

Fundación Instituto Internacional de Investigación en Inteligencia Artificial y Ciencias de la Computacion (AIR), Spain

University of Macedonia (UoM), Greece

Ethniko Kentro Erevnas Kai Technologikis Anaptyxis (CERTH), Greece

X/OPEN Company Limited (TOG), United Kingdom

Eclipse Foundation Europe GMBH (ECLIPSE), Germany

Wellness Telecom SL (WT), Spain

Unparallel Innovation LDA (UNP), Portugal

CONTACT Software GmbH (CONTACT), Germany

Kairos Digital, Analytics and Big

Data Solutions SL (KAIROS DS), Spain

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Control

Version	Notes	Date
0.1	Creation of the document	22.07.2020
0.2	First inputs from technology developers	04.09.2020
0.4	Overall Concept	02.10.2020
0.5	TRL4 Lab Validations	09.10.2020
0.6	SmartCLIDE features	16.10.2020
0.8	Internal review from all partners	27.10.2020
0.9	Internal review from dedicated reviewers	30.10.2020
1.0	Final reviews and QA version for EC submission	03.11.2020

Abbreviations

AB	Advisory Board	GDPR	General Data Protection Regulation
AI	Artificial Intelligence	GLSPs	Graphical Language Server Platforms
API	Application Programming Interface	HTML	HyperText Markup Language
App	Software Application	IaaS	Infrastructure-as-a-Service
APM	Adaptive Project Management	ICT	Information and Communication Technology
AWS	Amazon Web Services	IDE	Integrated Development Environment
B2B	Business-to-Business	i.e.	id est = that is to say
BC	Business Case	IP	Intellectual Property
BPMN	Business Process Model and Notation	IT	Information Technology
D	Deliverable	IPR	Intellectual Property Rights
DL	Deep Learning	KPI	Key Performance Indicator
DLE	Deep Learning Engine	LSPs	Language Server Protocols
DNS	Domain Name System	M	Month
DoA	Description of Action	ML	Machine Learning
DSLs	Domain-Specific Languages	NLP	Natural Language Processing
EA	Ethical Adviser	OWL	Web Ontology Language
PB	Plenary Board	PaaS	Platform-as-a-Service
EC	European Commission	PB	Plenary Board
e.g.	exempli gratia = for example	PC	Project Coordinator
etc.	et cetera	PO	Product Owner
EU	European Union	PQA	Project Quality Assurance
FP7	Framework Programme 7	QA	Quality Assurance
GA	Grant Agreement		

RDF	Resource Description Framework	UML	Unified Modelling Language
REST	REpresentational State Transfer	URI	Universal Resource Identifier
RMV	Runtime Monitoring & Verification	URL	Universal Resource Locator
RTD	Research and Technological Development	UX	User Experience
SaaS	Software-as-a-Service	VoIP	Voice over IP
SME	Small and Medium Sized Enterprise	WP	Work Package
SC	Steering Committee	WPL	Work Package Leader
SOAP	Simple Object Access Protocol	WPMT	Work Package Management Team
STQA	Scientific and Technical Quality Assurance	w.r.t.	with respect to
T	Task	WSDL	Web Service Definition Language
UDDI	Universal Description, Discovery and Integration	XML	eXtensible Markup Language
		XMP	eXtensible Metadata Platform

Executive Summary

The current document presents deliverable D1.4 - SmartCLIDE Concept. The work described in this document is part of T1.4 Design of SmartCLIDE System Concept for WP1 – Specification of SmartCLIDE concept and pilot cases.

The objectives of this task can be summarised in the points below:

- Definition of the features and the functionalities required for the project,
- Identification of the basic functionality for the SW components that will be developed within this project,
- Elaboration of the project concept based on the collected requirements.

This document summarises the concept of the envisaged SmartCLIDE solution, based on the requirements of the industrial partners, as well as the state-of-the-art, including the basic approach for each of the main SW services and components.

The main result of these activities is the high-level concept for the SmartCLIDE solution, which will serve as the starting point for the detailed design documents.

Table of Contents

1	Introduction.....	12
1.1	Document Purpose	12
1.2	Approach Applied	12
1.3	Structure of this document	13
1.4	Relationship to other deliverables	13
1.5	Contributors.....	14
2	Overall SmartCLIDE Concept.....	15
2.1	Purpose of SmartCLIDE	15
2.2	Targeted users and main use cases	17
2.2.1	Targeted Users:.....	17
2.2.2	Main Use Cases	19
3	SmartCLIDE Features	24
3.1	Smart assistance in the IDE.....	25
3.2	Main SmartCLIDE Workflow(s).....	26
4	SmartCLIDE Services and Components	28
4.1	Discovery of Services and Resources	28
4.1.1	TRL 4 Lab Validations (Minimum Viable Product)	30
4.2	Services Creation, Composition and Testing	35
4.2.1	TRL 4 Lab Validations (Minimum Viable Product)	37
4.3	Security.....	52
4.3.1	TRL 4 Lab Validations (Minimum Viable Product)	54
4.4	Runtime Monitoring and Verification.....	59
4.4.1	TRL 4 Lab Validations (Minimum Viable Product)	62
4.5	Run-time Simulation & Monitoring / Visualisation.....	65
4.5.1	TRL 4 Lab Validations (Minimum Viable Product)	66
4.6	Deep Learning Engine.....	69
4.6.1	TRL 4 Lab Validations (Minimum Viable Product)	70
4.7	Context Handling	76
4.7.1	Context Monitor.....	76
4.7.2	Context Extractor.....	79
4.7.3	Context Provider.....	84
4.7.4	TRL 4 Lab Validations (Minimum Viable Product)	85
4.8	User Interface / Workbench	87
4.8.1	TRL 4 Lab Validations (Minimum Viable Product)	90
4.9	Smart Assistant.....	96
4.9.1	TRL 4 Lab Validations (Minimum Viable Product)	97
4.10	Services Deployment.....	100

4.10.1	TRL 4 Lab Validations (Minimum Viable Product).....	100
5	Conclusions.....	106
6	References.....	107

List of Figures

Figure 1: Approach applied for elaboration of SmartCLIDE Concept	12
Figure 2: SmartCLIDE Targeted Users.....	17
Figure 3: Normal Flow example	23
Figure 4: The SmartCLIDE Feature Map	24
Figure 5: Service Discovery Approach	30
Figure 6: Service Discovery - Data Sources	32
Figure 7: Service Creation.....	33
Figure 8: Service Discovery - Data Sources	34
Figure 9: Overview of the “ <i>Service Creation Composition and Testing</i> ” Component.....	37
Figure 10: Main workspace for our Process Decomposition - Workflow Composition tool	38
Figure 11: Elements	39
Figure 12: Connectors	40
Figure 13: Gateways.....	40
Figure 14: Events	41
Figure 15: Project Explorer	42
Figure 16: Drag and drop capabilities for the task decomposition	43
Figure 17: Connect nodes with connector.....	43
Figure 18: Workflow Options Menu.....	44
Figure 19: Node Options Menu – Properties tab.....	45
Figure 20: Node Options Menu – Functionality Tab	46
Figure 21: Service Discovery.....	47
Figure 22: Node options.....	47
Figure 23: Complete Design Workflow	48
Figure 24: Service Discovery failed.....	49
Figure 25: Service Creation.....	50
Figure 26: Workflow Completed	50
Figure 27: Test and Integration	51
Figure 28: High-level overview of the Security component	54
Figure 29: The Workflow Composition tool with the Security menu under the Constraints tab .	55
Figure 30: The results of the Security-related Static Analysis component	56
Figure 31: The workflow updated with the results of Vulnerability Assessment.....	57
Figure 32: Application and Monitor Creation and Deployment	61
Figure 33: Use case scenario for monitoring a certain container	65
Figure 34: Run-time Simulation & Monitoring – Overview.....	66
Figure 35: Run-time Simulation & Monitoring – Terminal.....	67
Figure 36: Run-time Simulation & Monitoring - Settings	68
Figure 37: Deep Learning Engine - Overview	70
Figure 38: Deep Learning Engine - Import Data	71
Figure 39: Deep Learning Engine - Import Data Details	71

Figure 40: Deep Learning Engine - Model Creation..... 72

Figure 41: Deep Learning Engine - Model Creation - Target Features 72

Figure 42: Deep Learning Engine - Model Creation - Algorithm and Training..... 72

Figure 43: Deep Learning Engine - Data Visualization - Model selection..... 73

Figure 44: Deep Learning Engine - Data Visualization - Model metrics 73

Figure 45: Deep Learning Engine - Data Visualization - Model visualization..... 74

Figure 46: Conceptual Context Handling Architecture..... 76

Figure 47: Conceptual Context Monitor Architecture 77

Figure 48: Conceptual Context Extractor architecture..... 80

Figure 49: Context Model for Laboratory Validation..... 85

Figure 50: Overview of workspace organization in Eclipse Che..... 89

Figure 51: Diagram of SmartCLIDE IDE and SmartCLIDE components communication..... 90

Figure 52: Login page for SmartCLIDE ecosystem..... 90

Figure 53: Welcome page when a user logs in..... 91

Figure 54: Services page 92

Figure 55: Example of SmartCLIDE plugin to support the development lifecycle..... 93

Figure 56: Deployments page 94

Figure 57: Smart Assistant - Workflow Composition..... 98

Figure 58: Process to deploy new Docker containers 100

Figure 59: Service Deployment – Overview..... 101

Figure 60: Service Deployment - Stacks..... 102

Figure 61: Service Deployment - Containers..... 103

Figure 62: Service Deployment - Images..... 104

Figure 63: Service Deployment - New Deployment..... 105

List of Tables

Table 1: SmartCLIDE Purpose	15
Table 1: Monitoring of systems/sensors.....	77
Table 2: Parsing of monitoring data.....	78
Table 3: Analysing of monitoring data	79
Table 4: Context Identification	81
Table 5: Ontological Context Reasoning	82
Table 6: Rule-Based Context Reasoning	83
Table 7: Statistical Context Reasoning	84

1 Introduction

1.1 Document Purpose

The current document presents deliverable D1.4 - SmartCLIDE Concept. The work described in this document is part of T1.4 Design of SmartCLIDE System Concept for WP1 – Specification of SmartCLIDE concept and pilot cases.

The objectives of this task can be summarised in the points below:

- Elaboration of the project concept based on the collected requirements.
- Definition of the features and the functionalities required for the project,
- Identification of the basic functionality for the SW components that will be developed within this project,

This document summarises the concept of the envisaged SmartCLIDE solution, based on the requirements of the industrial partners, as well as the state-of-the-art, including the basic approach for each of the main SW services and components.

The main result of these activities is the high-level concept for the SmartCLIDE solution, which will serve as the starting point for the detailed design documents.

1.2 Approach Applied

The SmartCLIDE concept presented here is the result of a process already started in the previous deliverable *D1.2 Requirements Analysis*, and illustrated in Figure 1 below.

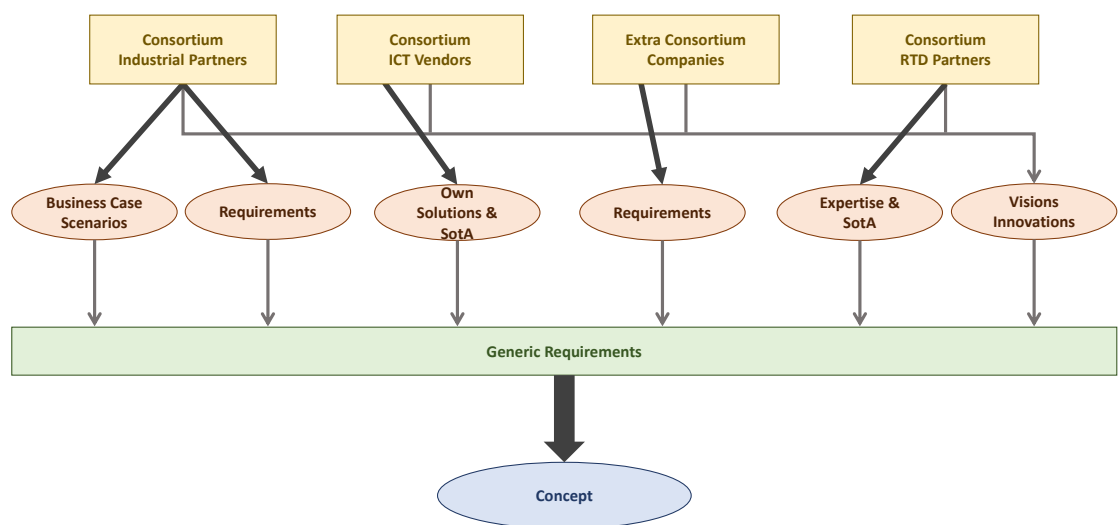


Figure 1: Approach applied for elaboration of SmartCLIDE Concept

The steps of that process were:

- Detailed analysis of the pilot cases by the three industrial partners
- Analysis of the survey results that were gathered from industrial developers from extra consortium companies.
- Creation of the textual descriptions of the pilot cases and extraction of needs and requirements.
- Collection of the information/insight into the market available solutions of corresponding application.
- On top of that, the RTD performers have created an in-depth analysis of the state-of-the-art R&D activities in the relevant areas, that was used and enriched by the expertise (of RTD performers), for creation of a generic set of requirements and generic application scenarios.
- All participants in the above described activities (see Figure 1) have also provided technical visions and innovation ideas to complete the generic requirements. The action was done to introduce the long-term visions for the future improvements of the solutions.

1.3 Structure of this document

The structure of the document is the following:

- Section 1, Introduction, includes a concise overview of the overall content of the document.
- Section 2, Overall SmartCLIDE Concept, describes the planned key purpose of the SmartCLIDE solution, and the targeted users of the SmartCLIDE solution as well as the main use cases for it.
- Section 3, SmartCLIDE Features, provides an overview of the main top-level features of the envisaged SmartCLIDE solution.
- Section 4, SmartCLIDE Services and Components provides a detailed description of the different SmartCLIDE ICT technologies covering the challenges that will be addressed, and the approach that will be utilised for development.
- Section 5, Conclusions, provides concluding remarks.

1.4 Relationship to other deliverables

This deliverable provides a description of the concept of the envisaged SmartCLIDE solution and provides the overall conceptual description of the software technologies based on the results of the task *T1.1 Analysis of SOTA and Market Requirements*, *T1.2 Specification of Requirements*, and *T1.3 Specification of pilots and Use Case Scenarios*. This deliverable complements the deliverable *D1.3 – Use Case Scenarios*

where the use cases are defined and is furthermore tightly related to *D1.5 The SmartCLIDE Architecture* where the SmartCLIDE architecture is defined.

1.5 Contributors

All project partners have substantially contributed to this deliverable. In particular, the RTD partners and technology providers have described their services and components. KAIROS has led the creation of the overall SmartCLIDE concept together with ATB. The creation of the SmartCLIDE features map was led by ATB supported by KAIROS, AIR, CERTH and UNP. ATB has acted as overall editor in preparing each version of the document using a collaborative and iterative process of increasing levels of refinement.

2 Overall SmartCLIDE Concept

2.1 Purpose of SmartCLIDE

In a nutshell, the final goal of SmartCLIDE is to **boost the adoption of Cloud and Big Data solutions**. Cloud computing can be considered as the main enabler of digital transformation, since it allows organisations to disengage their growth from the need to acquire more powerful infrastructures. However, the creation and composition of new services in the cloud have increased in complexity, thus slowing down the progress towards the digital transformation process of business and public administrations.

When companies face the creation or composition of new services in their clouds, they can **(a) develop services from scratch; (b) create new services by composition or (c) employ some pricing model**. To each of these scenarios, SmartCLIDE will set different aims in the pursuit of overcoming their limitations and achieving SmartCLIDE’s final goal.

Table 1: SmartCLIDE Purpose

Main Objective: Boost the adoption of Cloud and Big Data solutions		
Services Creation	Limitations	Aims of SmartCLIDE
A) From Scratch	Complex & Expensive & Time Consuming: - Large number of technologies in the whole stack. - Low supply of staff in the market - Uneven maturity level of the development technologies hinders debugging tasks and QA activities (e.g. specification and automatic execution of unit tests).	Faster and more effective development of cloud and big data services Gaining deeper insights on how cloud and code works (from novel to expert)
B) Composition of Services	Data and Services management from hybrid clouds and multiple sources is complex. Marketplaces coupled to IaaS and PaaS providers. Lack of a uniform classification or documentation. QoS or Security can be compromised by such services and their integration.	More secure and easy way to reuse quality code and automated cataloguing services Gaining trust and facilitating the reuse of services

Main Objective: Boost the adoption of Cloud and Big Data solutions		
	Discovery and validation of valuable and secure services mostly manual and demonstrated by trial and error.	
C) Pricing models	<p>Very difficult prediction and control of costs</p> <p>Models depending on different variables and their combination: time of usage, resources (memory, storage, capacity), predictions obtained, volume of transferred data...</p>	<p>Code learning tool</p> <p>Gaining deeper understanding about the cost of Big Data and Cloud</p>

SmartCLIDE will overcome these limitations by proposing a new **smart** and **cloud-native** IDE. Among the more relevant features of this IDE will be:

- SmartCLIDE will be a **C**loud-based IDE and will boost the adoption of Cloud solutions. The cloud nature of the environment will enable collaboration between different stakeholders, and the support for cloud solutions will pave the way for the digital transformation.
- SmartCLIDE will make use of a Deep **L**earning Engine to automatically categorize the available resources before presenting them to the end-users. The cloud workbench will provide the end-user with multiple high level abstractions at all stages (requirements, design, development, testing, deployment and run-time). SmartCLIDE will provide several categories of abstractions: at development stage, SmartCLIDE will provide abstractions on data transformations or processing; at testing stage, mechanisms to visualize flow and status or artefacts to automatically test the expected behaviour; at deployment stage, abstractions of physical and virtual resources; or at run-time, mechanisms to monitor the performance and operation of the services
- SmartCLIDE is a new smart cloud-native **I**ntegrated Development Environment, based on the *coding-by-demonstration* principle. It will support creators of cloud services in the collaborative discovery, creation, composition, testing, and deployment of full-stack data-centred services and applications in the cloud.
- SmartCLIDE will allow the **D**iscovery of IaaS and SaaS services will facilitate the composition and deployment of new services for staff with no previous experience in programming or the administration of systems and infrastructure. Hiding the complexity of infrastructure, and adding intelligence, will allow selecting the most adequate infrastructure services at each moment.
- SmartCLIDE is based on the *coding-by-demonstration/ programming-by-example* principle. It was conceived already in the early 1970s to teach new behaviours to computers by providing them with specific examples. The

disruptive approach of SmartCLIDE is that it will make use of this principle with the objective of generating the underlying software that make the computer behave in a specific way.

2.2 Targeted users and main use cases

2.2.1 Targeted Users:

The main target users are developers. The tool facilitates the developer's work through automations and pre-established commands that increase efficiency in tasks such as version deployment, security tests according to established acceptance criteria, or software development according to the highest quality standards.

The tool helps eliminate potential dependencies, leading to improved self-organization and increased end-to-end accountability of the entire development stack.

SmartCLIDE ultimately helps the developer to achieve quality software and reduce time, even for novice ones with little understanding of the underlying mechanisms of data-intensive applications. Therefore, the developer is the main user of the tool. Despite this, SmartCLIDE will add value to the entire team, including product owners and managers with some technical skills, the project and the end user who will use the product directly and indirectly.

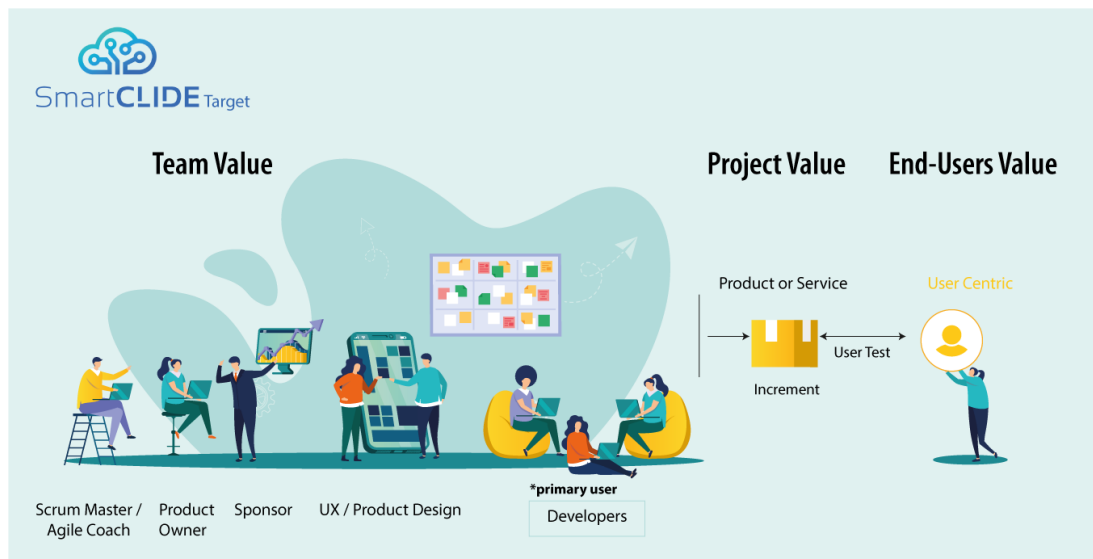


Figure 2: SmartCLIDE Targeted Users

Team Value:

Autonomous teams, with end-to-end responsibility, dealing with the full development stack, must also be able to select the best options for application deployment and make it available to end users, dealing with the associated complexity and multiplicity of available technologies.

UX. Product Design. Situated between Business and developers, they can work with feature specification, user stories, and acceptance testing. They facilitate the conversion of mock-up schematics to UML diagrams and the interpretation of User Story Maps, along with the rest of the team.

Product Owner / Scrum Master / Agile Coach. SmartCLIDE facilitates the overview of the project and the speed / performance of the team, with metrics according to this type of user. These users are also empowered in the specification, development, testing, deployment and operation of data-intensive applications in the cloud. They should be able to easily prototype features that can be enhanced later on by developers.

Sponsor. SmartCLIDE will support the provision of metrics and Product Roadmaps, by providing interfaces to integrate such solutions, to simplify the general monitoring of the product that is being built. Sponsors can receive feedback in a simple way, which facilitates the understanding of the tasks carried out by the team and improves communication with them.

Project Value:

SmartCLIDE adds value to the team and, consequently, adds value to the project. Also, an important factor that increases the capabilities of SmartCLIDE are integrations with other tools. This fact exponentially increases the use and scope in the different phases of the project from start to end (end to end), Business model design, Customer development and Agile product development.

Communication with the tool. Communication through **natural language** can bridge the possible gap for less technical users. Integration with tools, both spoken and written, improves the interaction of users with the platform. (Natural Language Processing Algorithms, DialogFlow, etc.)

Communication Developers / UX / Business. Integration with tools like Zeplin¹ helps in the development flow between designers and developers. Other team members can view the screens and the flow between them.

Test Users / Relationship with the client. Integration with User Test tools such as Invision², Lookback³, Hotjar⁴, facilitate learning and immediate incorporation into development cycles. (A / B test, heat maps)

¹ <https://zeplin.io/>

Integration with tools related to growth hacking, conversion funnels or Customer Relationship Management such as Active Campaign⁵, Typeform⁶, or CRM such as Salesforce⁷, SumaCRM⁸ or HubSpot⁹, helps to measure the interest and relationship with the customer or end user about the project or product. We can measure the interest of the ideas or the product.

Tasks / Team speed / efficiency. Jira¹⁰ or Trello¹¹ integration helps to gain greater control of the management of the entire team.

Own integrators. Integration with other integration tools, such as Zapier¹², can help the team automate tasks and add value to the projects generated with SmartCLIDE.

Metrics. Metrics related to team productivity, retrospectives and in general metrics related to digital transformation.

End User Value:

With the above, the end user benefits directly and indirectly from the tool. The team has greater control over what it produces and makes available to the user. For this reason, the perceived value of it increases. Likewise, given the need to make changes to the product, these can be made in a more agile and effective way within the product development cycles.

Ideally, SmartCLIDE could collect user feedback and automatically incorporate it into the build process.

2.2.2 Main Use Cases

The development of a product or service can be classified by its level of innovation. All developments arise from solving a problem detected in users. The way of approaching the projects is very similar. But the big difference is the uncertainty resulting from the innovation levels that we face.

Core. Generally, it is about improving an existing product within a company and it has a direct impact on its operation. Some examples could be improving the look and

² <https://www.invisionapp.com/>

³ <https://lookback.io/>

⁴ <https://www.hotjar.com/>

⁵ <https://www.activecampaign.com/>

⁶ <https://www.typeform.com/>

⁷ <https://www.salesforce.com/>

⁸ <https://www.sumacrm.com/>

⁹ <https://www.hubspot.com/>

¹⁰ <https://www.atlassian.com/software/jira>

¹¹ <https://trello.com/>

¹² <https://zapier.com/>

feel of a corporate website, improving management software to facilitate its use, or connecting distributed databases to improve customer service.

Adjacent. It can happen when it is detected that the target user of the product uses a specific functionality intensively and not the other parts. The company can divide the product as a spin off and turn that function into a new product or service, under a new brand / branding. For example, a corporate website of a company that has a test to prepare budgets for a sector, such as reforms, and it is detected that it provides the user with great value. This can be packaged as another product and sold with a B2B business model.

New. It is a totally new one. When these solutions have a great impact on one or several sectors at the same time, they can have a great disruption, such as Uber or Airbnb, which also have a series of management requirements and references to exponential organizations that are not relevant at this time.

We must consider how the SmartCLIDE tool fits into the product building process. To do this we plan a possible use case that facilitates understanding.

Business Model Design

Some member of the team, worker or researcher detects a possible need in the market or within the company itself. Then, the Sponsor meets with various members of IT, UX and a PO. With a clear challenge, the business case is analysed, along with the impact it will have on the customer segment or the value proposition. Is the user and the value proposition the same? We should remember that society and the needs of users change at an exponential speed. It is also important to visualize the environment, how it will affect the solution and if there are similar or substitute products.

It is convenient to analyse the target market size and finally build the business model canvas with all the information.

Customer Development:

In this phase we must validate the proposed business model. For this, we will use customer development techniques and agile product development jointly and cyclically.

It is very important to empathize with the end user, to know how they think, what they feel and what is the root of their problem.

Next, a first User Story Map is proposed to facilitate the construction of a first list of high-level functionalities (epics) that can be evaluated in a roadmap that facilitates planning and an approximate budget.

SmartCLIDE can offer an environment that facilitates the monitoring of the project at all times from very early stages like these, from planning to production.

The team starts working on the sprints. UX proposes some first Wireframes from where all the functionalities and UML diagrams are finally extracted. It is important to integrate SmartCLIDE with other tools that allow the information of all the phases of the process (initial or later) to be useful for the software that is being built. **All information must be part of a linear, constant and “seamless” process from start to finish.**

The union with other tools such as Jira or Confluence for the management and monitoring of the project by the entire team, or Axure¹³, Zeplin that have a greater impact at the Front-end and Back-end level with the specific functionalities that have previously been tested with the user.

Agile Development:

Developers evaluate the tasks they can perform during the sprint. SmartCLIDE detects functionalities and explores code in existing repositories to facilitate code reuse. In addition, an assistant based on artificial intelligence helps the team to improve the coding with warnings of possible security flaws, thanks to the integration of the tool with the acceptance criteria. The quality of the software is an important aspect.

At the same time, the repositories are connected to the cloud where there are pre-configurations of containers that facilitate microservices and the production of work.

At the end of the Sprint, the user tests are carried out in order to verify if this increase in functionality is really what the user needs. In the test, feedback is received from the user, collected and integrated into SmartCLIDE in the form of diagrams, and incorporated into the work flow again in the next Sprint.

The information of the process is recorded in SmartCLIDE and builds the basis for the following Sprints.

The life cycle of the product or service is alive and this adaptation process is continuous and infinite, because it will always be able to cover a real customer need.

Example:

In a logistics company, the need to optimize the flow of routes carried out by couriers is detected. The objective is to deliver packages in less time, increase the number of deliveries per day and reduce costs for trucks and personnel.

¹³ <https://www.axure.com/>

User Need. Optimization of delivery routes.

Innovation level. Core. Internal business improvement.

In the first phase of the project, the Sponsor and the team visualize the impact it will have on the business model and the target customer segment, the relationship with it or the necessary key resources, and even, if a key association with another business is necessary.

The team identifies three distinct parts.

1. Collect data. To optimize the routes, it is necessary to have a series of data to support decision-making:

- Destinations and addresses of the stops;
- GPS position at all times;
- Number of stops/place;
- Time of each stop;
- OK or KO on delivery;
- Number of attempts to get delivery OK;
- Vehicle, fuel, parts, wheels, etc. ;
- Traffic;
- Meteorology.

2. Construction of a **dashboard** that shows the collected data, statistics, vehicle use, etc., individually, globally and grouped under different filters.

3. Given the complexity for the elaboration of the new routes according to the data collected, it is planned to build an algorithm based on machine learning that learns about the data collected and continues to refine the routes. Decision making can be verified by the person in charge.

In the context of this example, which follows an agile product development life cycle, one of the use cases that shows how the SmartCLIDE environment works is “*Getting all trackings related to a specific courier, date and status*”.

Actor: A Product Owner

Preconditions: The user has been previously logged in. Therefore, the system knows the user is a PO working in tracking services

Normal flow:

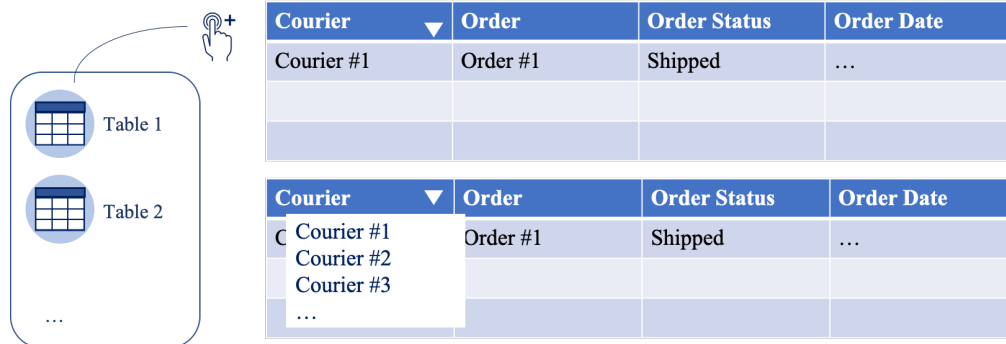


Figure 3: Normal Flow example

- The PO drags the desired table (Table 1) from the data sources panel into the canvas
- Then, the IDE shows a pre-visualisation of the table with its fields shown.
- The user selects the fields (e.g. “Courier”. “Order Status” and “Order Date”) then the user selects/types the desired one.
- The system monitors the actions of the user from the start, then it infers what the user is doing, what it is associated with, its specifications, tests, code, etc.

Alternative flow: The PO uses a Gherkin-like syntax, for example: “As a Tracking PO I want to obtain all trackings of courier X, state Y and date Z”.

3 SmartCLIDE Features

The general scheme of the SmartCLIDE architecture is illustrated in Figure 4. In this figure the main features of the SmartCLIDE solution are depicted, such as the Smart Assistant and the SmartCLIDE workflows.

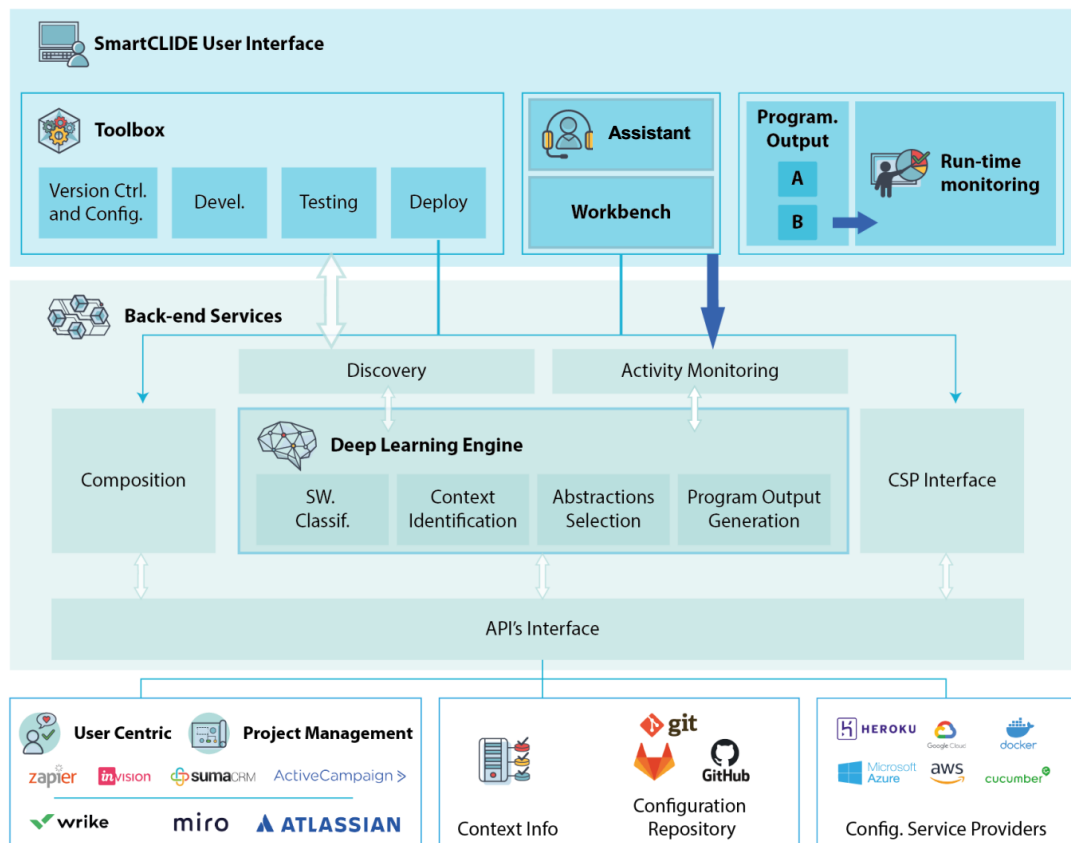


Figure 4: The SmartCLIDE Feature Map

The big difference with the initial approach is the inclusion of an API's that manages the possibility of integration with other tools that allow the information of all the phases of the process (end to end) to be useful for the software that is being built. Thereby, the focus within the SmartCLIDE project will be on developing the SmartCLIDE technologies including the API interface.

This aspect increases the power of SmartCLIDE and helps to build more robust user-centred solutions, that manage all the information.

The goal is to incorporate validated knowledge into the project or product in a "seamless" start-to-finish process.

3.1 Smart assistance in the IDE

The purpose of the Smart Assistant is to help both technical and non-technical users during the Software Development Stages. That is, ranging from the requirements and design to the deployment of the services.

This will be performed in two different ways: (i) through a menu in the general interface of the IDE, where all active suggestions will appear classified by software development stage, (ii) by a context menu attached to non-intrusive, but visible enough, marker where the suggestion makes sense. With this interface, interruptions to skilled developers are minimised, as they do not need assistance on a regular basis, but in case of an explicit request of help, while keeping away from a very detailed design with lots of information, confusing for the non-programming crew. The IDE layout can be adapted depending on the kind of user, changing the type and "deepness" based on the users' skills.

The Smart Assistant will potentially provide recommendations on the following topics to help users during software development:

- During the Requirements & Design phase:
 - Guidance to define requirements and user stories using the Gherkin syntax.
 - Reusability of previous resources, such as BPMN schemes, or components that probably are close to being needed in a task flow, based on previously existent schemes.
- During the Development phase:
 - Code analysis and syntax checking.
 - Code autocompletion based on the developer behaviour and context.
 - Guidance through the process of pushing changes into a version control repository.
- During the Testing phase:
 - Interpretations of the results from the tests.
 - Suggestion and offer of a set of acceptance tests, preferably based on initial Gherkin specifications.
- During the Deployment phase:
 - Suggestions or guidance through the deployment according to the Services Deployment component, such as the convenience of an architecture.
 - Suggestions on configurations for deployment based on services properties.

- As an all-round helper, advising on the next steps to be taken based on the current context of the user.

The Smart Assistant is supported both by the DLE and potentially by IDE plugins and guided by the monitoring of user actions.

3.2 Main SmartCLIDE Workflow(s)

Key features of backend services and service discovery

The key components, along with their most important features, are presented in the following list:

- **SmartCLIDE RESTful API Gateway**
In a complex ecosystem like the one of the SmartCLIDE project, a module responsible for the orchestration of the overall operation is a necessity. Therefore, the design and development of a communication/network gateway, namely the SmartCLIDE REST API is proposed. It will be a RESTful API that will expose the several SmartCLIDE functionalities to the outside world. In particular, the SmartCLIDE REST API will be responsible for:
 - Exposing the SmartCLIDE platform’s functionalities to the users through the designed and developed user interfaces.
 - Integration with external tools.
 - Routing, data-transformation and load balancing (where applicable).
 - Ensuring adequate security level using best industry standards such as OAuth 2.0.

For building the SmartCLIDE REST API the Python Eve RESTful Framework [6] is proposed.

Message Oriented Middleware (MoM)

This component will be responsible for the inter-component communication within the SmartCLIDE platform. It will be implemented as a message broker and provide asynchronous communication functionalities based on the publish-subscribe (i.e. pub/sub) pattern. The MoM component will be responsible for providing the following three functionalities:

- **Message routing:** MoM should support several message routing policies and message delivery guarantees (e.g. at-most-once and exactly-once).
- **Message transformation:** MoM will transform the data/messages from the sender’s native format to the receiver’s native format.
- **Message validation:** MoM will be able to check if the exchanged messages, either at the sender’s or at the receiver’s end, comply with a specific format.

There are several message broker's software available, with popular choices being Apache Kafka [6], Apache Qpid [7]. They support the following critical features:

- **Loose coupling of the components**, which will enhance their developability and maintainability.
- **Increased scalability**, as pub/sub pattern can serve multiple senders and receivers simultaneously.
- **Increased security**, as the MoM component will implement specific security policies.
- **SmartCLIDE Databases**
A key characteristic of the microservice architecture is that the services are loosely coupled and communicate only via APIs. One way to achieve loose coupling is by each service having its own database. Thus, each microservice will be in charge of controlling and managing its own database and importing and converting any external data sources through appropriate wrappers, adhering to the SmartCLIDE data model.
- **Containerization and deployment mechanisms**
The developed backend services along with the other modules of the overall SmartCLIDE platform will be containerized in order to be easily deployable. This process will utilize the dominant Docker [16] mechanism for this purpose. Additionally, the overall deployment mechanism of the SmartCLIDE platform will make use of the open-source container-orchestration system Kubernetes [17].

4 SmartCLIDE Services and Components

4.1 Discovery of Services and Resources

This section explains the approach described in D1.2 for the discovery of services from a conceptual perspective.

Service Discovery is the IDE component responsible for three features: retrieving the existent services' data from a previously determined array of sources, store them at the SmartCLIDE Service Registry, and act as an interface with this last component regarding service access (e.g. provide them to the classification implementation running at the DLE). In other words, it will act as a proxy for the service request queries, handing over the results to the Services Creation and Composition component, the DLE, or the Security Component in case they are requested.

Services will have to be fetched from the user-specified sources before their classification and storage. There are some key differences between regular service querying to an existing registry component, and scraping services from other sources not aimed at this purpose, such as web pages or code repositories. Hence, two approaches will be followed:

- a) Query regular, registry-based services
- b) Scraping of services from HTML sites

Regular service registries take the responsibility of communications amongst/with services, monitoring the health of existing processes and broadcasting the existence of newly available ones along with their endpoint. Configurations and extra information on service properties, such as functional and non-functional (QoS) parameters, are to be desired, in the form of ontologies. Most of these registries have a REST API which allows to query information regarding existing services along with their current status, and enabling to interact with them for management purposes.

The idea behind this approach is to query registries to retrieve services' status and the most significant amount of information, independently from the kind of ontology or descriptor –even missing- which contains it.

This information will be used by the classification process in the DLE to extract and store services in an IDE-Registry, composed by the service data + classification results in a structured/standardised way. This will be queried by the Service Discovery to retrieve and hand over the best suggestions to the Service Creation and Composition component. This approach is chosen because the combination of Service Discovery and Classification executed for each service discovery request will not potentially be fast enough to deliver results in a proficiently enough manner.

Likewise, public web pages containing links to services will be taken as a reference to develop both a process which extracts them given an initial URL and to compose a

dataset to train DLE classifiers. Code repositories will be queried in case services have their matching code available.

Later, this process will be tested against repositories and particularly considering those belonging to the use cases with the aim to a) enhance the classification methodology b) adjust their behaviour to different needs. Finally, new services created within the IDE will enforce the user to add the information needed to deal with classification for later reuse, or using direct transformations from code such as WSDL+Java and stored along with other service data.

Once the extraction is performed and the dataset is conformed, a model will be built to perform further classifications independent –as far as possible- from the structured form. More information on the classification experiments can be found at section 4.6.2.

The approach can be observed in Figure 5.

Three processes will conform to the Service Discovery:

- **Extraction** of the services from different sources along with a minimum set of features. This set will be defined with the collaboration of the project consortium.
- **Check** information fields and **save** them into SmartCLIDE Service Repository (new services/existent services).
- **Fetch** services from the SmartCLIDE Service Repository matching specific constraint parameters.

A daemon will check the repository to find unclassified services. These services will be sent to the DLE to the classification process and subsequently updated

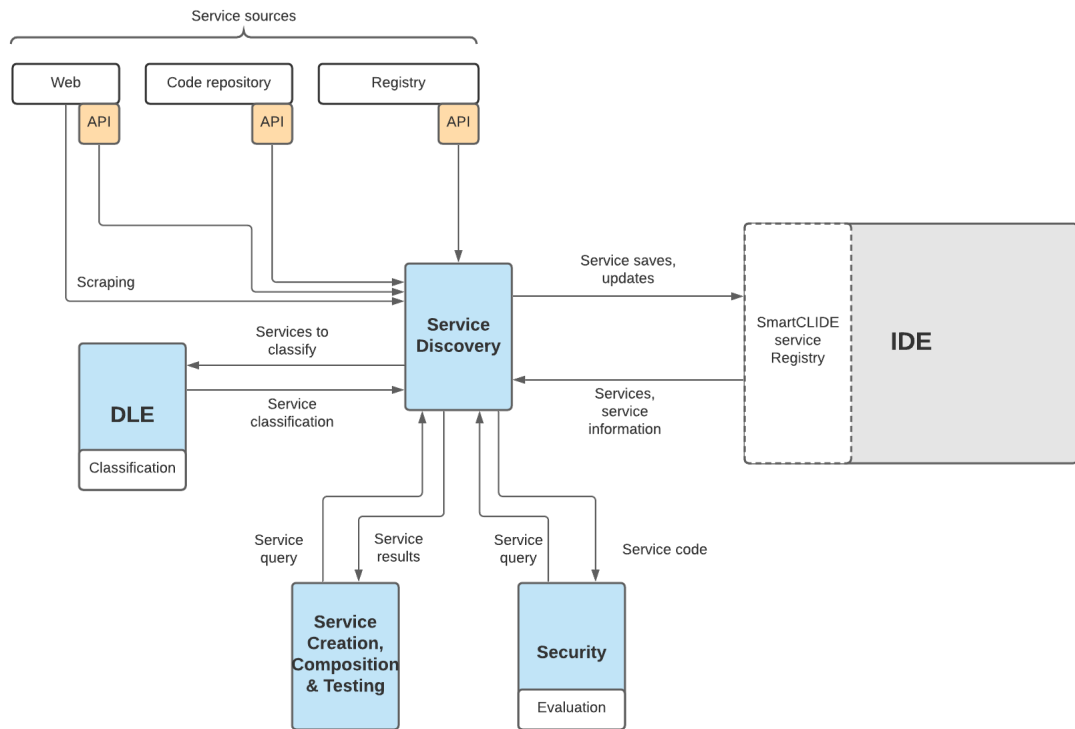


Figure 5: Service Discovery Approach

4.1.1 TRL 4 Lab Validations (Minimum Viable Product)

In microservice architectures, one of the main goals is to allow services to discover and interact with each other. Distributed platforms not only make this task more complex but also imply a challenge in terms of health check (services status) and broadcasting newly available assets. It is crucial to decide where and how to store the available information, the same as the setups needed by the applications.

Service registries extend the concept of an application-oriented to Web Services, by allowing clients or applications to access to a variety of services which match specific search criteria. Different standards, as discussed in the D1.2 Requirements Analysis, have arisen to define and standardise services' information to become a common registration and discovery method.

Here we will briefly recap and narrow their concepts down to the experimental approach, and which features will be employed.

UDDI registry aims to be a Service Discovery method using WSDL. WSDL is part of the original specification of Web Services. It can be searched in different ways, depending on the target of the query, to retrieve information. ebXML is similar to UDDI; it allows companies to find each other, define agreements and communicate using XML messages to support commercial transactions. The purpose is to act

without human intervention through the Web. It has a lot of points in common with UDDI/WSDL/SOAP.

The application of these standards must be executed in a planned manner and observing some limitations. Hence, in many cases, rules are not applied to the registries, and valuable service discovery is not performed. This fact is aggravated by the circumstance that public repositories are not that popular, being private companies and, therefore, their internal needs what stands for the service discovery implementation.

The cloud paradigm has deepened this problem, flourishing registries for each provider architecture and allowing companies to have their customised services. These services follow the standard definition but have evolved to fit the nowadays needs. For example, AWS provides integrated service discovery capabilities for containers deployed in their platform, using a component called Route53 and an auto-naming feature via DNS – to provide dynamic resolving with understandable names – and providing health checks which ensure that only alive services are returned. The usage of third parties software as an alternative is also common, like Consul¹⁴ or Eureka¹⁵.

The combination of the items mentioned above (standards, data formats, querying techniques, information delivery and communication/health methods) is what the Service Discovery component will tackle and take as a reference to query and store the services data.

In a first approach, public web service listings will be used to retrieve a dataset. This dataset will consider the heterogeneity of the data with no further references or standards; and will resolve the web service scraping for particular use cases. Besides, it will feed the DLE classification process.

A first implementation will only consider the re-utilisation of already existing and successfully deployed services. This is motivated by the fact that retrieving the code or container image for a working asset does not seem to be feasible, because of the limited availability of resources outside of a particular company. Moreover, an external registry independent from SmartCLIDE IDE component may have to be used to deploy and manage the services.

The experiment process will stand as follows. As mentioned, the system will have to be able to work both with standard and non-standard data schemes, containing the services information. It will look into predefined sources for services already deployed, or just barely available at repositories in the form of code or container images. It will look at the predefined sources (at IDE level or project level, Figure 6) and query them using an API, or scraping if plain web URLs are provided. In both cases, health checks will be queried or performed to filter unavailable items. This process will feed an internal IDE registry or database for later classification.

¹⁴ <https://www.consul.io/>

¹⁵ <https://github.com/Netflix/eureka>

In terms of web scraping, if no API is provided, each endpoint will have to be inspected to check the availability and information (descriptors) of its work. A search for descriptors will be also performed.

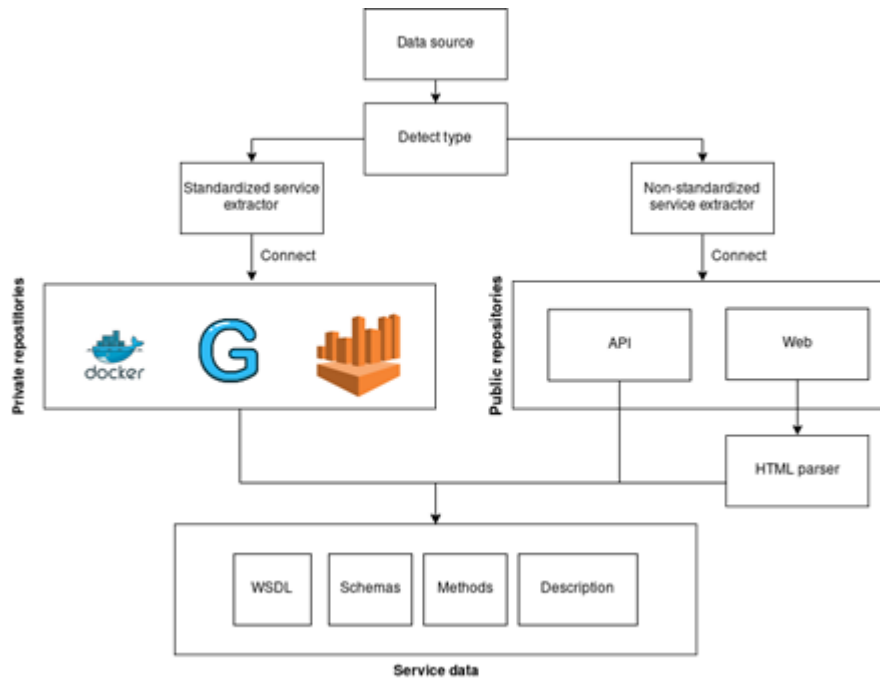


Figure 6: Service Discovery - Data Sources

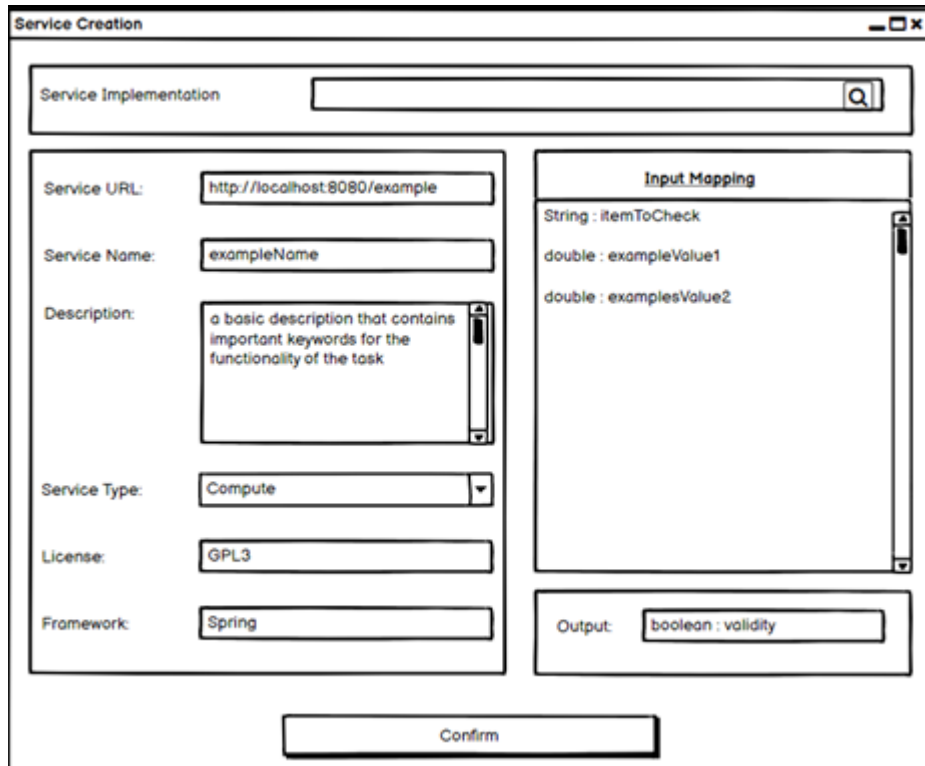
In conclusion, Service Discovery experiment will extract information from web sources, registries of already working services and potential repositories from companies; being the services' related information contained in different kinds of ontologies or other non-standard structures, such as web pages. Service Discovery will also be fed by newly created services, which will have a descriptor containing the properties required for classification, needed by the partners/use cases and filled in during their creation or updated while being coded.

Both types of services, new ones and discovered, will be stored in an internal registry/database for their latter classification by the DLE. Thus, updated services, along with their classification, will be handed over when queried by the Services Creation & Composition component in a faster way. As a side note, classification heterogeneous information will only be needed for discovered services, as new ones will be enforced to fill up a convenient descriptor to classify and reuse the services.

The interface of the Service Discovery component will have to:

- a) Request standardised data while creating or composing a service, with classification purposes.
- b) Allow to set up, at IDE level or even project level, where the Service Discovery has to search for the resources. This information will be stored at the component.

These interfaces will be as follows. Item a) is already defined by UoM in Figure 7 (detail below). In this screen, only final fields related to specific service features related to classification will have to be changed. With respect to b), the Service Discovery set up will include three tabs. One of them for regular repositories, which will contain an editable list of sources obtainable through traditional registry repositories. Another tab for public web scraping, where a base URL and some extra, including parameters for managing checks and search combinations. The third tab will contain all the sources present in repositories (Figure 8).



The screenshot shows a window titled "Service Creation" with the following fields and sections:

- Service Implementation:** A search bar with a magnifying glass icon.
- Service URL:** `http://localhost:8080/example`
- Service Name:** `exampleName`
- Description:** `a basic description that contains important keywords for the functionality of the task`
- Service Type:** `Compute` (dropdown menu)
- License:** `GPL3`
- Framework:** `Spring`
- Input Mapping:**
 - `String : itemToCheck`
 - `double : exampleValue1`
 - `double : examplesValue2`
- Output:** `boolean : validity`
- Confirm:** A button at the bottom center.

Figure 7: Service Creation

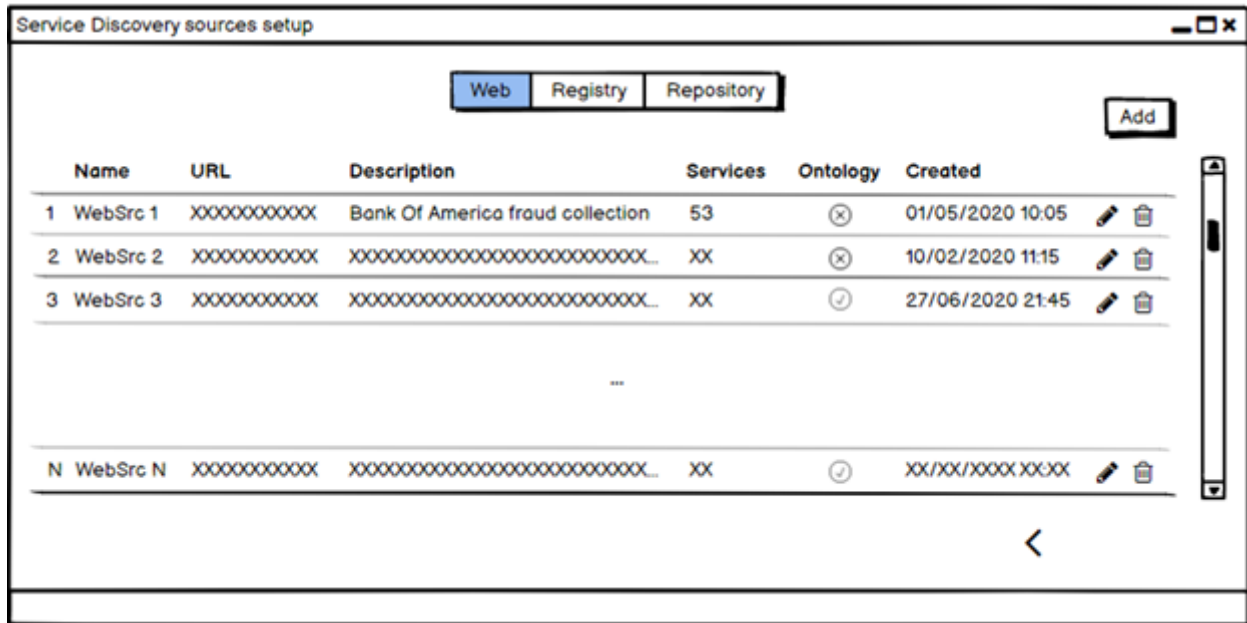


Figure 8: Service Discovery - Data Sources

The Service Discovery component will supply a REST API to connect with it and forward the service queries to the Service Registry in a homogenous way. The means of connection will depend on the Service Registry implementation (e.g. Nexus¹⁶) and its querying methods. These connections will be either directly implemented REST endpoints, or via a middleware. Sources information, as said, will be stored in an internal component storage.

¹⁶ <https://www.sonatype.com/nexus-repository-oss-vs.-pro-features>

4.2 Services Creation, Composition and Testing

In this section, we specify our approach for delivering the requirements, related to the “*Service Creation, Composition, and Testing*” component, as defined in D.1.2 “*Technology Requirements*”. We note that this document is not a technical one (details on the internals of the component will be given in D.1.5 “*SmartCLIDE Architecture*”), but only aims at defining the main concepts of the SmartCLIDE project. Therefore, diagrams do not correspond to a specific language (e.g., UML component diagrams) and the Minimum Viable Product is comprised of indicative mock-up screens.

The starting point (and main input) for defining the concept of the “*Service Creation, Composition, and Testing*” component is the requirements analysis document, as outlined below.

SmartCLIDE “Service Creation, Composition, and Testing” component shall be able to support:

1. the creation of workflows, composed from simple tasks
2. the editing of existing workflows
3. the definition of connectors for single services
4. the use of BPMN for workflows definition
5. the retainment of existing services in a repository
6. the definition of constraints and qualities of interest for a workflow and a service
7. the definition of the functional requirements of a service through an IDE-integrated XML editor
8. the definition of the licence for a newly developed service
9. the assessment of test coverage of services and workflows
10. the execution of test cases and unit tests
11. the use of linters to identify errors
12. the invocation of external (installed) tools through a command line interface

SmartCLIDE “Services Creation, Composition & Testing” component should be able to support:

1. the development of usage scenarios for safeguarding the quality requirements of workflows and services
2. the development of models for predicting the quality levels of services and workflows
3. the delivery of autocomplete functionalities for workflow definitions and service creation
4. the assessment of service and workflow maintainability and reusability
5. the performance of integration and acceptance testing, based on external tools

6. the mapping of services to virtual containers
7. the verification and validation of system configuration

Given the above, and: (a) an analysis of existing technologies and tools; (b) an analysis of the needs of pilot providers, the “*Service Creation, Composition, and Testing*” component, can be decomposed to three sub-components interacting with several other internal components and external tools. We note that 2nd level reliance to external tools, libraries, or technologies, e.g., the one from “*Smart Assistant*” to OWL, are only presented in the corresponding sub-section. An overview of the “*Service Creation Composition and Testing*” component is outlined in Figure 9: sub-components are presented as ochre boxes (termed as Managers), the main SmartCLIDE (external) components are presented as blue boxes, whereas external tools, libraries, or technologies as gears. The continuous arrows denote dependency, whereas the annotations on arrows explain the request. As explained before, Figure 9, does not correspond to the detailed architecture, but only illustrates the relevant concepts.

Workflow Manager: The *Workflow Manager* will be responsible for handling all requirements related to the workflow (i.e., 1-4, and 15-16). The *Workflow Manager* will be the entry point for this component, activated for any new project. For the case of BPMN-based workflows, the *Workflow Manager* will use technologies similar to jBPM and Kogito¹⁷ for exploiting: (a) BPMN and DMN modellers; (b) creating a Maven project representing the complete workflow; (c) running the workflow; and (d) interacting with the workflow for testing and QA purposes. As explained before, since *Workflow Manager* is central to this component, it interacts with (usually initializes) all other managers or SmartCLIDE components. For example, it invokes the *Smart Assistant* to get suggestions for autocompletion of the workflow.

Service Manager: The *Service Manager* will be responsible for managing the services that are included in the Workflow (identified through Service Discovery, or developed as new in the SmartCLIDE environment). To this end, it will cover requirements: 5, 7-8, 15-16. A service in the repository will be represented using a predefined ontology schema; thus, the developed repository will be an Ontology Repository. The representation, (among others) will include service name, description, inputs, outputs. The *Service Manager* will invoke the *Service Discovery* component, and an online *IDE* (e.g., Eclipse Theia) for creating services (in the form of a swarm of Microservices—e.g., Kubernetes, or Docker Swarm), as well as the *Smart Assistant* to get coding templates for newly created services.

Testing and QA Manager: The *Testing and QA Manager* will handle requirements: 6, 9-12, 13-14, and 17-19. This manager will be invoked from the other two managers, and accordingly will be responsible for specifying the testing and quality assurance process for services and the workflow as a whole. The goal of this manager will be to set up and execute the testing process, to set up and execute maintainability analysis, and to set up run-time simulation and monitoring. The execution of the last part of QA will be handled by the *Run-time Simulation &*

¹⁷ <https://kogito.kie.org/get-started/>

Monitoring component. Given the fact that service specification will be Ontology based and workflow specification will be based on BPMN, any QA process will need to rely on corresponding parsers. Regarding testing, we will rely on jBPMN, whereas for maintainability analysis on software quality models.

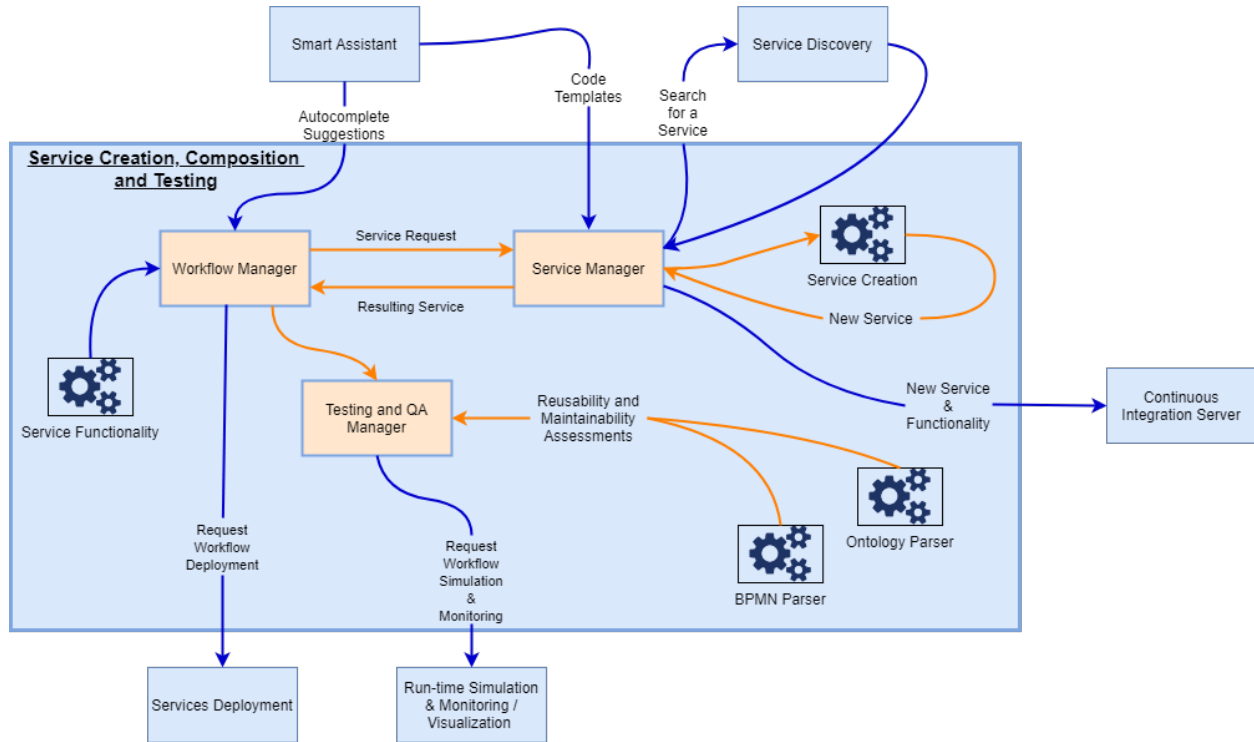


Figure 9: Overview of the “Service Creation Composition and Testing” Component

4.2.1 TRL 4 Lab Validations (Minimum Viable Product)

After having analysed the initial requirements gathered from the industrial partners, as well as their importance, we present the initial mock ups of how the SmartCLIDE platform would look and feel. SmartCLIDE would be a cloud platform in which the user will have access by a web browser. The basic functionalities UoM has to provide are Service Composition, Service Creation and Testing and Integration. We have created mock-ups for each one of these functionalities by presenting an example of usage. The proposed mock-up screens have been developed based on the INTRASOFT pilot case. Along the project, we expect to integrate the feedback and requests from all the other partners. Prototyping (through mock-ups) can support an efficient presentation of the platform that takes into account the requirements, in the sense that they visualise the basic functionality of a system. However, we need to note that none of the following suggestions are final and by no means represent the final user interface of the platform, since the specification and architecture of the SmartCLIDE platform will be finalized in later deliverables.

The workflow starts from the *Service Composition Functionality*. In Service Composition, the user can draw a BPMN diagram (a flow chart) that depicts the

functionality of a new process. In Figure 10, we present the main workspace of Service Composition and the options that it provides.

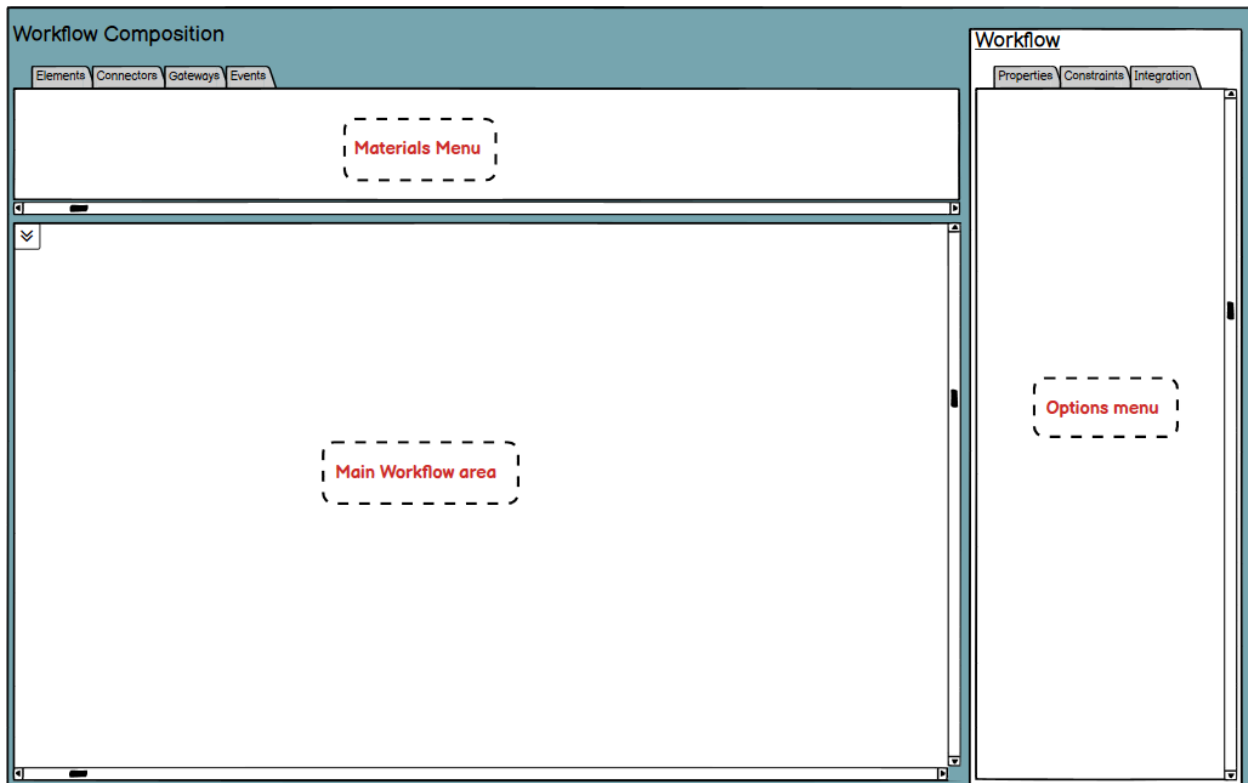


Figure 10: Main workspace for our Process Decomposition - Workflow Composition tool

The GUI of the “*Services Creation, Composition and Testing*” component is composed of three main areas:

- The Main Workflow area is used for the composition of the Workflow. Here, a process is defined with the use of nodes and the relations between them. Each node represents a service and the relations between them can be either sequential or include conditions.
- The Materials Menu provides the necessary tools to create and visualize the Workflow.
- The Options Menu contains the information, options and general properties of the workflow and all of its nodes.

Next, we focus on how a workflow is built. In Figure 11, we present the elements, the nodes and services that the user can add to a workflow. For example, there are elements for a manual task, a user task, a script task, a business rule, a service task, or a send or receive task. These elements will support the Service Discovery process, in the sense that if a user chooses a scripted task then the Service Discovery can look only for this type of services in the database. In Figure 12, we present the second tab of the Materials Menu that contains the node connectors (services). Some examples of possible connectors are sequence flow, flow fork, association and data association. In Figure 13 we present the third tab that contains the Gateways. Gateways are a

decision option for example an *if-then* rule. Some of the provided options are: the basic, the parallel, the inclusive, the exclusive, the complex and the event based. In Figure 14, the last tab shows the Events. Some options are the boundary, the parallel, the inclusive, the exclusive and the complex.

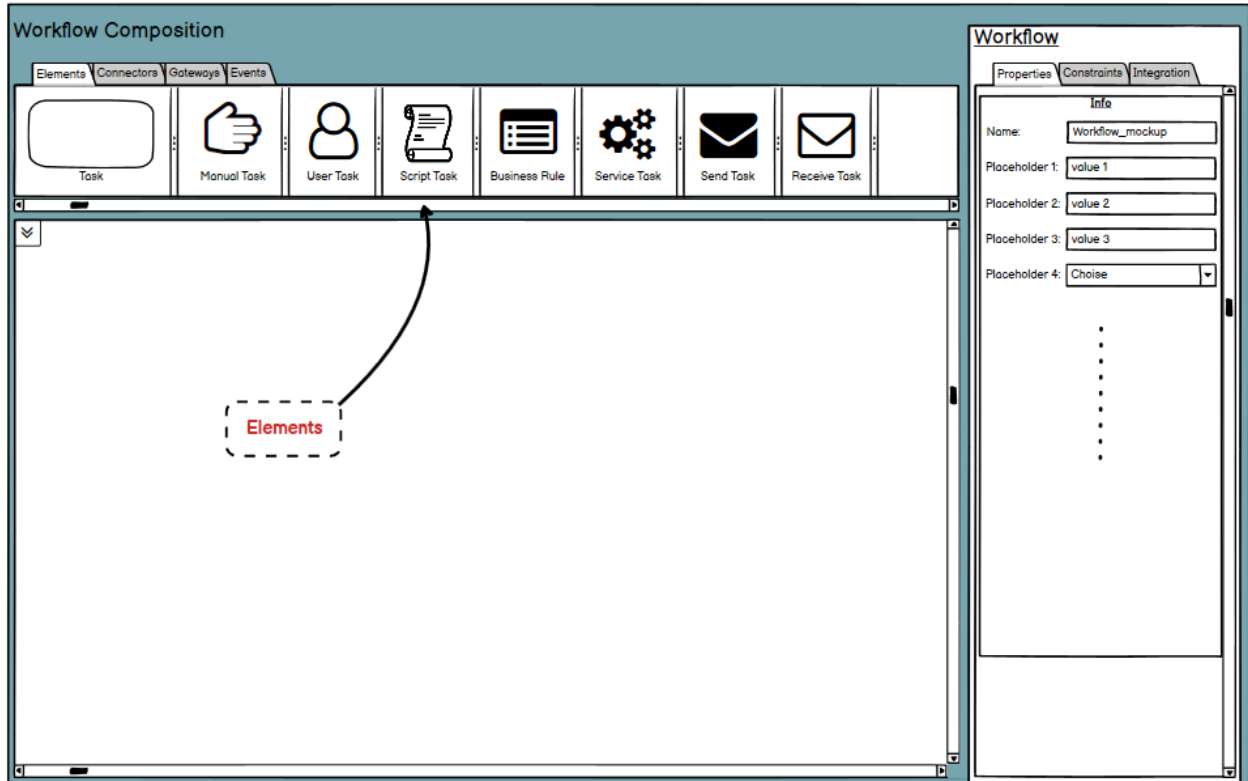


Figure 11: Elements

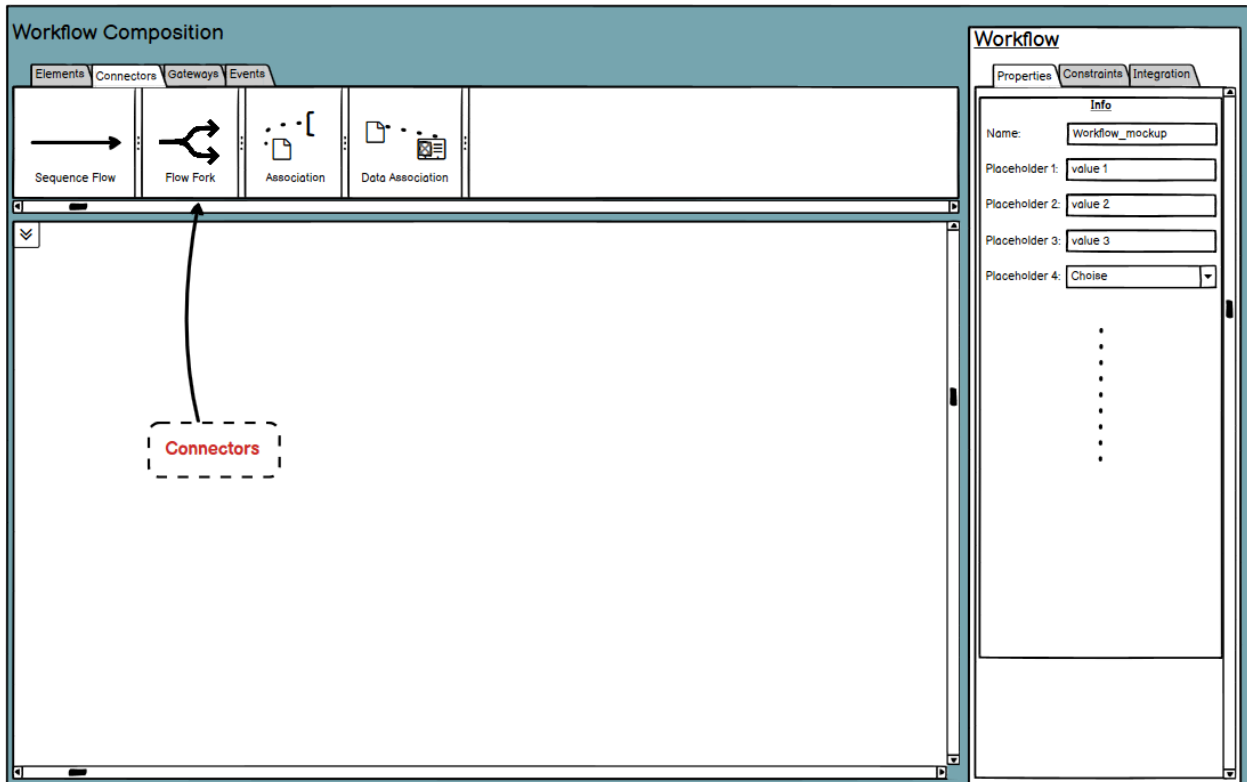


Figure 12: Connectors

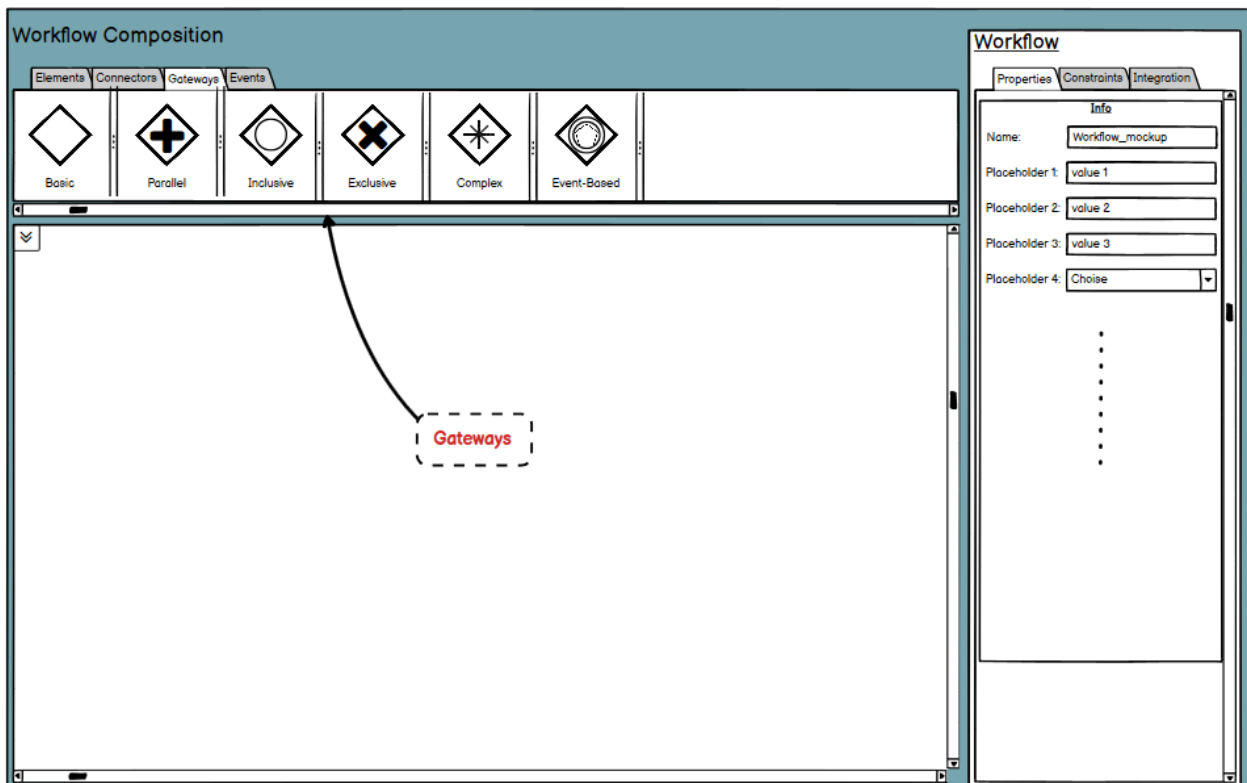


Figure 13: Gateways

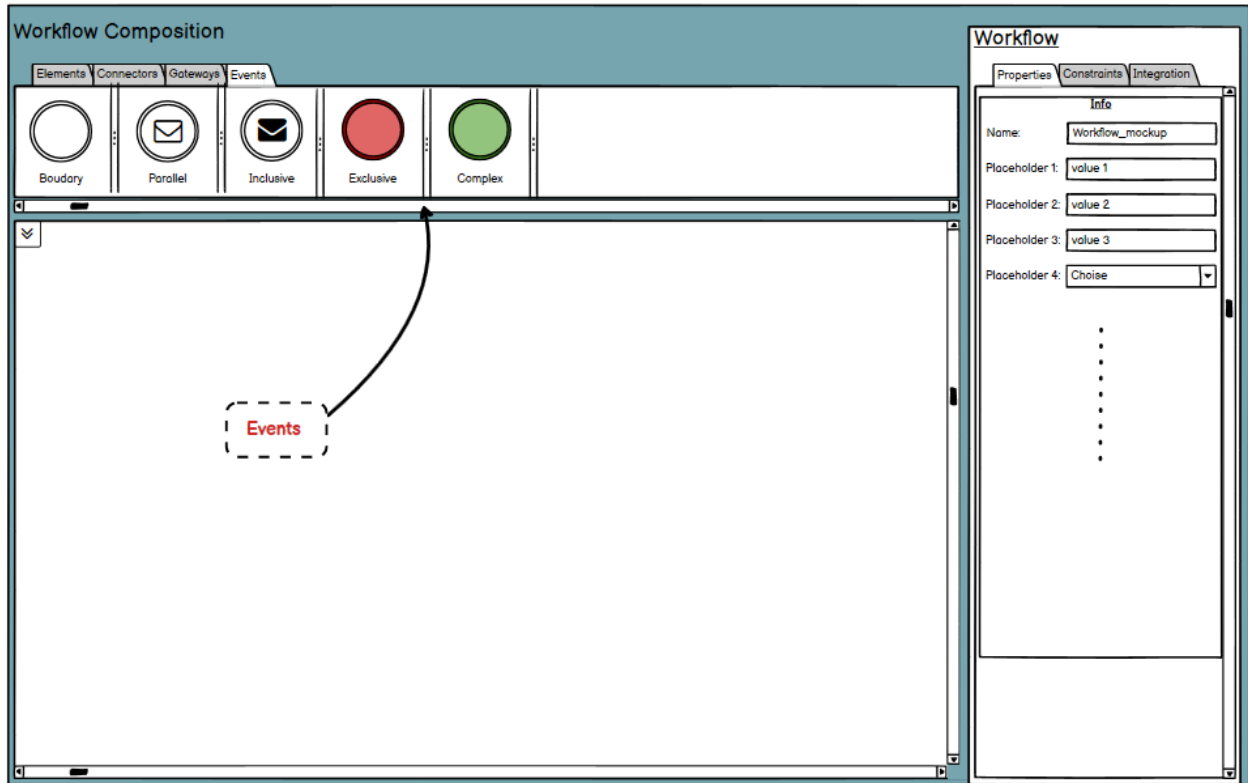


Figure 14: Events

An element that is common for all IDEs is the *Project Explorer* (see Figure 15). The project explorer button allows the navigation through project files. It is collapsible, with the ability to Lock or Unlock the collapse function. In a Locked state, the user has the ability to interact with the workflow, its nodes, properties etc. without it collapsing it enables ease of use and faster navigation. In an unlocked state, as soon as the explorer window loses focus, it collapses.

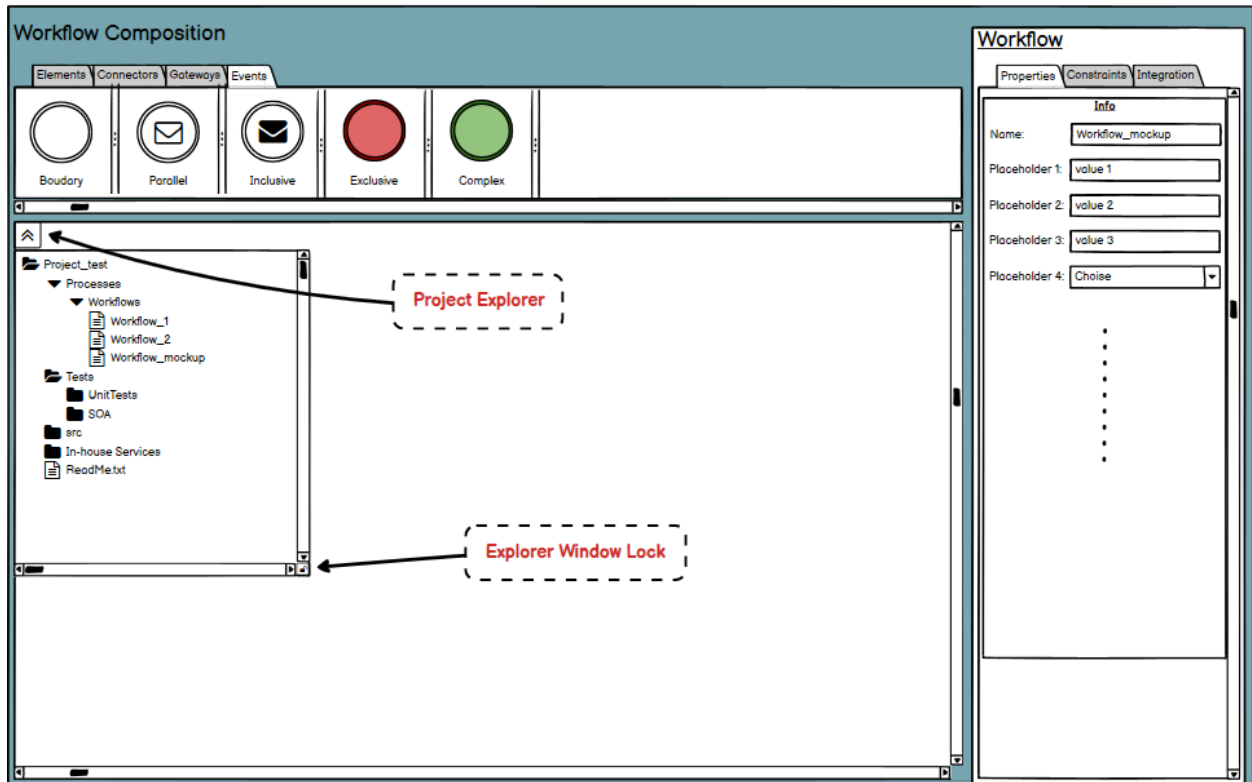


Figure 15: Project Explorer

An example of how a user can create a new workflow is shown below. The first step is to drag and drop an element from the *Materials Menu* and the tab *Element*. For example, suppose that he/she chooses the element Task, he/she clicks on it and drags and drops it on the Main Workflow Area as shown in Figure 16. The circle is the start node and it is the default node from which a workflow starts. The next step is to connect the start node and the Task node with a connector from tab *Connectors* as shown in Figure 17. This connection implies that when the process starts the first node (the node that was just dragged and dropped) is the first service to be called.

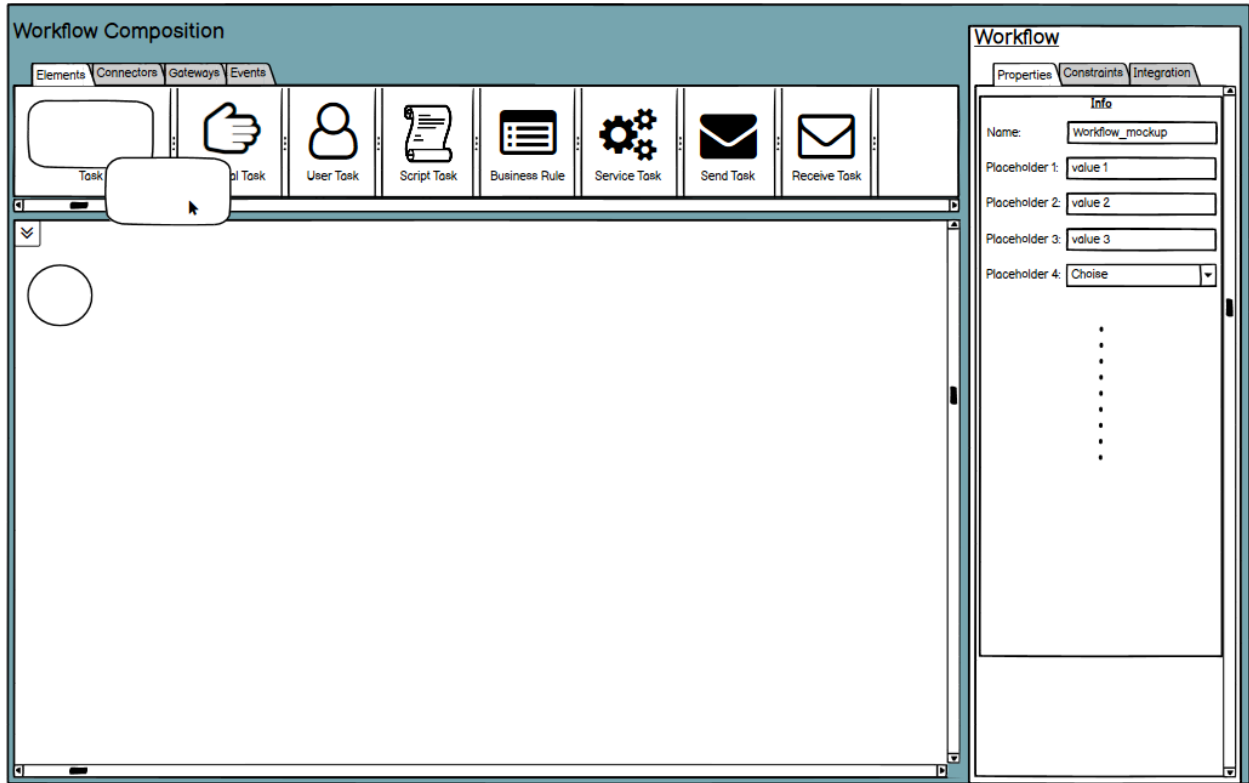


Figure 16: Drag and drop capabilities for the task decomposition

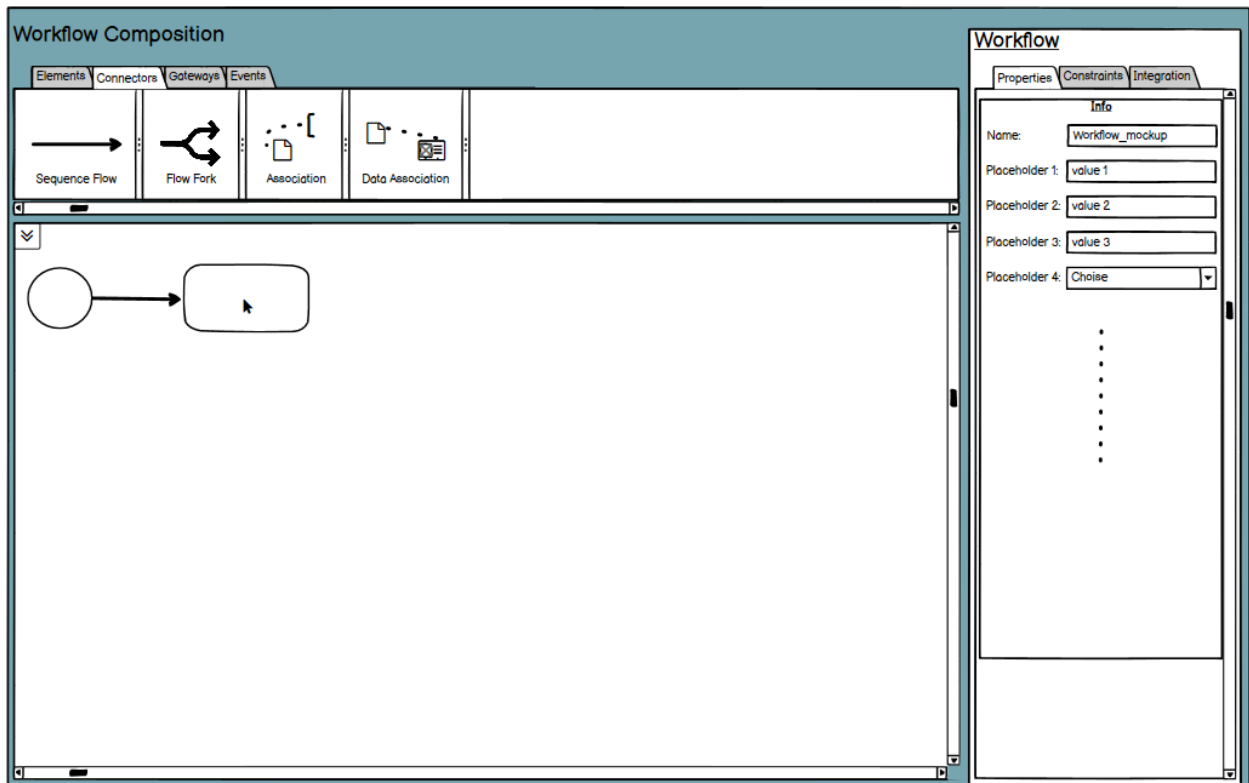


Figure 17: Connect nodes with connector

Subsequently, the *Workflow Properties* panel is displayed, which consists of basic information that characterize the workflow. The exact fields have not yet been specified, so instead, placeholders have been put in their stead. Workflow Options Menu is selected by default, or by clicking on the workflow as shown in Figure 18.

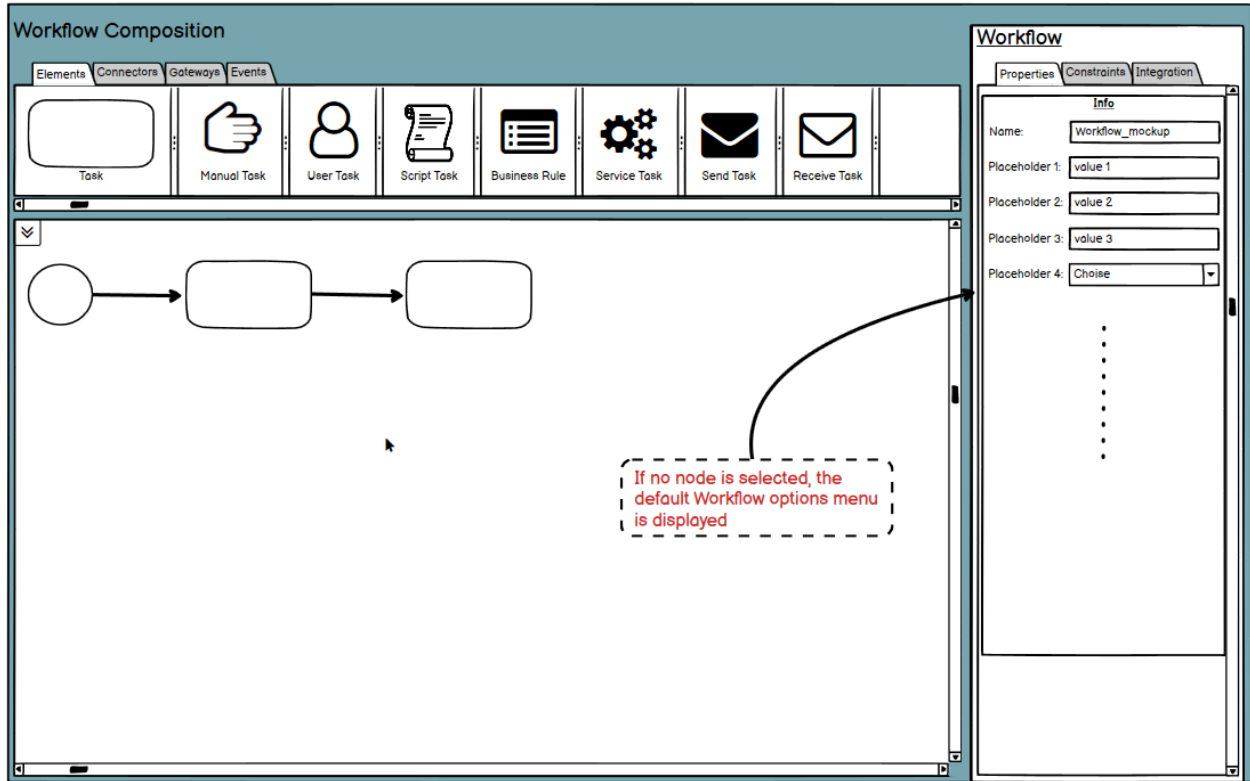


Figure 18: Workflow Options Menu

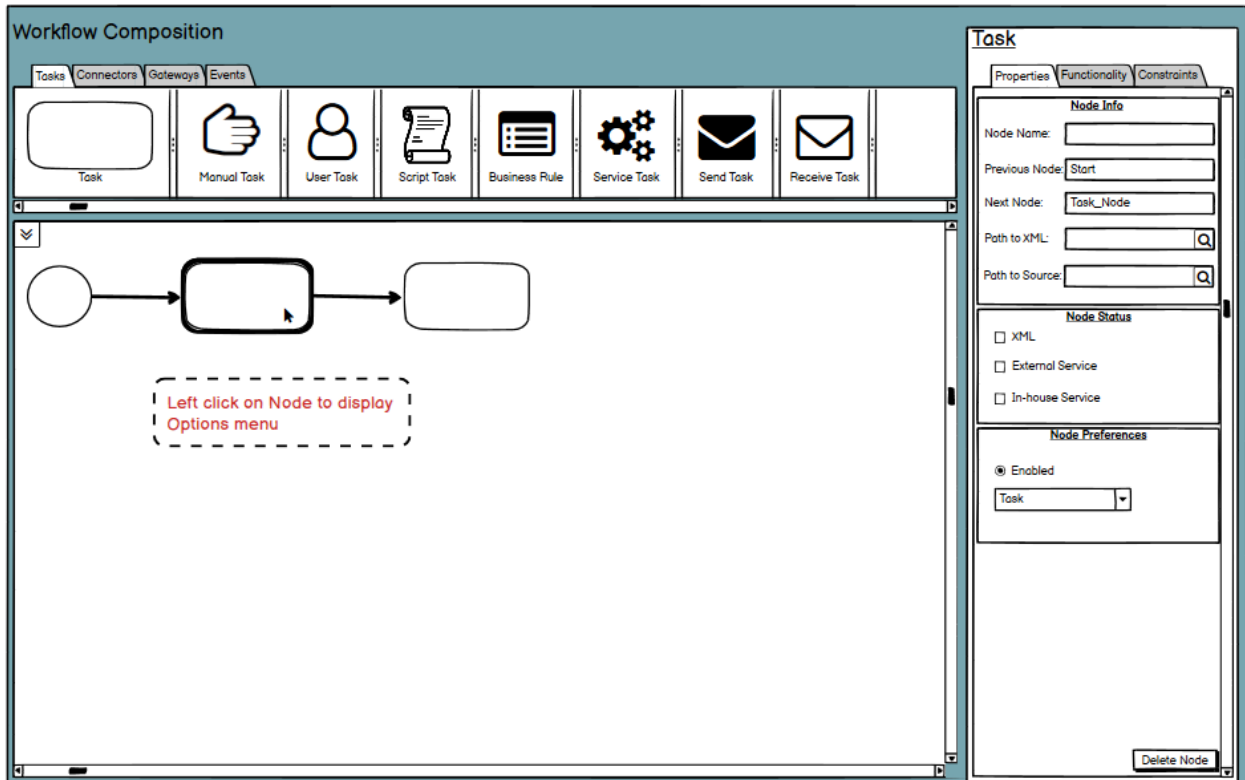


Figure 19: Node Options Menu – Properties tab

By clicking on a node, the *Node Options Menu* opens (Figure 19). This menu consists of three tabs. The *Properties* tab include some basic information about the node such as its name, its previous and its next node and the Node type (Task, Script Task, etc.). Also, the node’s status is shown. The status indicates if the XML - Functionality is completed, if an *external service* is found for the node or if an *in-house service* was implemented (in case an External service was not selected or found).

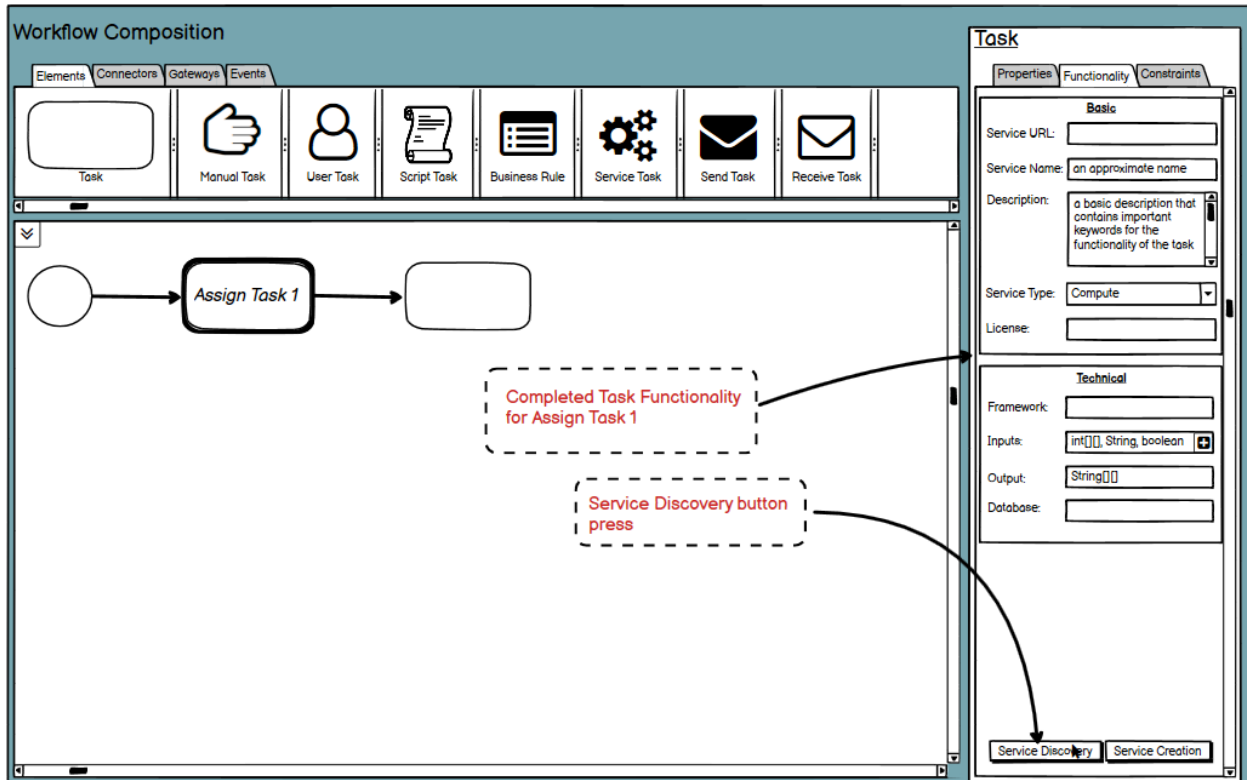


Figure 20: Node Options Menu – Functionality Tab

On the *Functionality* tab, we declare the Information and desired Functionality of the task. Some of these fields may/will be used in the Service Discovery or Service Creation process. By clicking on the Discovery Process button, the discovery process starts and the results, if any, are returned. After clicking the Service Discovery button, a new window is displayed (Figure 21). On the top panel, the Services are ranked based on their similarity to our Task Functionality values. Here we have the ability to inspect all the available Services by clicking on them, or by using the browse buttons. When a service is being inspected, it's Basic and Functional information are displayed on the bottom panel, along with its description. Once we are satisfied with a Service, we can press the final Select button, and select it.

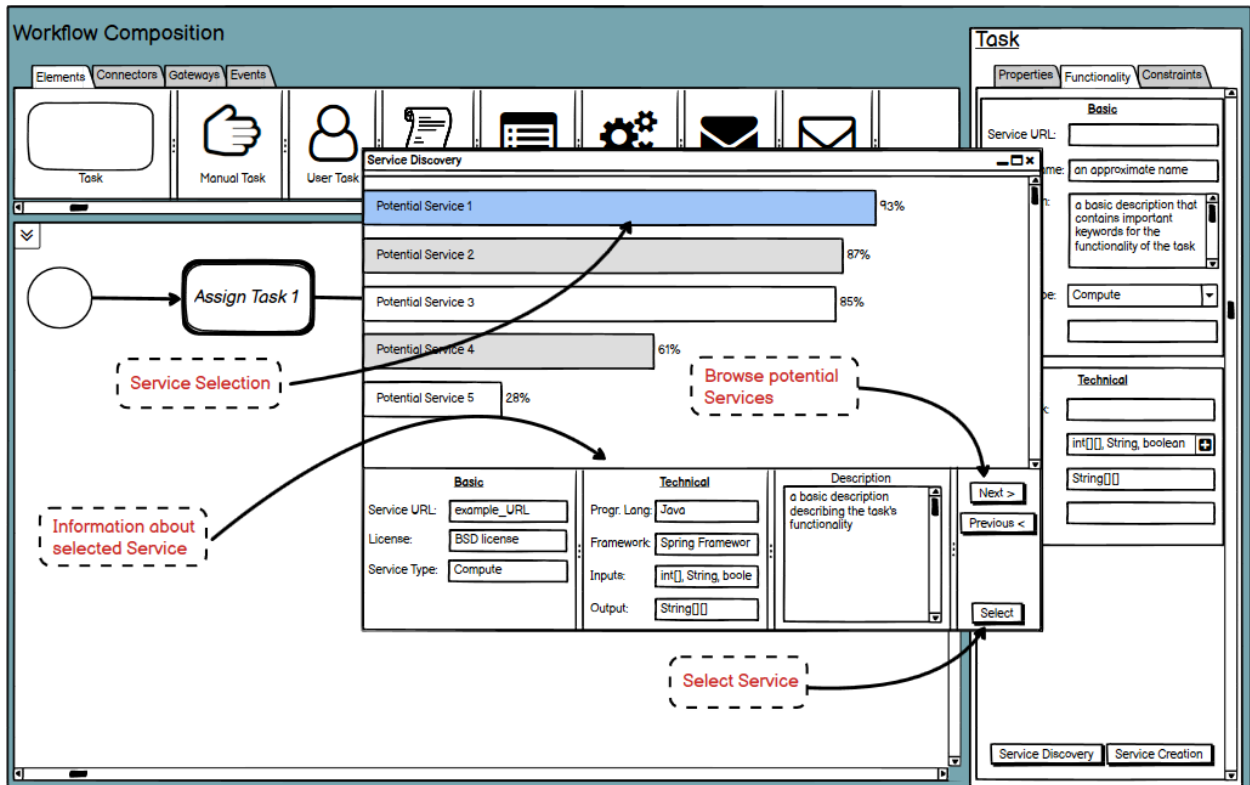


Figure 21: Service Discovery

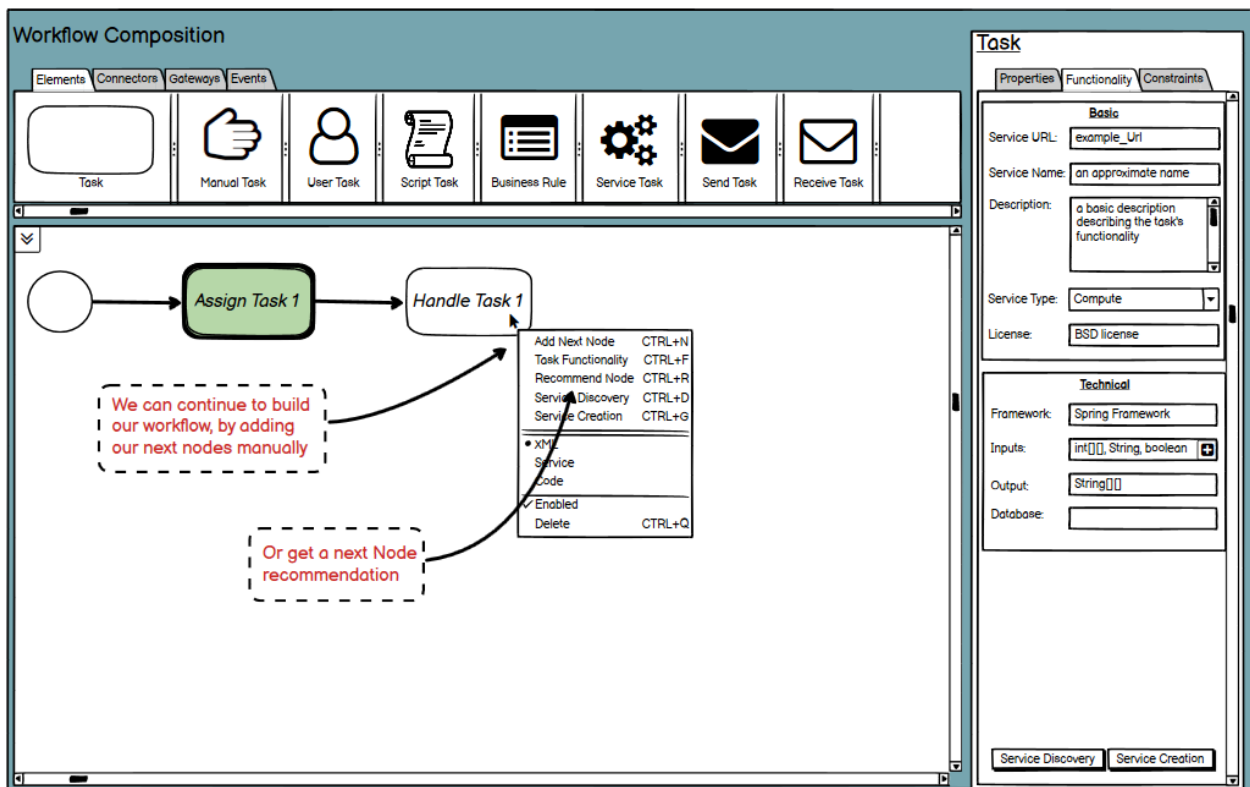


Figure 22: Node options

Afterwards, the Assign Task 1 is complete and ready for use. We now have to complete the rest of the process decomposition, using the tabbed top menu, or by using the pop-up right click menu.

The right click menu (see Figure 22) provides a more usable and user-friendly interface. It gives the ability to quickly add a new Node or use the Recommender Option in order to get a recommended one based on past decisions or workflows. It also shows the state of a Node and allows us to define the task functionality and search for a service without selecting a node and going through its properties.

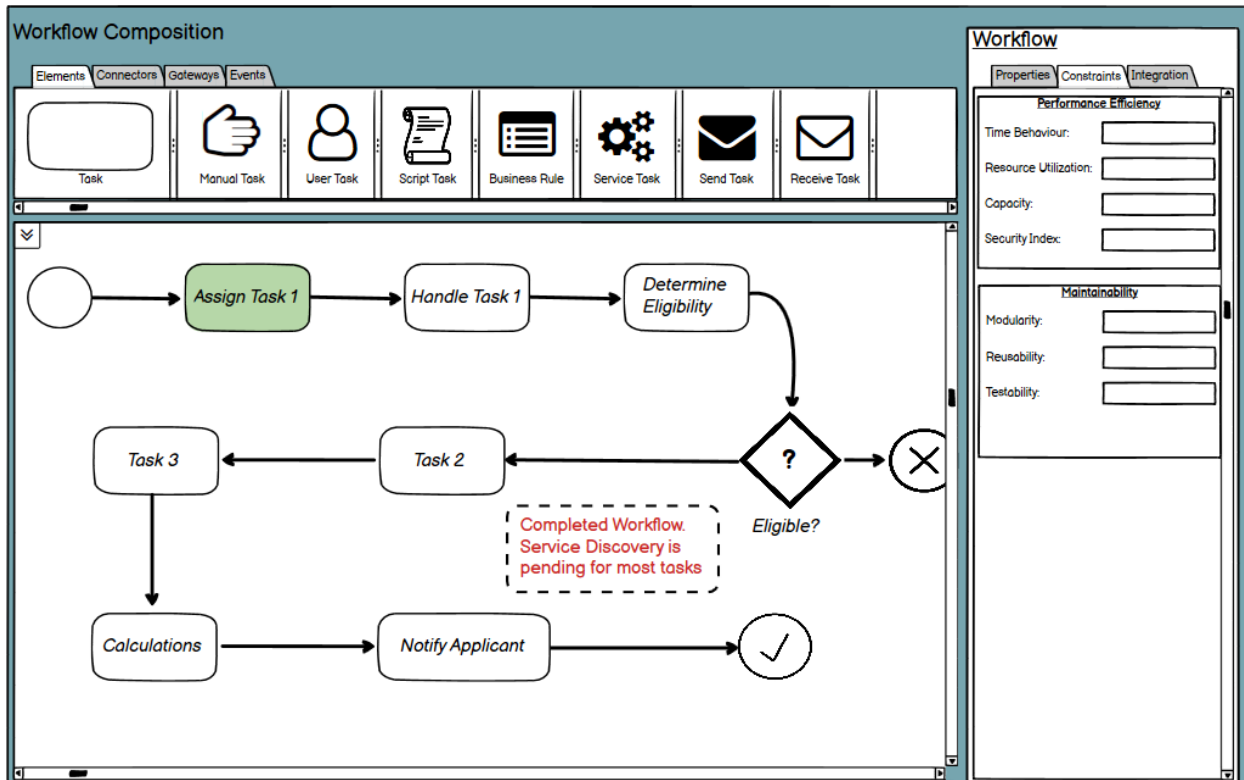


Figure 23: Complete Design Workflow

Figure 23 shows a completed Workflow with all the necessary nodes. However, most nodes still require to be paired with a Service, so the Service Discovery function must be repeated for all of them. In Figure 24, we see that the tasks with name “Handle Task 1” and “Determine Eligibility” are successfully paired with a service, but for “Task 2” the Service Discovery process has failed to detect a suitable service. This situation leaves us with two alternatives. We can either preserve the functionality of our Workflow as it is and create the required service for the Task, or we can modify it aiming to find already existing services. Let’s assume that Service Discovery did not return any results and we talked with our colleagues that reassure us that we don’t have a service like this in our repositories. In this case, we have to click the Create Service in order to create the new service as shown in Figure 24. Once the Service Creation window opens, we see the information we had previously entered for the Service Discovery (Figure 25). Through this UI, we have the ability to review and edit any previously completed fields. For example, adding a more comprehensive description to our newly made service. To pair the information to a

service, we have to locate the service file through the top browse menu. We then need to map the input and output of the service, and do the final review. Once everything is in order, we press the Confirmation button.

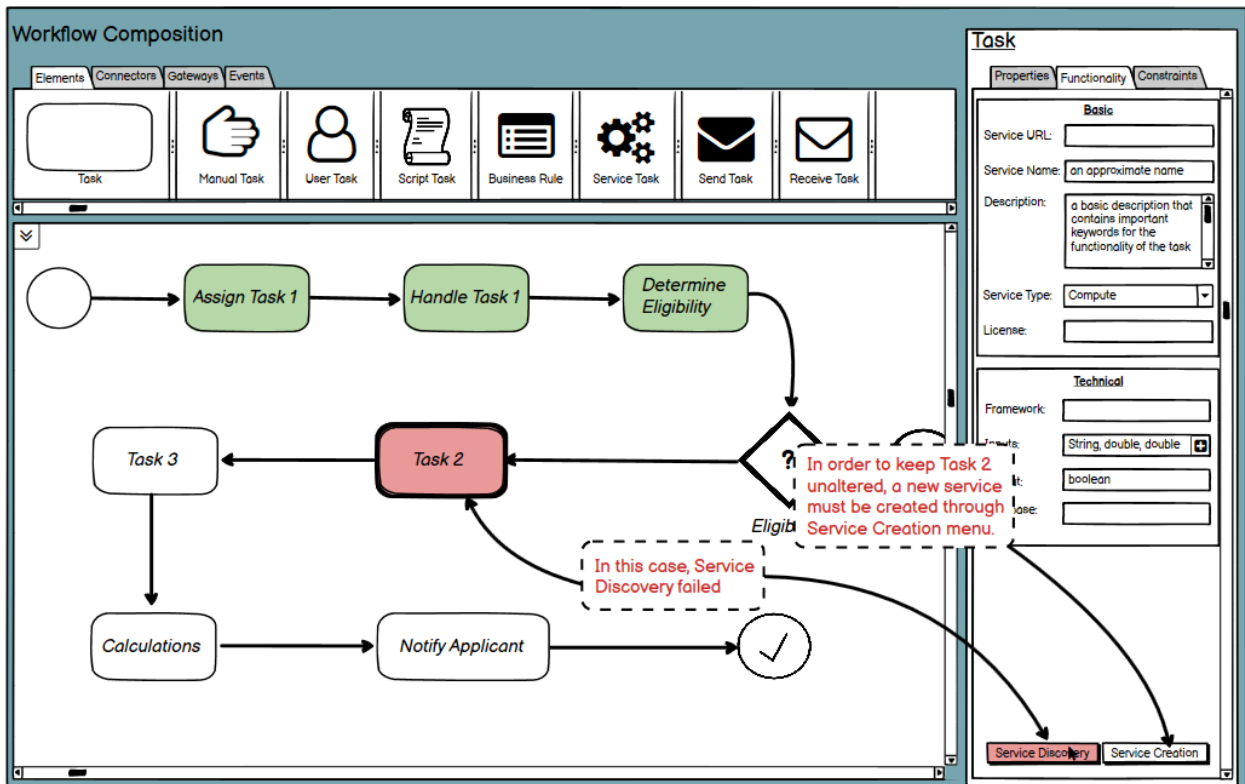


Figure 24: Service Discovery failed

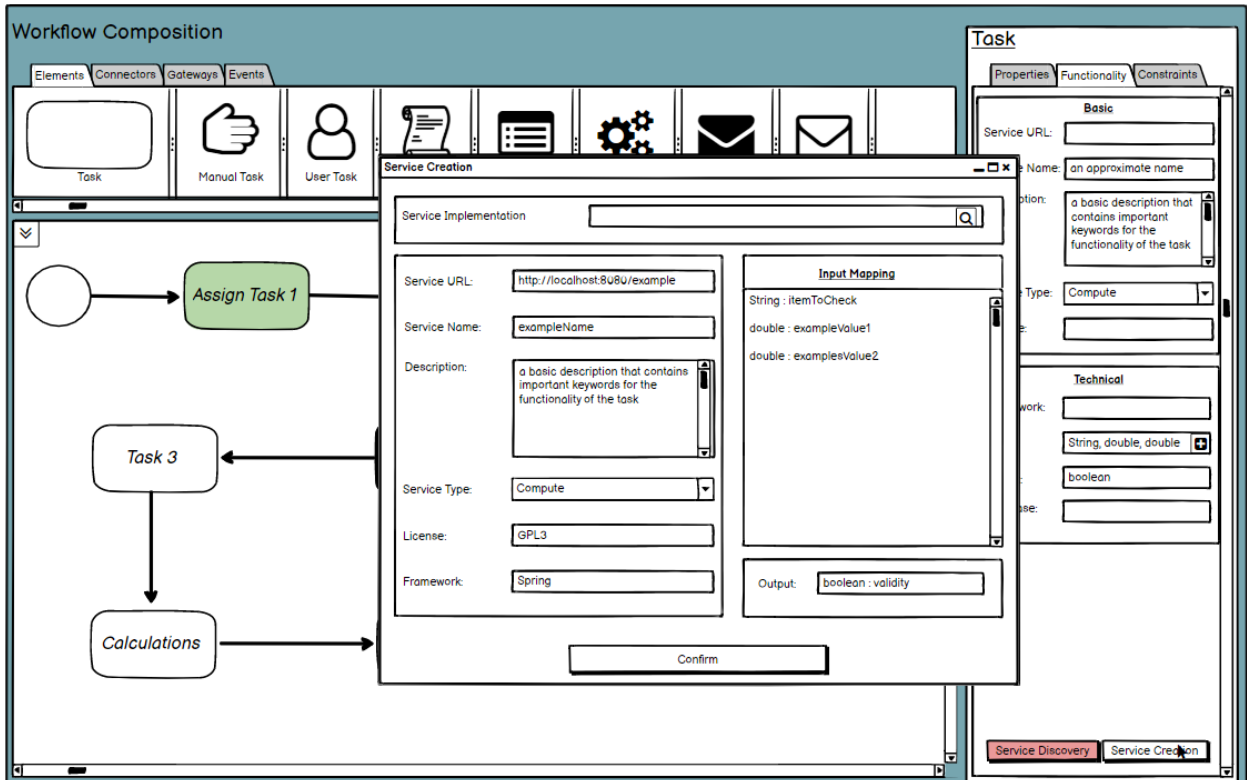


Figure 25: Service Creation

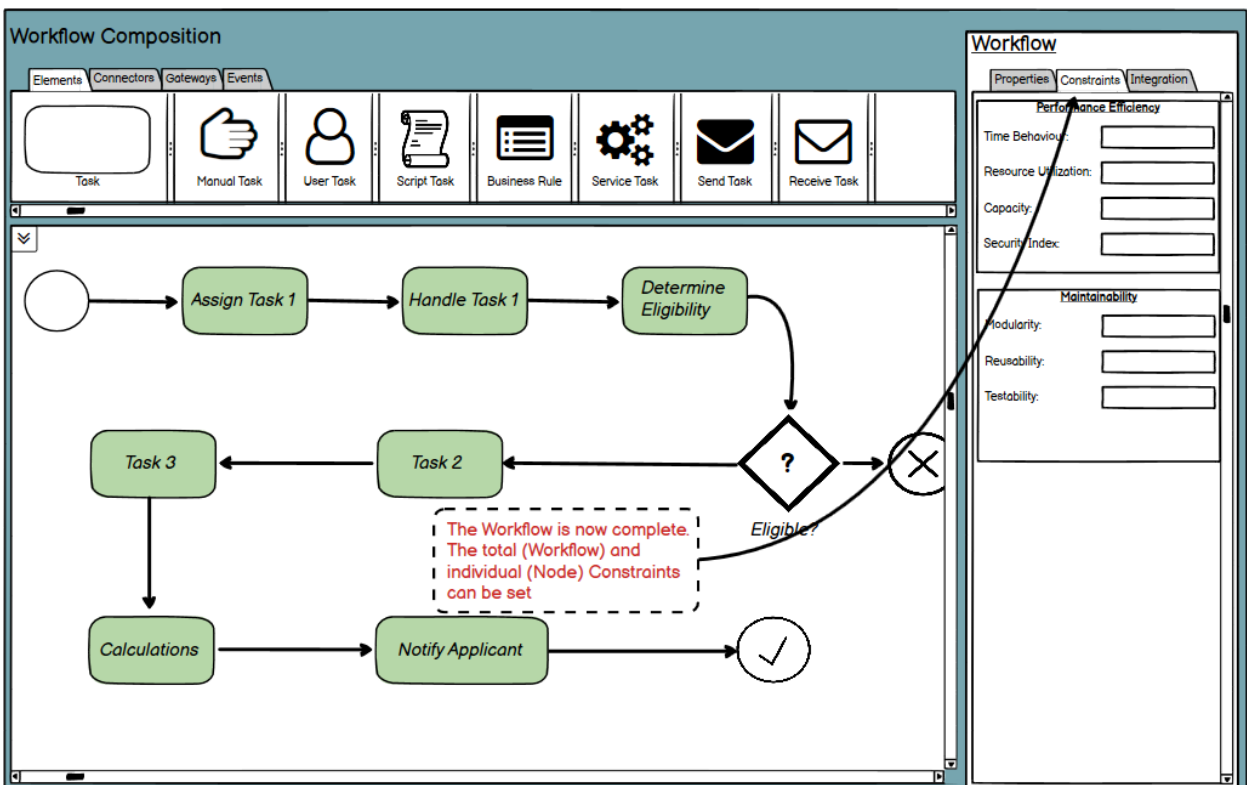


Figure 26: Workflow Completed

In Figure 26, we can see a screenshot of a complete workflow in which every node has been successfully matched with a service, either an existing, newly created, or an In-house one. On the Option Menu, the Workflow Constraints are presented. These constraints are the same with the Task Constraints. Here, we specify the non-functional requirements of the Workflow, or of an individual Task. The non-functional requirements purpose is to evaluate whether our process meets our performance guidelines / requirements. These fields are not crucial to the completion and execution of the process, so it's not mandatory they be filled in. The last step before the deployment of the process is the Test and Integration phase. This is achieved through the Integration and Testing menu, which is the third tab of the Workflow Options as shown in Figure 27. For evaluation purposes, the system provides some tests. Tests that can be performed are unit tests, from the local project folder, and SOA tests, that can be local or web based. After the Testing phase, several metrics are displayed as a result for us to evaluate. If we are satisfied with the overall state of the process, we can move to the deployment phase.

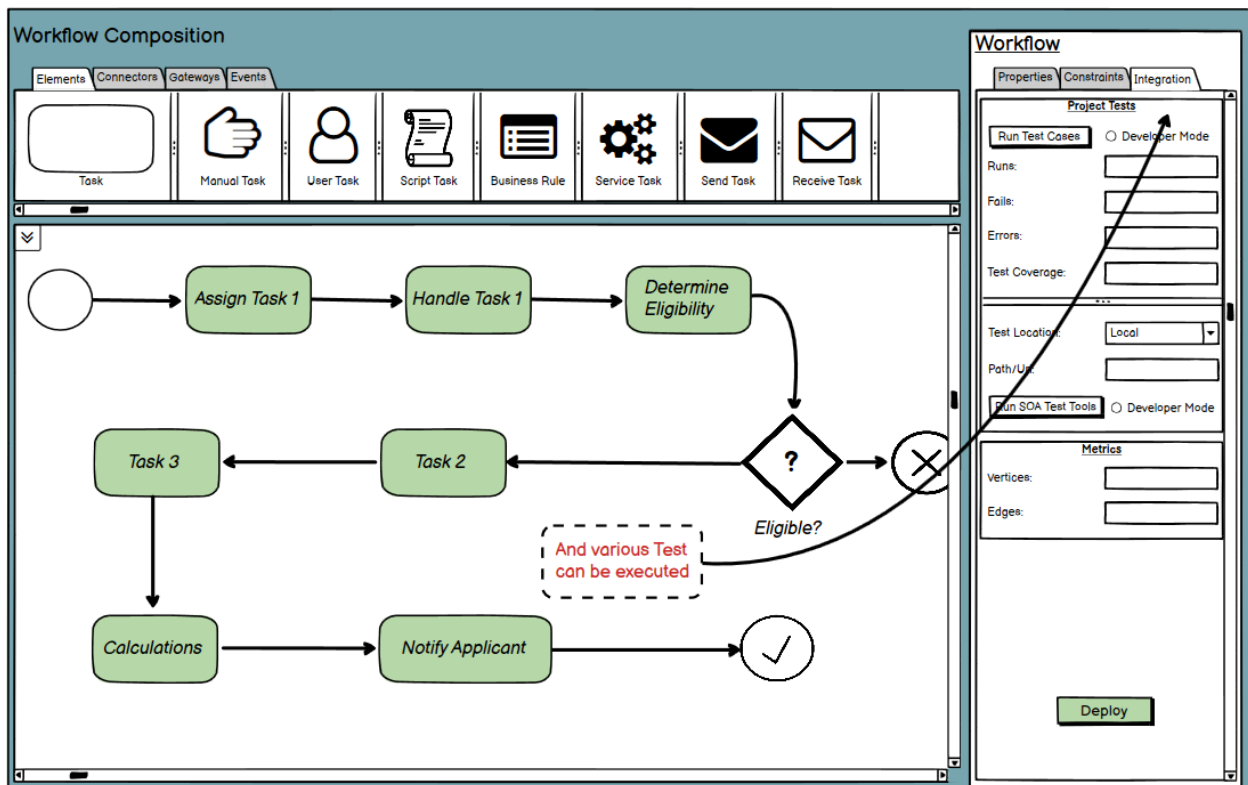


Figure 27: Test and Integration

4.3 Security

In this section, we specify our approach for delivering the *Security* component requirements, as defined in D1.2 *Requirements Analysis*. Similarly to section 4.2, we need to stress out that this document aims to introduce the main concepts of the SmartCLIDE project. Besides, we would like to note that we use indicative mock-up screens instead of a formal modelling approach to represent Minimum Viable Product. The formalized overall system architecture will be specified in the D1.5 *SmartCLIDE Architecture*.

The “*Security*” component could be specified as a subcomponent of the “*Service Creation, Composition and Testing*” component that was thoroughly described in Section 4.2, and is responsible for providing information for the *Security Constraint* tab of the implemented software (i.e., defined workflow). Taking into account the requirements specified in D1.2, the “*Security*” component can be decomposed into three sub-components, which are interacting with additional internal components and external tools, libraries, and frameworks. A high-level overview of the “*Security*” component (that corresponds to the first level of decomposition) is illustrated in Figure 28. In this figure, similarly to Section 4.2, the sub-components are presented as green boxes, whereas external tools, libraries, or frameworks are represented using relevant icons. The continuous arrows denote dependency, whereas the dashed arrows denote potential (i.e., optional relationship) that needs further investigation. As already mentioned, the high-level overview presented in Figure 28 does not necessarily correspond to the actual architecture of the component. It is used as a means to describe better the underlying concepts and the envisaged functionalities of the “*Security*” component in a more comprehensive and easy-to-understand way.

Security-related Static Analysis: The *Security-related Static Analysis* subcomponent will be responsible for handling all the requirements related to the identification of potentially critical security issues (i.e., potential vulnerabilities) that reside in the source code of the produced software. More specifically, this subcomponent will retrieve as input the source code of the different tasks that will be defined in the *Workflow Manager*. This source code will be retrieved from: (i) the code repository, in case a reusable code template/snippet is used, (ii) the service repository, in case that the task corresponds to the invocation of a service, or (iii) the actual code that is written by the developer, in case that no reusable service or template is available and the developer needs to create this task from scratch. The subcomponent will then invoke the execution of popular static code analysers (either open-source or commercial) in order to detect security issues that reside in the different tasks that constitute the overall workflow.

Vulnerability Assessment: The *Vulnerability Assessment* subcomponent is responsible for assessing the security level of the different tasks that constitute the overall workflow, as well as of the workflow (and, in turn, the actual software) itself. In order to achieve this, this subcomponent will retrieve the source code of the different tasks and apply a set of carefully constructed Vulnerability Prediction Models. Similarly to the *Security-related Static Analysis* subcomponent, the source code will be retrieved either from the code template repository, or from the service

repository. It can also retrieve the code directly from the user, in case that the task is created from scratch. The Vulnerability Prediction Models will be based mainly on text mining and Deep Learning, using popular deep learning libraries like Tensorflow¹⁸ and Keras¹⁹. The utilisation of the static analysis results (i.e., produced by the *Security-related Static Analysis* component) will be also considered, in order to examine whether they would enhance the accuracy of the models.

Report Generation: The *Report Generation* component will be responsible for aggregating the results produced by the *Security-related Static Analysis* and the *Vulnerability Assessment* components for facilitating their further processing and comprehension. More specifically, the raw results produced by the *Security-related Static Analysis* component will be aggregated and presented to the user in an intuitive way through Visual Analytics constructs (e.g., charts, graphs, etc.). For this purpose, well-known visual analytics libraries will be utilised. In addition to this, the reports of the *Vulnerability Assessment* component will be presented to the user in an intuitive way, in order to facilitate the identification of security hot spots (see Figure 30). Finally, both the static analysis and the vulnerability assessment reports will be available in a machine-readable form (e.g., JSON) in order to facilitate further processing by other components of the SmartCLIDE framework (if necessary).

¹⁸ <https://www.tensorflow.org/>

¹⁹ <https://keras.io/>

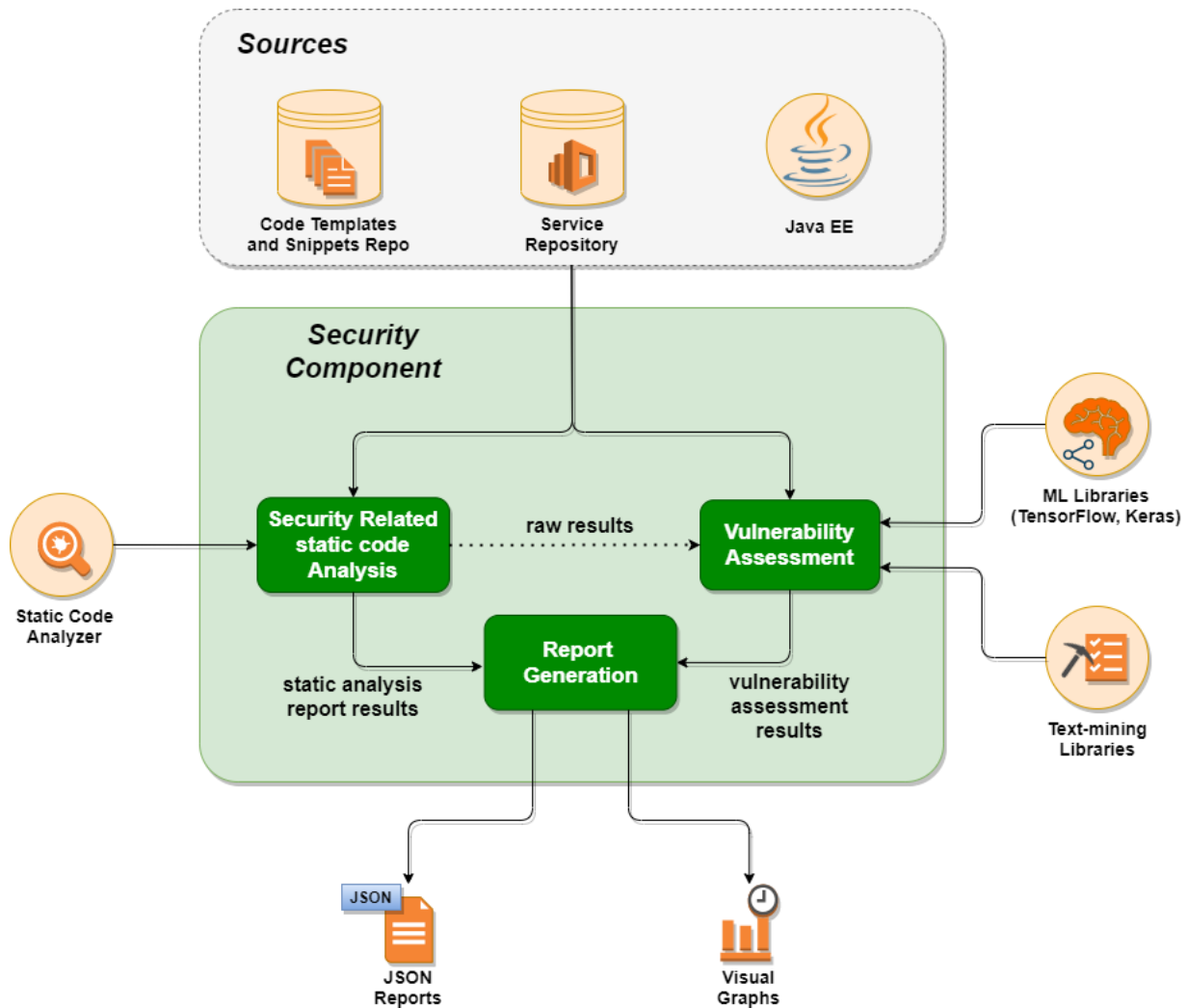


Figure 28: High-level overview of the Security component

4.3.1 TRL 4 Lab Validations (Minimum Viable Product)

In this section, the initial mock-ups of the SmartCLIDE platform that are related to the Security component are presented. As already described in Section 4.2.2, SmartCLIDE would be a cloud platform in which the user will have access by a web browser. We have created a set of mock-ups demonstrating how the functionalities of the *Security* component should actually be integrated into the broader SmartCLIDE platform. The main entry point of the functionalities provided by the *Security* component is the *Constraints* tab of the *Workflow Composition* tool (see Section 4.2.2). As already mentioned, in the *Constraints* tab, important constraints like Maintainability are provided, which will be determined through Maintainability analysis. In a similar manner, a dedicated place for the Security of the defined process (i.e., workflow) should be also provided, as it is an important constraint of the overall workflow. The updated *Workflow Composition* tool having the Security Menu under the *Constraints* tab is illustrated in Figure 29.

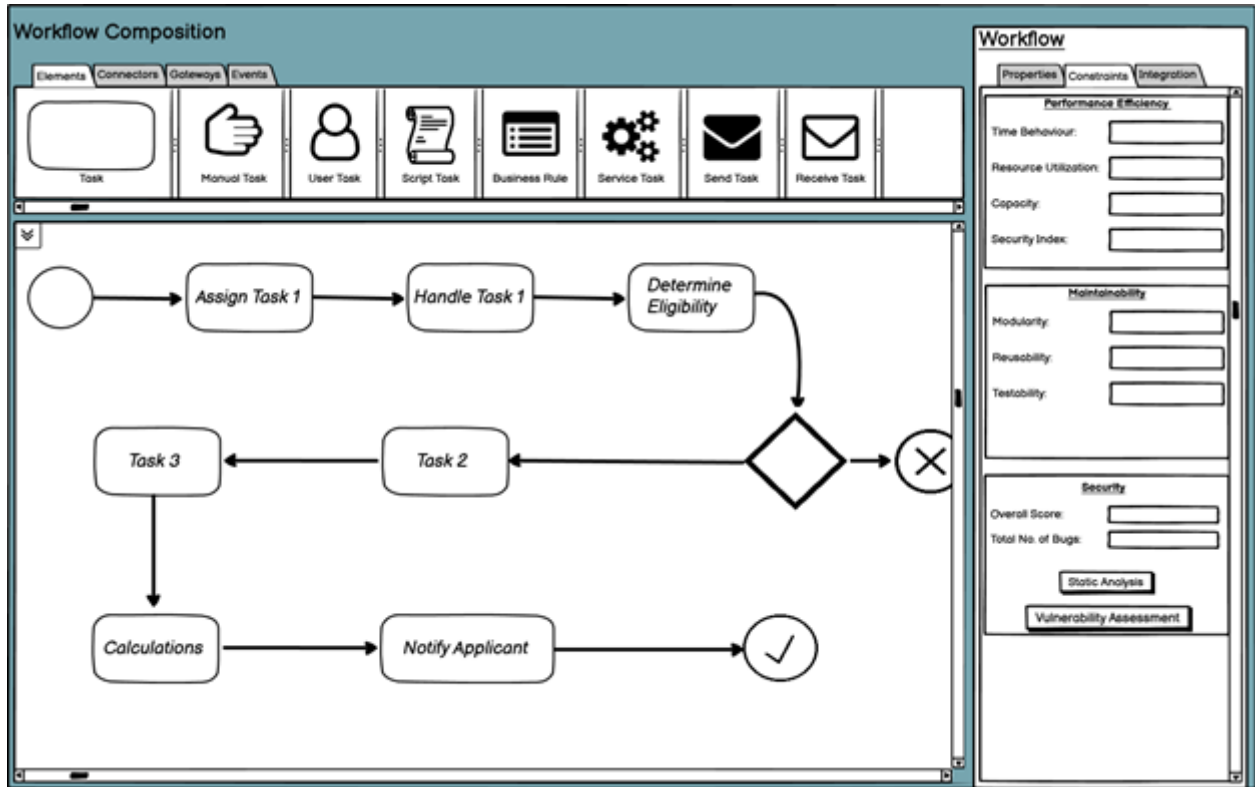


Figure 29: The Workflow Composition tool with the Security menu under the Constraints tab

As can be seen in Figure 29, when the developer has finalised the definition of the workflow (i.e., process) they can navigate in the Constraints tab on the right part of the screen, in order to get information about important non-functional requirements. At the bottom right corner of the screen, there is a dedicated Menu for Security analysis. In brief, the developer can see some high-level security-related information produced by the Security component, which are: (i) the Overall Vulnerability Score (i.e., the vulnerability score of the whole workflow after aggregating the scores of the tasks from which it is built), and (ii) the Total Number of Security Bugs (i.e., the total number of security issues that were detected through static analysis). Apart from this information, the user is equipped with two options: (i) Static Analysis, and (ii) Vulnerability Assessment. By selecting one of these options, the user triggers the corresponding component (described in Section 4.3), the analysis is performed, and the screen is updated in order to inspect the produced results.

If the user clicks on the Static Analysis button the *Security-related Static Analysis* subcomponent (see Figure 30) is invoked. The different tasks are retrieved and it is analysed using the static analysis tools with the proper configuration. When the analysis is complete, the screen is rendered and the results are demonstrated to the user (see Figure 30).

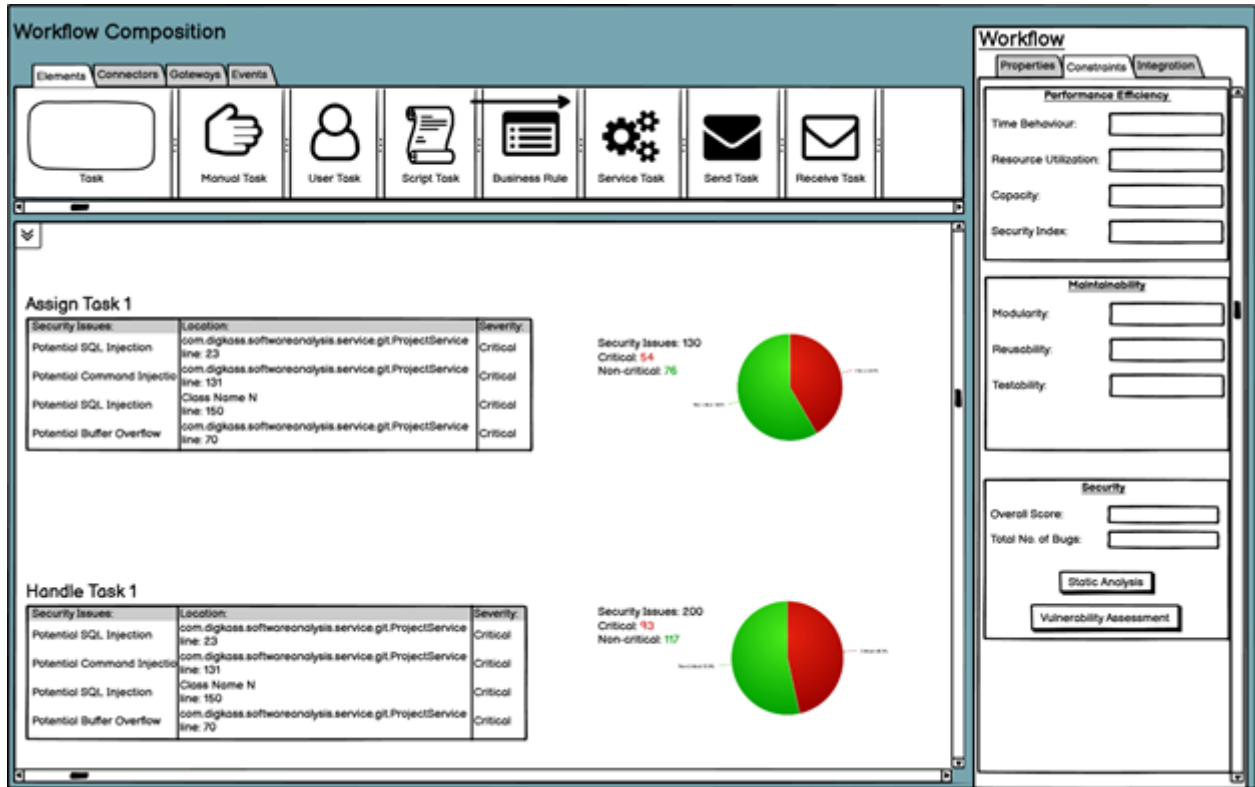


Figure 30: The results of the Security-related Static Analysis component

As can be seen by Figure 30, the static analysis results are presented in an intuitive way to the user. First of all, they are grouped based on the task to which they belong, allowing the user to focus separately on the different tasks, which is very useful for judging the security of the different reusable components. A detailed table is provided presenting the identified security-related issues (i.e., potential vulnerabilities). Important information is displayed for each identified issue, including its type (e.g., Buffer Overflow issue, Injection issue, etc.), its exact location in the code, and its severity. Additional useful graphs will be provided illustrating high-level information, by aggregating the raw results produced by static analysis. For instance, as can be seen by Figure 30, charts can be provided showing the percentage of critical issues that each task contains. This could be used by the user in order to decide whether the option to use any of these tasks may be risky and needs to be reconsidered.

If the user clicks on the Vulnerability Assessment button in Figure 29, the Vulnerability Assessment subcomponent (see Figure 28) is invoked. The different tasks are retrieved and analysed using the dedicated deep learning-based vulnerability prediction models, in order to compute their vulnerability score, i.e., their likelihood to contain a vulnerability. When the analysis is complete the screen is rendered in order to present the results in an intuitive way, as show in Figure 31.

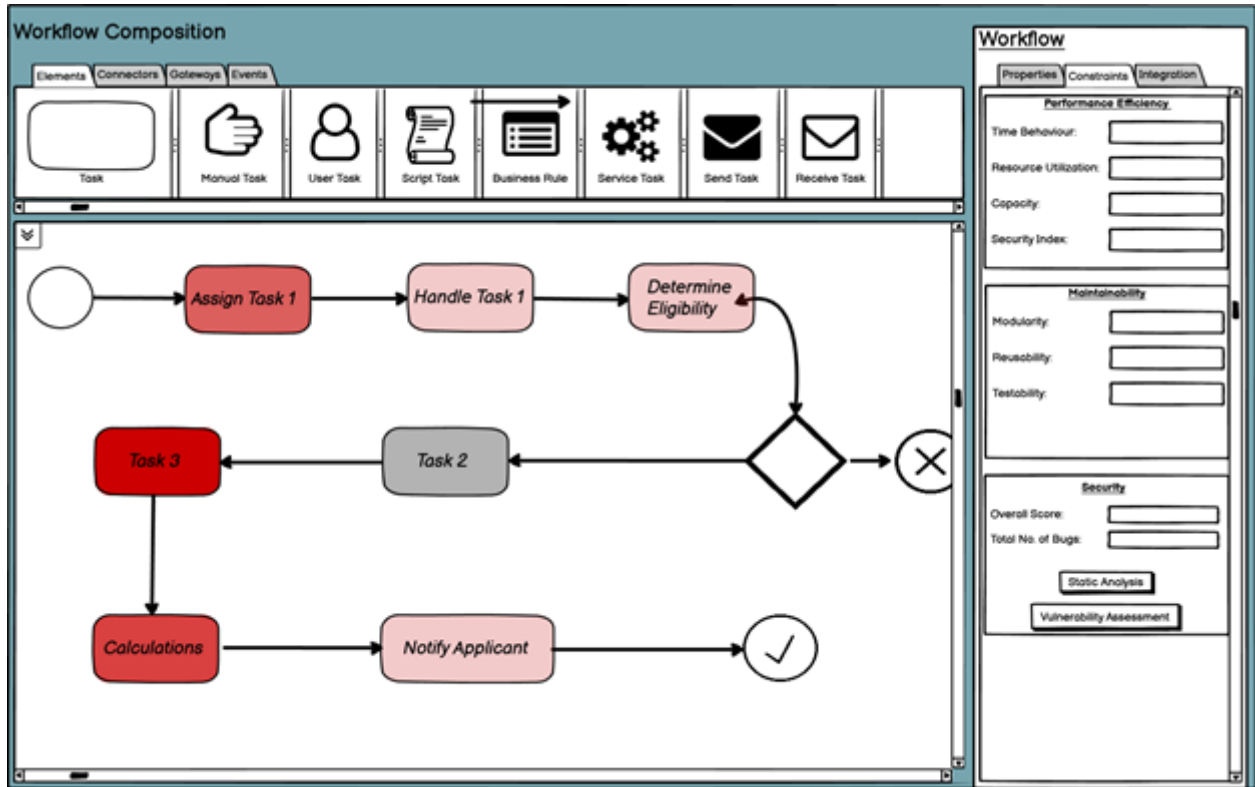


Figure 31: The workflow updated with the results of Vulnerability Assessment

As can be seen by Figure 31, the results are presented by changing the colours of the developed workflow. Different shades of red are used to denote how likely it is for a specific task to contain a vulnerability. The redder the colour, the more likely it is for the corresponding task to be vulnerable. Hence, this view is very useful for the developers as it allows them to pinpoint security hotspots, i.e., potentially vulnerable tasks. Based on this information, the developers can revisit their decisions. For instance, in the given example Task 3 was found to be very likely to contain a vulnerability. The developers can then decide either to use another service that is more secure (based on our analysis), or write a service from scratch. As a third option, the developers could revisit the overall workflow, and come up with an alternative one that does not require this task.

Finally, the greyed-out boxes in Figure 31 denote tasks that they were not analysed, because their source code was not accessible (for example, Task 2 in Figure 31). The user will be provided with the option to manually supply the code for this task and repeat the analysis for this task. However, if the task is based on a commercial (i.e., closed-source) service, the analysis cannot be executed. The user needs to be informed in this case, in order to make the best decision. The developer can either accept the service, because it may be created by a trusted third-party organization with available security certificates (hence the risk of not analysing it is low), or find a suitable service for a given task (similarly to the case described in Section 4.2.2 concerning the failure of the Service Discovery component). The user can either decide to implement this task from scratch (to better control its code, and, in turn, its

security) or re-design the overall workflow using an alternative that does not need this task.

4.4 Runtime Monitoring and Verification

SmartCLIDE provides the capability for non-programmers to construct applications and new services using smart automation. Quality assurance of the constructed applications involves both construction-time and run-time assurance. Assurance of the correctness of the construction of the application is addressed by the manner of construction. Assurance of the runtime behaviour is addressed by validating that the application exhibits behaviour consistent with the user's specification, and that the assumptions made about the runtime environment, made at the time of the design of the construction approach, and thus built into the construction of the application, continue to be valid as the application executes.

The intended behaviour of the application may involve both functional and non-functional properties. Key characteristics of the application, such as security, safety, privacy, resilience and reliability are general categories of runtime quality attributes that may be required.

Validity of environmental assumptions involves necessary conditions under which the construction and execution of the application are expected to be correct. The assumptions made by individual components or services from which an application is composed, and the consequent assumptions of their composition, need to be established and maintained during the execution of the application.

These concerns occur in statically constructed applications as well as those that are dynamically constructed and executed on demand. However, in the dynamic case some runtime quality attributes arise that do not exist in the static case. For example, in the dynamic case the identification of the components or services to be composed, the correctness of the service composition, and testing of the composition arise during construction runtime rather than during a conventional design, implementation, test and deployment cycle.

When an application is dynamically composed for a specific purpose, known properties of the component services can be used to find composition solutions that yield a final property that suits the purpose. However, because the dynamic case does not afford the same opportunity to apply the conventional disciplines for assurance during design and implementation, or for conventional testing, it is prudent to supplement the measures that can be done during construction of dynamically composed applications with automated runtime monitoring to provide ongoing collection of evidence supporting the claim of correctness of the application. The defining property of the application as well as the assumptions should continue to hold during runtime.

The Runtime Monitoring and Verification (RMV) framework is intended to provide the capability for constructed applications to be automatically monitored at runtime for their validity of their properties and their environmental assumptions. If the SmartCLIDE services creation, services discovery, services composition, and services deployment functionalities are thought of as "programming for non-programmers" then the runtime monitoring and verification features provide integrated "runtime quality assurance for non-programmers".

It is not yet fully known the extent to which this objective can be completely automated. It depends to a great extent upon the manner in which the desired application functionality is specified, the detail of the behavioural specification of the microservices used to compose service-oriented applications, and the methods for composing the application. If the service composition reasoning and the service composition details are available externally, and if the properties and assumptions of the component services are sufficiently formal, then properties and assumptions of the composition can be inferred and corresponding monitors can be generated. In this case a sufficient library of monitoring components would be created and their composition automated in concert with composition of the application to create the corresponding monitoring application which can be deployed with and run beside with the application.

Though there is some technical risk of falling short of the most ambitious objectives due to these uncertainties, the resulting monitoring framework will nonetheless provide several useful benefits:

- The Runtime monitoring and verification components are used by the Context Handling system to subscribe to get monitored data about runtime values and events that it needs to accomplish its purpose.
- Developers using the service composition capabilities of SmartCLIDE will be able to develop monitoring applications using the monitoring and sensor services that can detect sequences of events specified by patterns specified by the monitoring application. These may be used to detect exceptional conditions or may be integrated into the design of the application to enable a response to monitored conditions external to the application.
- The security component can construct a monitoring application to monitor security relevant events that other application services are designed to generate. These may be stored in the Log as an audit trail for post-execution forensic investigations.
- Other subsystems can construct monitoring applications to assist with instrumentation, testing, or debugging of applications or the SmartCLIDE platform itself.

The RMV provides a flexible and configurable framework (see Figure 32) with programmable interfaces to permit the construction of a monitoring application, using standard or bespoke monitoring components and sensors, in parallel with the construction of a service-oriented application. The framework will also provide for the capture and retention of data gathered by sensors and monitoring applications in a log according to configurable criteria.

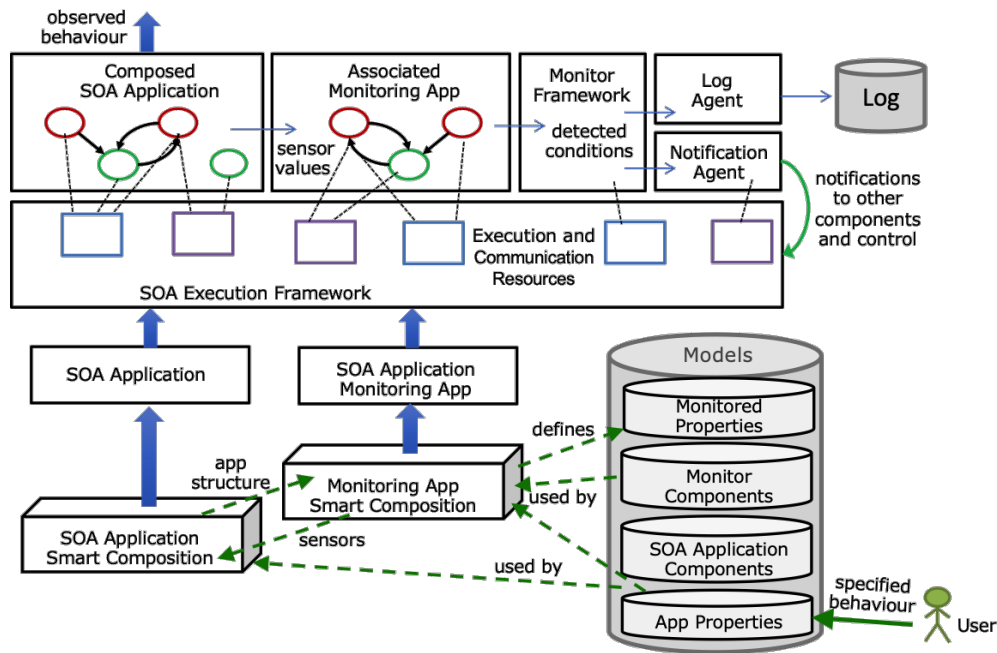


Figure 32: Application and Monitor Creation and Deployment

The technical elements of the approach include:

- A library of monitoring logic components for constructing monitoring applications.
- A library of sensor types that can be instantiated and installed within composed SOA applications and infrastructure components to gather the data/events needed by monitoring applications.
- A mechanism for composing monitoring applications in parallel with the SOA application.
- An SOA execution framework for executing SOA applications, monitoring applications, the monitor framework and the agents.
- A notification mechanism that can transmit alerts to registered participants when monitored events meet a specified condition.
- A method and mechanism for selecting monitoring components and sensors and constructing appropriate monitoring applications according to the construction of a corresponding SOA application.
- A mechanism for the collection and persistent storage of the Log of monitored data and notifications.
- A mechanism for the configuration of log collection and storage, including where the Log is to be stored, how large logs should be archived, etc.

The ambitious objective of automated determination of conditions to be monitored, and automated construction of corresponding monitors, will attempt to leverage the automated application programming infrastructure for automated monitor construction.

The starting point for defining the concept of the “*Runtime Monitoring and Verification*” component is the requirements analysis document:

SmartCLIDE “Runtime Monitoring and Verification” component shall be able to support:

- A flexible and configurable framework for the construction and deployment of diverse monitoring applications.
- Programmable responses according to the findings of the monitoring applications, including notifications to the application execution framework and activation of predefined responses.
- Programmer-directed explicit construction of customised monitoring applications.
- Cooperative monitoring, that is, passive monitors that are called by the application and/or the execution framework to report various predefined conditions.

SmartCLIDE “Runtime Monitoring and Verification” component should be able to support:

- Capture and storage and/or forwarding, according to configurable filters, of monitoring data to other components.

SmartCLIDE “Runtime Monitoring and Verification” component may be able to:

- Leverage the SmartCLIDE application service composition to automatically construct monitoring applications to run in tandem with the composed applications and to detect in the applications’ behaviour violations of the applications’ specifications.

4.4.1 TRL 4 Lab Validations (Minimum Viable Product)

The Runtime Monitoring and Verification (RMV) component comprises at a minimum:

- a library of monitoring primitives and virtual sensors
- a runtime monitoring framework that arranges for communication among monitoring applications and actions/logging/notifications that are triggered by conditions detected by a monitoring application
- a log agent that collects events and log data from all monitored applications and directs it to a log according to directives in a configuration file
- a notification agent that upon certain detected conditions notifies the SOA execution framework of exceptional conditions, and may notify other processes that have subscribed to select notifications

The RMV component provides APIs to its services including:

- reporting of events by SOA applications or monitoring applications
- an event pattern matching mechanism (potentially spanning a range of pattern classes, from simple matching of event properties to matching of event sequences to temporal logic formulas)
- logging to a runtime log which is backed to persistent storage
- management of persistent logs
- subscription to notification of specified events or event patterns collected from monitoring applications
- subscription to forwarding of specified kinds of monitored data

The Context Handling component is expected to use the RMV Framework to subscribe to monitored data and notifications to gather the runtime information it needs for its purpose of constructing a model of the current context, or to construct specialized monitoring applications, as described below, to work in conjunction with special applications.

The RMV may also provide:

- a library of monitor components and virtual sensors
- a monitoring application smart composition procedure, operating in concert with the SOA application smart composition process to construct a monitor for the essential properties, representing the correct operation or necessary conditions for the correct operation of the SOA application

The RMV component does not have an explicit user interface, except in the special case of the programmer-directed explicit construction of customised monitoring applications, or as an explicit component of an application. In this special case, the UI to RMV is the very same interface provided for assisted programming, where in this case the application being constructed is a monitoring application.

When the RMV is employed as a runtime quality assurance measure on applications/services composed from other available, and adequately specified, components/services it operates silently behind the scenes unless the composed application deviates from its expected behaviour. The monitoring application, constructed from a library of component monitoring services and sensors by the assisted service composition component, operating in concert with the monitoring framework, is a detector for violations in the event trace of the application for which it is constructed to operate in tandem by the assisted programming service. In this case, the monitoring framework notifies the application execution framework of the violation so that it may take appropriate action. Such actions may include termination or restart of the errant application as defined in advance by the assisted service composition component.

When the smart SOA application composition component creates an application, it is presumed to receive or construct a specification of the needed behaviour of the application to be composed and to use specifications of available primitive services to construct the application. The a priori component properties and their environmental assumptions, as well as the expected properties of the composed application, provide opportunities for runtime monitoring to assure that assumptions continue to hold and the application continues to deliver expected behaviour.

The RMV requires input from the service composition process to determine what properties of the SOA application being constructed should be monitored at runtime and what values/events in the application represent the information needed to verify the properties. The RMV in response provides “virtual sensors” for identified application events for installation into the composed application. These sensors comprise calls into the Monitoring Application to register the occurrence of related events.

The RMV requires input from the running SOA application as events. These events are consumed by the Monitoring Application specifically constructed for the specific SOA application. The event definitions are selected to enable detection of the properties of the application to be verified at runtime. There may be properties that are common to multiple applications. That is, some properties may be instantiated from a library of properties.

4.5 Run-time Simulation & Monitoring / Visualisation

The “*Run-time Simulation & Monitoring / Visualization*” component will provide a front-end for the monitoring solutions in SmartCLIDE. This interface will allow the developer to visualize the status of deployed services through visual and text-based elements inside a Run-time Monitoring Console, which may be integrated inside the SmartCLIDE UI. To do so, the “*Run-time Simulation & Monitoring / Visualization*” component will exploit the “*Activity Monitoring*” module inside the “Backend Services” component, which will allow the console to extract information about running container and services, and to connect to them.

The following diagram (see Figure 33) represents a use case scenario of the “*Run-time Simulation & Monitoring / Visualization*” component, where the developer wants to monitor a certain container:

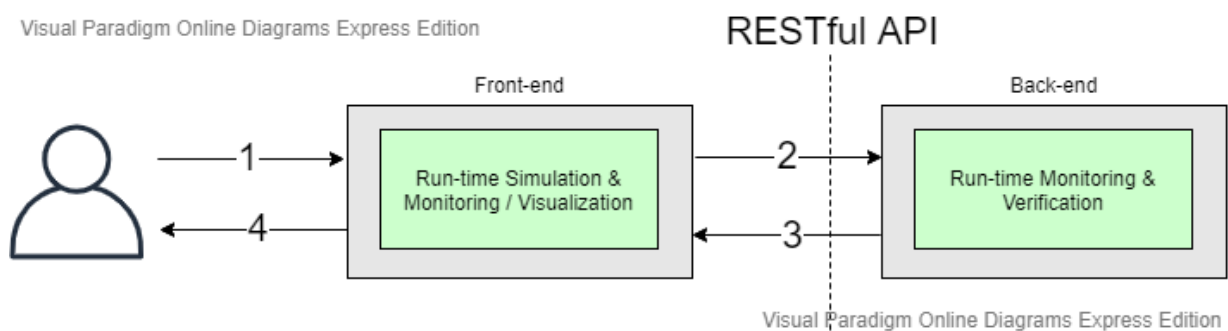


Figure 33: Use case scenario for monitoring a certain container

In the previous diagram, the steps are:

1. The developer selects the container he wants to monitor by using the “Run-time Monitoring and Simulation / Visualization” module in the SmartCLIDE UI (i.e., see mocks up in Section 4.5.1).
2. The “Run-time Monitoring and Simulation / Visualization” component subscribes to the “Run-time Monitoring & Verification” component by using a RESTful API.
3. The “Run-time Monitoring & Verification” component reports monitoring data from that container.
4. The “Run-time Monitoring and Simulation / Visualization” component represents received run-time monitoring data using both, visual and text-based techniques.

4.5.1 TRL 4 Lab Validations (Minimum Viable Product)

After analysing the state of the art and initial requirements, we present the initial mock ups of the ‘Run-time Simulation & Monitoring / Visualization’ component. Its main element will be the monitoring console, which will allow the developer to monitor and track the state of services deployed in SmartCLIDE. To do so, its core menu will present three different horizontal tabs: overview, terminal and settings.

The **Overview** tab (see Figure 34) will allow the developer to visually see the status of some indicators, which will depend on the chosen target (i.e., selected container). The developer will be able to choose which service (e.g., container) to monitor through a vertical tab selector, further enabling the developer to rapidly switch between them. In the mock up below, we show how some performance indicators could be visually showed, but its content may depend on what backend services allow to track.

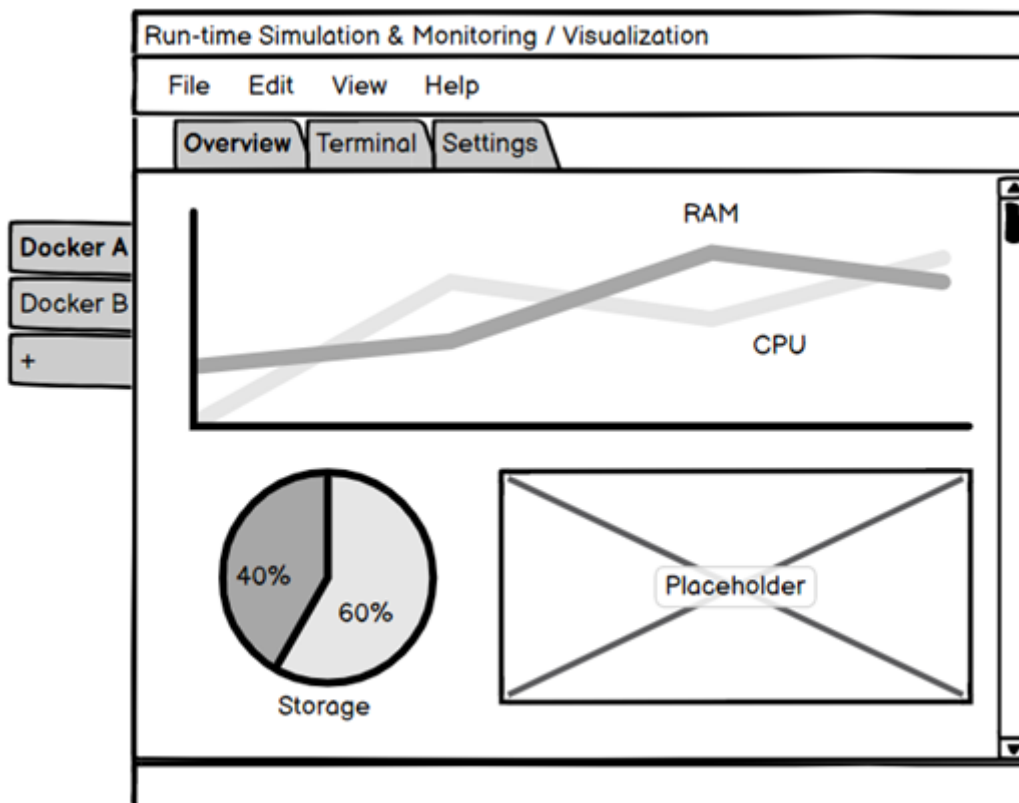


Figure 34: Run-time Simulation & Monitoring – Overview

The second tab is the **Terminal** itself (see Figure 35), and will allow the developer to start terminals to any application deployed in SmartCLIDE. This will be useful to watch logs, check deeper performance indicators through server-side CLI applications (e.g., top), or to make runtime changes in configuration files.

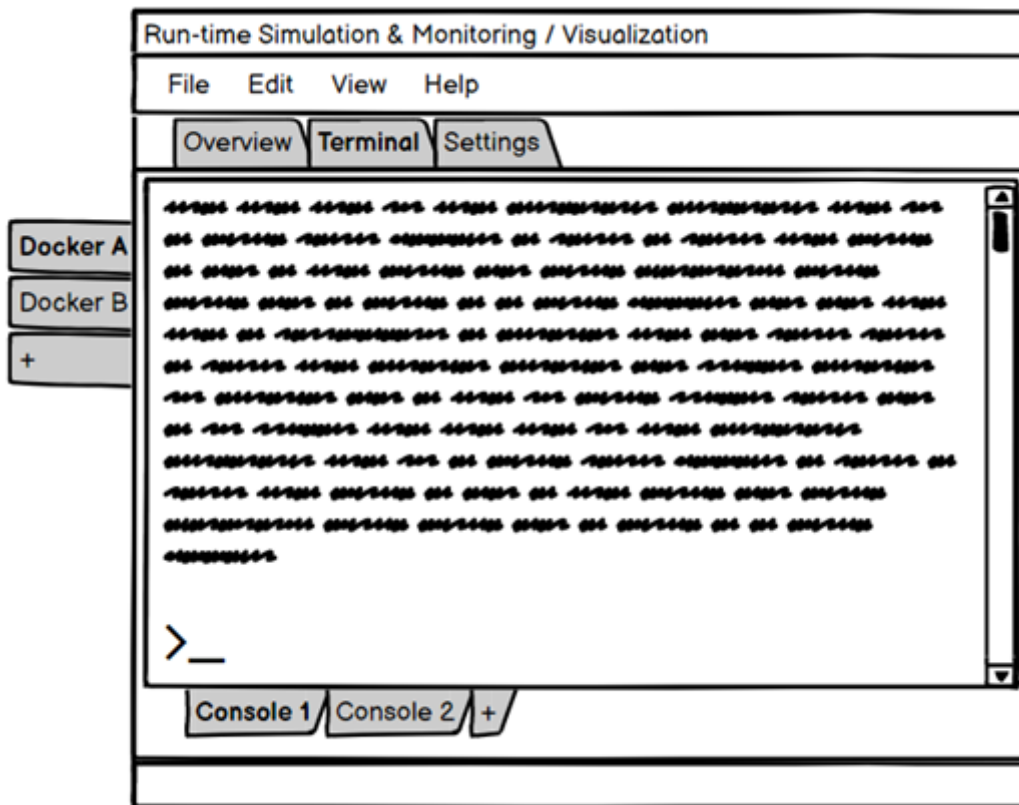


Figure 35: Run-time Simulation & Monitoring – Terminal

The last tab is the **Settings** one (see Figure 36), and will allow the developer to customize the settings of the console. The content in this tab will be explored during the execution of the project, and may contain both visual or functional options, such as customizing font style and size for the Command Line Interface.

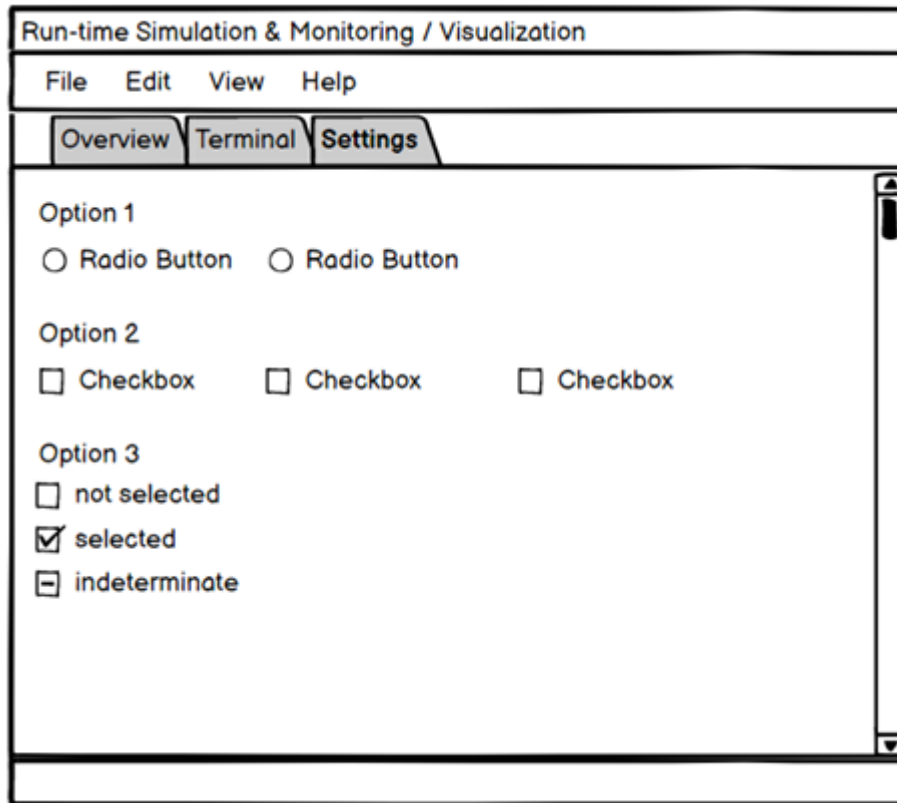


Figure 36: Run-time Simulation & Monitoring - Settings

4.6 Deep Learning Engine

In this section, the approach to the Deep Learning Engine (DLE) from a conceptual perspective is described. The DLE is the base for code classification, generation of code templates, services classification and abstraction management. Besides, it has to support predictive models' generation and presumably different Smart Assistant recommendation fields, such as code autocompletion and software lifecycle suggestions. Smart suggestions' utility will depend on the results shown by AI algorithms at enhancing the different support fields (e.g. development, testing, deployment, etc.) versus existing IDE plugins. Consequently, a single component which supports several AI algorithms/methodologies has to be built. At a later time, iterative approaches on each required feature will conform their first stable release at the early prototype. Considering the wide range of applications, the DLE service will potentially have to be split to maintain functional atomicity.

Different approaches will be taken according to the final destination of the predictions. The DLE will be connected to other components using a REST API, providing results in a flexible, homogeneous way. Further visualizations on the DLE data and models can be later on considered for implementation to deliver its outputs. A breakdown on the functionalities is as follows:

- Code templates generation. This functionality will explore the generation of quality code templates for atomic functionalities, by means of various AI techniques. The models will be trained with code selected for specific functionalities.
- BPMN flow suggestions. This functionality will support the suggestion of suitable nodes at the creation of BPMN flows, relying on predictive models trained with successful existent BPMN templates.
- Service classification. Connected with the Service Discovery Component, service classification functionality will retrieve the data gathered from pre-defined sources and put them into pre-trained models to extract classifications, based on the existent services' features. These features have to be determined based on the availability of existing ontologies or descriptors and the quality of the information at the required fields.
- Generation of predictive models. An assistant embedded into the IDE interface will provide the possibility of creating standalone containerised reusable models with an API, given a data source. This assistant will lead through data transformation to model attainment, pre-visualising the data, suggesting algorithms based on the data type, slicing the set automatically, and giving metrics to test the obtained model.
- Support to the Smart Assistant. Features provided by the Smart Assistant are prone to be supported by the DLE. It will be necessary to state if they will remain in the IDE interface via existing plugins, or if they will be fully supported by the DLE and subsequently called by the Smart Assistant. The Smart Assistant will deliver the results based on the development context. Suggestions on the next suitable component (BPMN) or code autocompletion will be supported by AI algorithms, while some other lifecycle recommendations (e.g. syntax, best practices, security) are still to be proven.

4.6.1 TRL 4 Lab Validations (Minimum Viable Product)

As mentioned, the DLE aims to support several AI operations. These essential functionalities will be used to conform an AI engine, and address the rest of the DLE attributions.

The easy generation of AI models will be the first experiment to develop. These models can be composed of known algorithms and techniques (e.g. neural networks, classifiers, regressors, etc.) and by methods designed and formulated specifically for each case. Once prepared, the efforts of the other experiments will focus on applying these models to the available data, given the demanded functionalities.

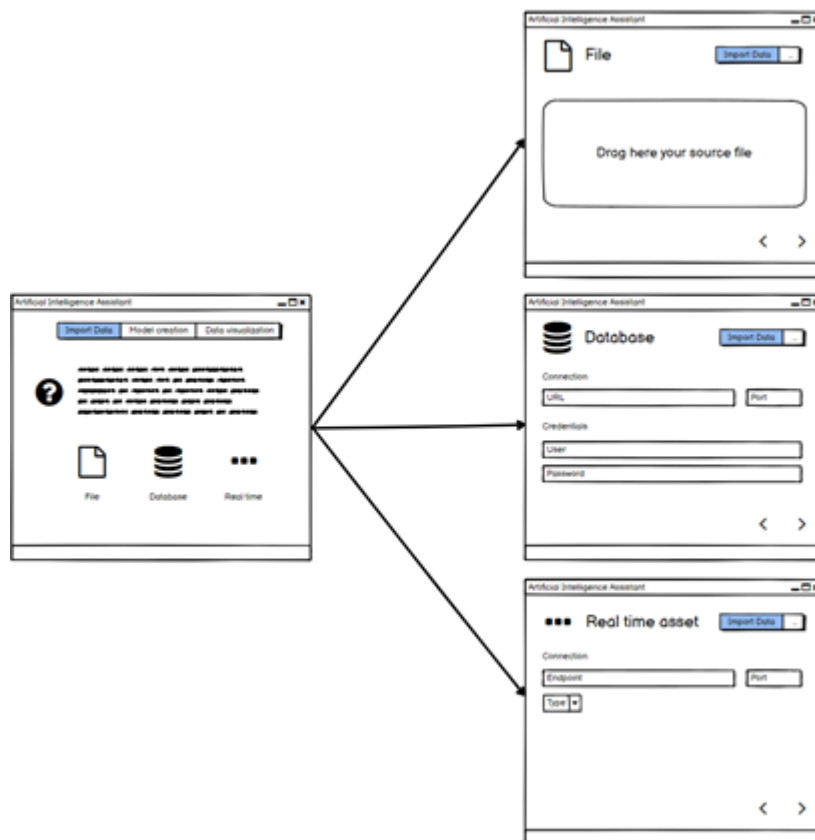


Figure 37: Deep Learning Engine - Overview

The system will bring in data from different sources, both from static files and real-time sources. It has to be able to accept data for every future functionality, from sensors to context abstractions. Once the data are retrieved, they will be pre-processed to detect the type and format of each field. Data will be pre-visualised to detect outliers, offer filtering and to create new computed fields.

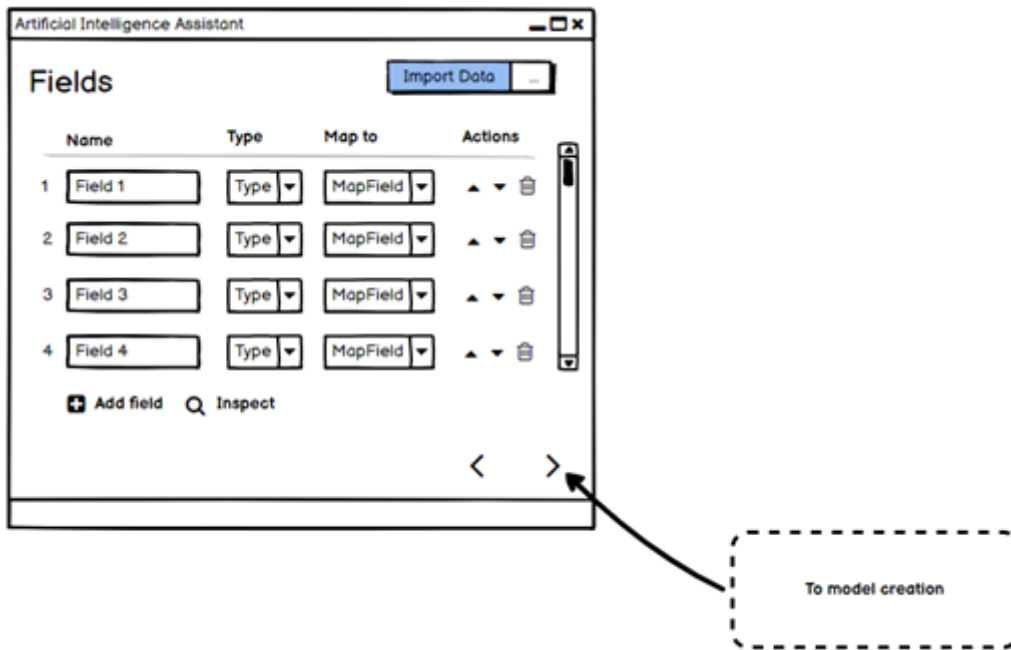


Figure 38: Deep Learning Engine - Import Data

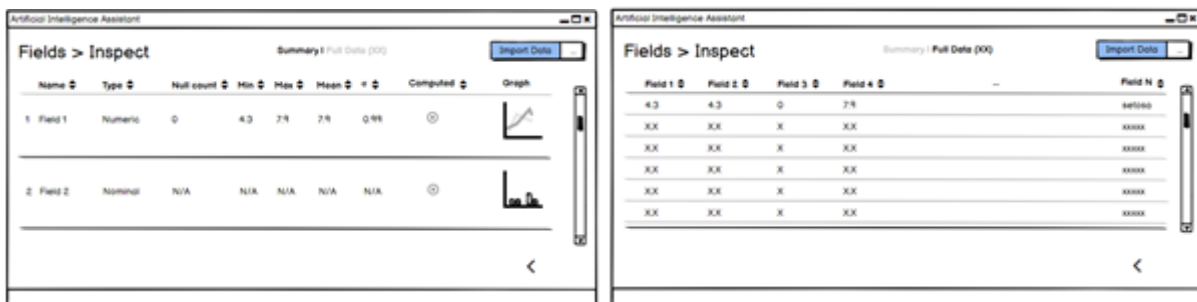


Figure 39: Deep Learning Engine - Import Data Details

Once the data is prepared in the form of a source, the target variable will be defined and the algorithm will be selected by the assistant. The dataset will have to be split into training and testing, where the default values will be manually controlled. This whole process will be either driven in a fast-forward way (automatic) or manually, constraining the generation parameters.



Figure 40: Deep Learning Engine - Model Creation

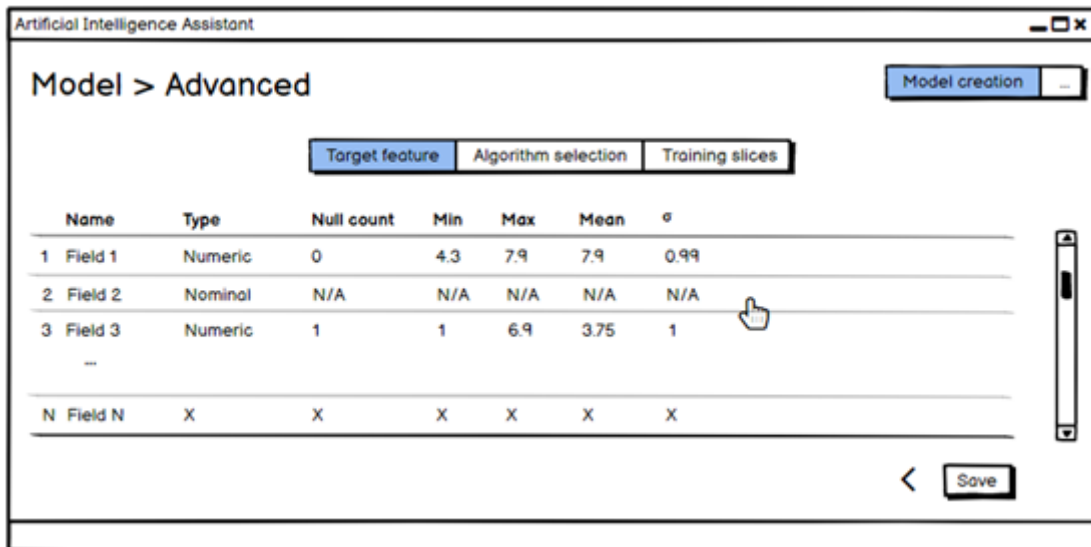


Figure 41: Deep Learning Engine - Model Creation - Target Features

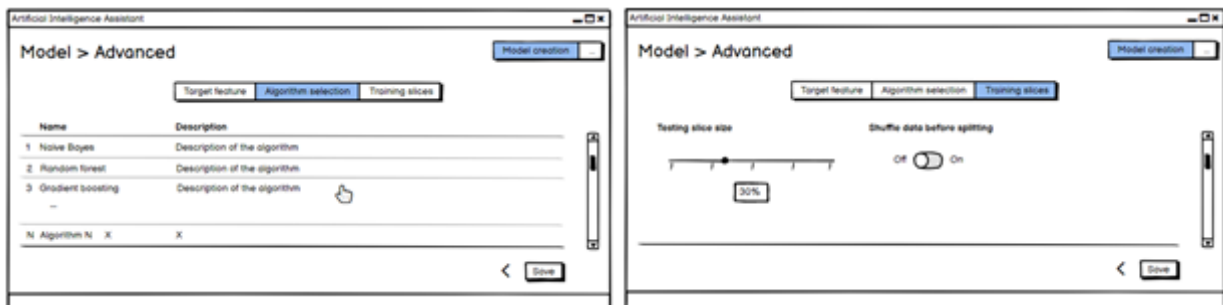


Figure 42: Deep Learning Engine - Model Creation - Algorithm and Training

After the model is trained, it will be stored and subsequently exposed in a small playground, to try it by choosing the input values. The user will be able to measure

the model performance by checking its metrics and review the work of the model over new sets of data.

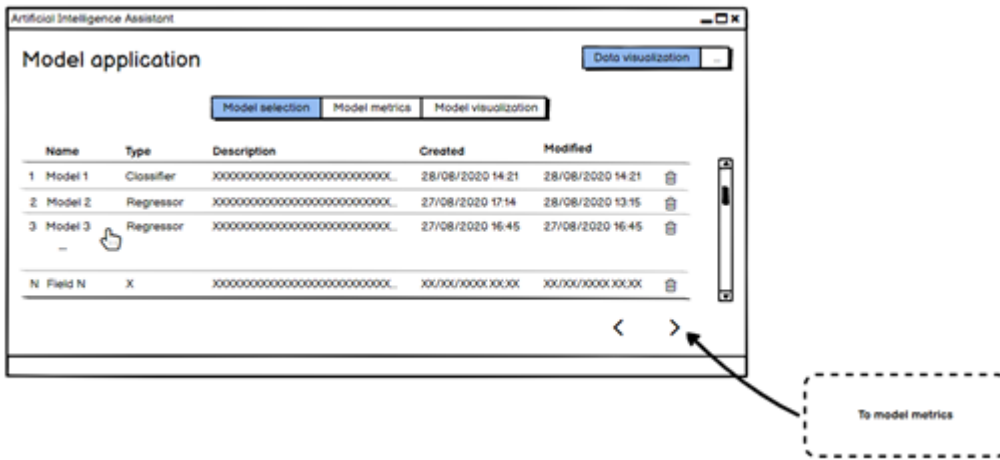


Figure 43: Deep Learning Engine - Data Visualization - Model selection

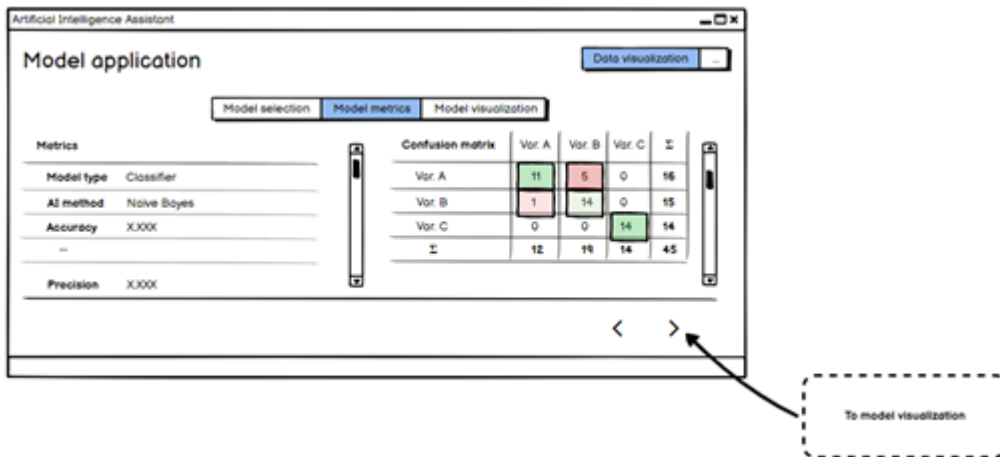


Figure 44: Deep Learning Engine - Data Visualization - Model metrics

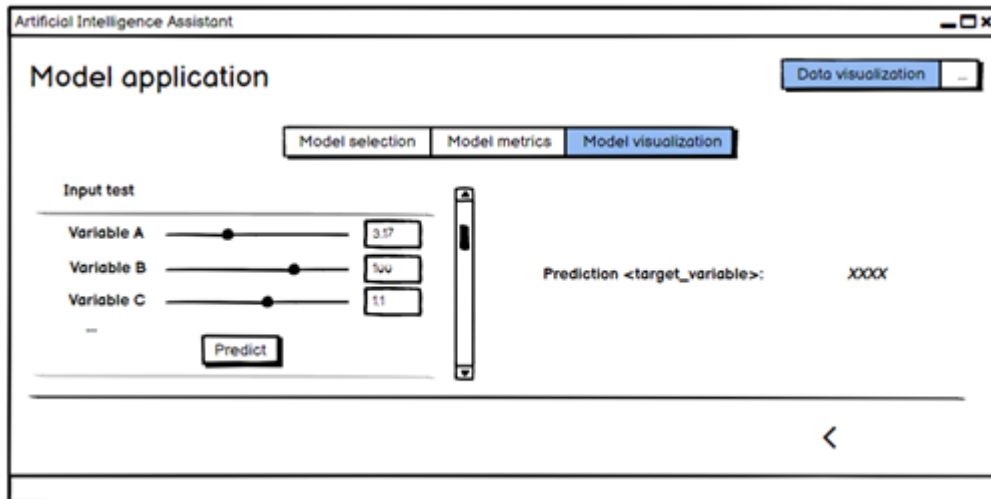


Figure 45: Deep Learning Engine - Data Visualization - Model visualization

Visualisation features can potentially be added to extract knowledge in a more efficient way by using chart libraries. As a final step, the models will be containerised with a REST API for their deployment.

Regarding other mentioned functionalities, some experiments can be performed on top of the formerly described AI backend.

- Code templates generation. Programming by Example paradigm projects, will be researched in terms of code generation approaches for different languages, preferably Java. The generation of code templates using AI methods, as well as other existing project conceptualisations will be explored. Once successful results are achieved in a reference language and depending on the used method, more languages will be added. Here it will be interesting to observe the usage of DSLs LSPs and GLSPs to provide easier language interactions and generalizations.
- BPMN flow suggestions. Previously generated BPMN flows will be analysed to extract common features and create a model to predict which elements are proved to be next to the others. This, combined with the Context Monitoring information, will allow the Smart Assistant to give accurate suggestions on what components are to be expected in a flow design based on the intentions of the user.
- Service classification. The combination of several AI techniques or semantic approaches will be used to extract clusters from existing data in bare services scraped from public web links. This will lead to an understanding of which techniques can be used to classify services depending on the final desired features.
- Support to the Smart Assistant. Generally speaking, communications with the Context Monitoring will have to be defined and settled and examine its outputs (abstractions) to determine which algorithms can be applied to predict user behaviours.

- Suggestions on the next feasible component (BPMN).
 - Code autocompletion. To this end, the strategy will be to examine existing autocompletion plugins and try to determine to what extent AI techniques are capable of improving their performance by cache managing or similar techniques.
 - Syntax highlighting. In this particular topic, linters will be explored to determine the most suitable options and how they can be enhanced or implemented within the IDE.
- Best practices, security. Combined with Context Monitoring analysis, some predefined rules related to software development and their application (correct timing and means of interaction with the user) can be explored to enhance the experience of the user.
 - Automatic tests generation based on Gherkin language. NLP techniques plus code generation via ML and DL algorithms will be explored along with automatic test generation tools to leverage their functioning.

Concerning context identification, a fluid and early connection between the Context Monitor and the DLE will leverage the learning from Context abstractions and their possibilities.

The scope of AI research achievements related to the functionalities above, and therefore their degree of adequacy, will be weighted on the go, as use cases specific requirements are prioritised (practical approach) and subsequently generalised as far as possible. The results will strongly depend on the training data and the suitability of the used methods.

For all the presented functionalities, integration with the Smart Assistant will be provided when required via REST API.

4.7 Context Handling

Within the SmartCLIDE project the Context Handling will be used as a baseline to gather and represent knowledge within the full-stack development lifecycle in the SmartCLIDE IDE. It will be based on a context model, which will be a set of concepts and their relations which describe the stages, entities, attributes and stakeholders within the collaboration of a full-stack DevOps development context. Figure 46 shows the conceptual architecture for the context handling services, which consist of the three services “Context Monitor”, “Context Extractor” and “Context Provider”. The conceptual approach for each of these services will be explained in the following sub-sections.

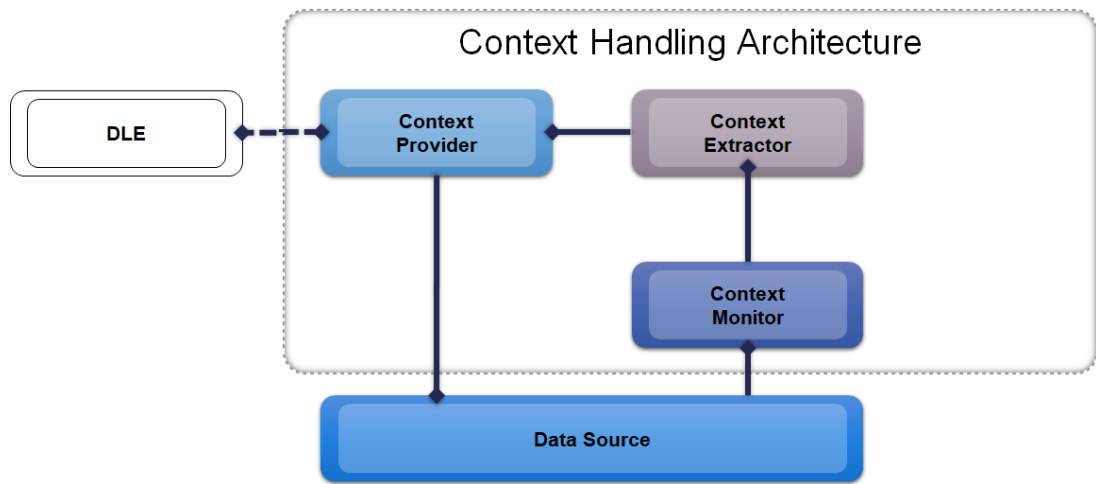


Figure 46: Conceptual Context Handling Architecture

4.7.1 Context Monitor

The objective of the Context Monitor service is to receive raw data and provide aggregated context data. It is a generic solution for monitoring data sources, which is customisable for different communication protocols and data structures. It enables data pre-processing or data aggregation. Figure 47 shows the conceptual architecture of the Context Monitor service.

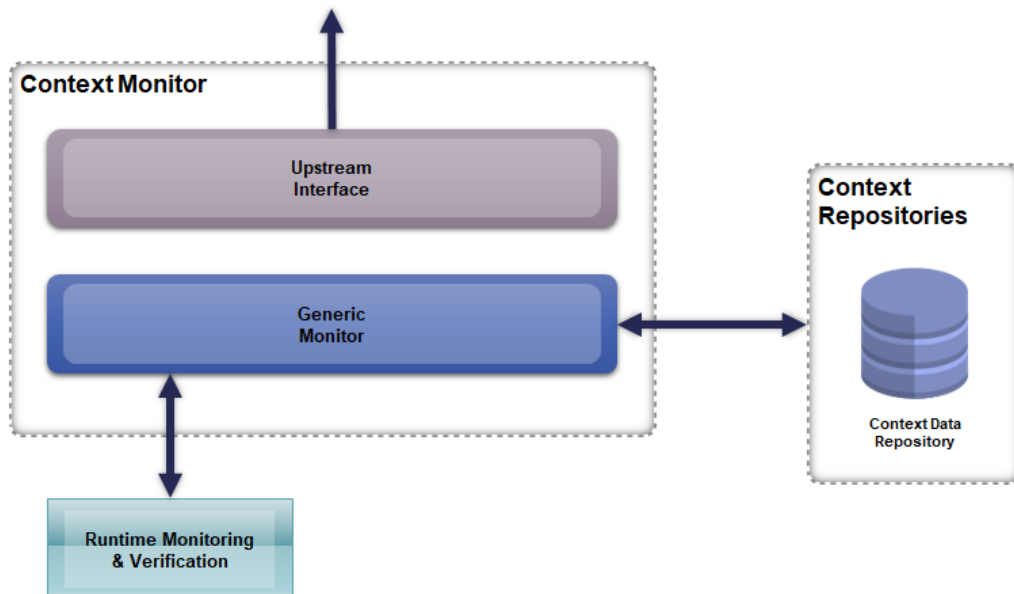


Figure 47: Conceptual Context Monitor Architecture

To achieve its tasks, the Context Monitor service utilizes monitoring of the Runtime Monitoring and Verification Service. It is therefore able to standardise and correlate data from distinct systems (e.g. map actions from file systems and Web/REST-Services), which later serve as a basis for identification and extraction of situations.

The main feature of the Context Monitor is the modular monitoring process, used for all monitoring features with an extendible and configurable standardized process. It is the process description for all features how to attach and monitor data from Runtime Monitoring and Verification, as well as how to process the captured information.

4.7.1.1 Monitor

The Monitor module performs a permanent loop of monitoring for changes or creation of resources, which indicated a change in context.

Table 1: Monitoring of systems/sensors

Monitoring of systems/sensors	
Description	Monitoring of Runtime Monitoring and Verification Service, checking for usage, handing content information to subsequent processes.
Trigger Event	The process is a monitoring service, which performs a permanent loop that checks for changes (or creation) of resources indicated by a change in the content.
Parameters	The parameters to the analyser are the parsed content and Meta-Data from the Parser.
Process/Stages	The process performed is as follows: <ul style="list-style-type: none"> monitoring is started with the context configuration parameters

Monitoring of systems/sensors	
	<ul style="list-style-type: none"> • it starts monitoring the observed Runtime Monitoring and Verification Service • the information about its origin and all gathered data are given to the parser processes
Input/Output Data/Interfaces	Monitor provides (pre-processed) context data coming from monitored data sources.

4.7.1.2 Parser

The Parser module is connected to the monitoring input and the backend process of analysing and aligning the information. It is important to state that the parsing process itself does not analyse the data but offers access to it and allows as a connected upstream of the Analyser to parse and therefore sort the information in a specific manner. For example, plain text files may be accessible by the whole system and therefore the Analyser without the need of a Parser, but the Parser may utilize a schema or other utilities to arrange the information in Content and Meta-Data in a specific way. This will further be the foundation for the Analyser process which expects only this information and analyses on that without specific rearrangement of the resource. Another example may be the access of an external Web-Service where the Parsers controls and manages calling the Web-Service and prepares the data received from the service to hand it over to the Analyser.

Table 2: Parsing of monitoring data

Parsing of monitoring data	
Description	The parser offers access to data by sorting the information in a specific manner, parsing data coming from the monitoring system process. It selects based on the content and resource type the specific parser from a set of type specific computations and parses the Content and the special type Meta-Data and hands the data to the Analyser.
Trigger Event	Parsing process is triggered by the monitoring process. If the monitoring does not identify a change in the monitored resource, it is not triggered.
Parameters	The parameter handed over to the parser is the data monitored, containing environmental properties like the location of the resource to be parsed.
Process/Stages	<p>The process of the Parser is as follows:</p> <ul style="list-style-type: none"> • the parser gets the monitored data • the data is checked for their type • the appropriate specific parser according to the configuration is selected • the parser prepares the monitored data and possible meta-data • the data is handed to the appropriate Analyser
Input/Output Data/Interfaces	The Parser hands a set of content and Meta-Data to the according Analyser.

4.7.1.3 Analyser

The module for transferring the “raw” monitoring data into the standardized monitoring data is the Analyser. This function builds the monitored and parsed data into an RDF-based XMP representation of the recently monitored status.

The correlated Monitoring Data representing the last monitored status is needed to be stored in the context monitoring data repository as comparable source for other components.

As all data of all Context Monitor service modules will be based on the POJO-based principles, the serialization of the information provided from the Parser will be easily converted through self-describing classes and functions. The Analyser therefore invokes the handed over data and comprehends all single information in a single representation, which will be stored in the context monitoring data repository. The constructed data is based on a RDF-based XMP description.

Table 3: Analysing of monitoring data

Analysing of monitoring data	
Description	Constructing a RDF-based XMP monitoring data based on the data handed over.
Trigger Event	The Analyser is invoked by the Parser.
Parameters	The parameter to the Analyser is the parsed content and Meta-Data from the Parser
Process/Stages	The process of the Analyser is as follows: <ul style="list-style-type: none"> receives instances of parsed content and meta data constructs the Standardised Monitoring Data based on analysed data
Input/Output Data/Interfaces	The outcome of this function is a string-representation of a RDF-based XMP file, comprehending one standardized Context Monitoring Data, which is stored in the Context Monitoring Data Repository.

4.7.2 Context Extractor

The objective of the Context Extractor service is to identify the context of development processes or specified systems and to provide it for further use within the SmartCLIDE solution to other modules or external systems. Figure 48 shows the conceptual architecture for the Context Extractor service.

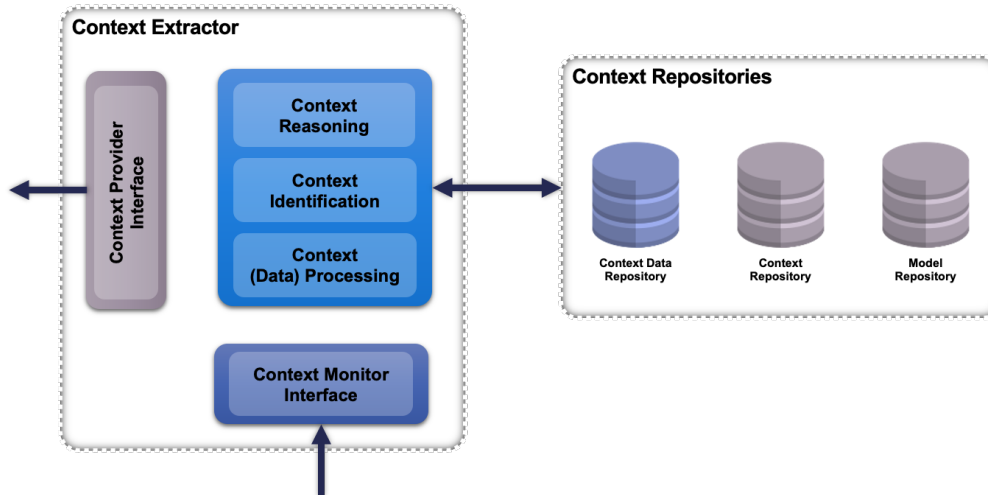


Figure 48: Conceptual Context Extractor architecture

The following sections describe the approach of the modules composed within the Context Extractor service. The service uses a context model for an integrated representation of the observed environment (e.g. development processes or users).

As shown in the figure above, the Context Extractor service identifies contexts based on context monitoring data, provided by the Context Monitor service (see Section 4.7.1), enhances it through different types of reasoning techniques (context reasoning), and provides the refined contexts for further exploitation to SmartCLIDE modules (or external systems).

The Context Extractor service is based on the services developed within the projects U-Qasar and SAFIRE. The service will be extended to address the specific needs of the generic SmartCLIDE solution, as well as the particularities of the four business cases. The collection of input data will be done by the Context Monitor service (see Section 4.7.1), and the Context Extractor service will analyse data in order to specify the current context and identify the parameters of the development process environment that could affect its performance. The identification of the respective context is being done based on the context models, which define each time what information is relevant to the observed context. The identified context information is being stored in the context repository as annotation to the content that is used in the observed context.

The Context Extractor service is composed of the following parts:

- The context model, tailored either to cover the generic needs of the SmartCLIDE solution (generic context model) or to support the specific needs of the business cases or companies (BC-specific and company-specific context model). The model is defined as high-level-structured representations. All methods included in the Context Extractor service have as basis for their functionality, this input context model.

- The context identification module , which analyses the monitoring data handed over by the Context Monitor service (see Section 4.7.1) and extracts a description of a context.
- The context reasoning module that reasons on the context provided by the context identification module and generates more precise context, which cannot be directly identified from the context identification module.

The parts are described in more detail in the following sections.

4.7.2.1 Context Identification

The context identification module analyses the monitoring data handed over by the Context Monitor service, and extracts knowledge such as information from runtime monitoring and verification services, and information on the activities performed and the environment characteristics occurring during the current on-going context.

The context identification is the first module to be executed in the context extraction process. It is triggered by the Context Monitor service upon receiving the monitoring data or on a certain period according to the customisation made. The data are analysed and mapped onto the given ontology by the context identification part to recognise as much as possible detailed contextual information. The identified initial context itself is sent to the context reasoning module for further processing and refinement.

Table 4: Context Identification

Context Identification	
Description	Analyses the context monitoring data provided by the Context Monitor service to identify contexts.
Trigger Event	The Context Monitor service provides context monitoring data or is done on a periodic basis based on the component configuration.
Parameters	The context monitoring data from the Context Monitor service in XMP/RDF format.
Process/Stages	<p>The process works as follows:</p> <ul style="list-style-type: none"> • Interpret the context monitoring data as RDF model. • Create an initial context instance to represent the current situation , since it may not be determined yet. • Analyse the context monitoring data RDF model to retrieve finer grained contexts. Add the identified context elements to the initial context instance.
Input/Output Data/Interfaces	An RDF model representing the identified context (an initial context instance and other relevant contexts and contextual elements attached to it).

4.7.2.2 Context Reasoning

This part reasons on the context provided by the context identification module , and generates more accurate contexts, which cannot be directly identified during the process in the context identification module.

Three types of context reasoning will be provided by the context reasoning module, namely ontological context reasoning, rule-based context reasoning and statistical context reasoning. They are performed one after the other, to refine the identified context.

4.7.2.2.1 Ontological Context Reasoning

Table 5) explores the semantics of the OWL ontology language and the definitions in the SmartCLIDE context model, to infer deductive results out of the identified knowledge context.

At the ontological level, deductive reasoning is based on the semantics of the OWL ontology language and the definitions in the SmartCLIDE context model. By performing ontological context reasoning, implicit information can be inferred out of the explicit information.

As SmartCLIDE uses RDF/OWL to model the contexts, deductive reasoning, such as transitive, or sub-class hierarchy reasoning is supported. After reasoning, each instance (identified with its URI) of the context element is clearly positioned in the context model. This is called “sub-sumption” and serves as basis for the next step rule-based context reasoning, as it requires all conditions in a rule to be explicitly stated.

Table 5: Ontological Context Reasoning

Ontological Context Reasoning	
Description	Context reasoning based on the semantics of the OWL ontology language and the definitions in the SmartCLIDE context model.
Trigger Event	Called by context identification.
Parameters	The identified knowledge contexts in RDF format. The SmartCLIDE context model definition in RDF/OWL format.
Process/Stages	Use the OWL ontology language rules to infer implicit RDF statements from the explicit statements in the SmartCLIDE context model and the identified context elements (i.e. “ <i>subsumption</i> ” as a first step).
Input/Output Data/Interfaces	Refined identified context.

4.7.2.2.2 Rule-Based Context Reasoning

Rule-Based context reasoning (Table 6) applies user defined domain specific rules to infer new contextual knowledge from the existing contextual knowledge.

Rule-Based context reasoning uses the same deductive techniques as in the ontological context reasoning, but with application-specific rules. During configuration phase, domain specific rules can be defined to help the Context Extractor service generate more useful and finer grained contextual knowledge.

The rules will be defined in rule files and can be updated any time. The Jena rule engine will be used to implement the rule-based context reasoning, so the rules will be defined in Jena rule syntax²⁰.

Table 6: Rule-Based Context Reasoning

Rule-Based Context Reasoning	
Description	Context reasoning based on the user defined domain specific rules.
Trigger Event	Called by ontological context reasoning.
Parameters	The refined knowledge contexts after ontological context reasoning. The domain specific rules defined by the user.
Process/Stages	Use the user-defined rules to infer more accurate contexts from the available identified, ontologically refined context.
Input/Output Data/Interfaces	Refined identified and reasoned context.

4.7.2.2.3 Statistical Context Reasoning

The purpose of statistical context reasoning (Table 7) is to determine the current context based on the available context information so far, and historical contexts.

Statistical context reasoning does not rely on strict logical rules, but instead tries to correlate information into possible relations, as suggested by the empirical data. The Context Extractor uses statistical context reasoning to rank on-going contexts according to the current available knowledge context, so as to determine the activity of the current systems and the corresponding context.

This reasoning is performed as the final step, as its purpose is to determine a current activity or processing step based on currently available context information, which can only be provided after the previous ontological context reasoning and rule-based context reasoning have been performed. Normally, a system might have several on-going activities, or processing steps, under certain contexts. Which is the currently active one (defined as the current context), needs to be decided. The statistical context reasoning compares the similarities between the contexts after the rule-based

²⁰ <https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/reasoner/rulesys/Rule.html>

context reasoning (which already contains an initial context created in the context identification) with the current on-going contexts and ranks them.

The context similarity measure computes the similarity value between two given contexts, which plays a very important role in the context determination. It supports the statistical context reasoning function and context-aware adaptations and selections. Basically, it compares two given contexts, by using the context hierarchy tree defined in the context model, to tell how similar they are. The actual output of the context similarity measure is a value between 0 and 1.

If the highest similarity value is above a configurable threshold (e.g. 0.95), the according on-going context is selected as the current one. Otherwise the initial context itself is used as the current one, which means a new knowledge-based context is created. This is one of the ways how contexts are populated in the Context Extractor service.

Table 7: Statistical Context Reasoning

Statistical Context Reasoning	
Description	Determines the most probable current context based on available knowledge context.
Trigger Event	Called by rule-based context reasoning.
Parameters	The refined contexts after rule-based context reasoning. A list of on-going contexts with their related historical contexts from the context repository.
Process/Stages	The process works as follows: <ul style="list-style-type: none"> • Call context similarity measure to compute the similarity between the initial context and the on-going contexts. • Sort the on-going contexts according to their similarity values. • If the similarity value of the first on-going context is above a given threshold, select it as the current context. • Otherwise use the initial context created in context identification as the current context.
Input/Output Data/Interfaces	A determined current context.

4.7.3 Context Provider

The purpose of the Context Provider service is to provide the current knowledge-based context and a list of other similar knowledge-based contexts to other modules/services/components.

The Context Provider service is responsible for fulfilling the following two tasks:

- to perform a context similarity check: the context similarity measure compares the current on-going contexts with previously acquired historical contexts in

the repository and provides the results to any other SmartCLIDE service (e.g. the DLE).

- to wrap the results into an RDF-based file format: this part wraps the extracted context to an RDF model, which is a more flexible format and readable by other modules/services/components, which can further use the SmartCLIDE context.

4.7.4 TRL 4 Lab Validations (Minimum Viable Product)

In order to demonstrate the readiness of the Context Handling services for TRL 4, a laboratory prototype was developed and made available on the project's Github repository²¹. The laboratory prototype includes three parts, namely the context model, the Context Monitor service, and the Context Extractor service.

To define what contextual information is important and will be taken into consideration for the context extraction, a context model was created. The context model was created using an ontology modelling tool (Protégé²²) and is saved structured in Resource Description Framework (RDF) format, as a Web Ontology Language *.owl* file. For this prototype, the model includes only some limited information defining the basic context framework of software services and development processes. The context model in this stage does not include advanced relations between the included concepts. The following figure presents the context model used for the laboratory validations.

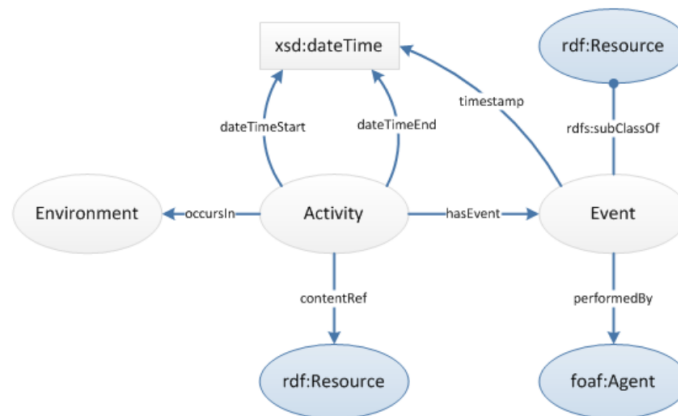


Figure 49: Context Model for Laboratory Validation

As seen above, the model for the lab validation defines that the important information for the Contextual Handling of SmartCLIDE includes for example data for the activities performed, information about the environment (e.g. services used), which should be monitored in order to identify the current context of operation. More

²¹ <https://github.com/eclipse-researchlabs/smartclide/tree/master/context>

²² <https://protege.stanford.edu/>

specifically, the context model configures the Context Monitor and Context Extractor service to monitor and identify the related information from the environment of operation. The context model is used as an input resource to the Context Extractor service, to identify and extract the related information from the monitored data.

The Context Handling does not have an user interface. It will use the RMV component as input and will provide its output to the DLE component.

The parsing of resources begins after the looping monitoring process receives a notification about new available data from the RMV. For the laboratory prototype the RMV data was simulated by providing the monitoring process information from a logfile.

The module for transferring the “raw” monitoring data into the standardized monitoring data is the Analyser. This function builds the monitored and parsed data into an RDF-based XMP representation of the recently monitored device’s status. The Analyser invokes the handed over data and comprehends all single information in a single representation, which is stored in the monitoring repository. The constructed data is based on an RDF-based XMP description.

The monitored data, in the form of RDF statements is given as an input to the context identification module of the Context Extractor service. Using SPARQL queries the identification module maps the monitored data to the context model (ontology) given as configuration input.

The result of the context identification is an RDF structure which describes all the identified context according to the provided context model.

4.8 User Interface / Workbench

In this section are presented approaches to address the main requirements and challenges for the development of SmartCLIDE User Interface and how the workbench will integrate the diverse tools provided by the several SmartCLIDE components in a seamless experience for the SmartCLIDE user.

The following requirements were identified by SmartCLIDE use case users as characteristics that the user interface shall provide:

- P40 - SmartCLIDE environment is easy to use by business stakeholders with limited programming knowledge;
- P41 - SmartCLIDE utilises a drawing canvas for "drag & drop" programming by arranging functional and decision blocks on a canvas rather than writing complex source code;
- P42 - SmartCLIDE provides live low-code programming capabilities for more complex scenarios;
-
- P46 - SmartCLIDE allows a BPMN editor to run in a web browser;
- P48 - SmartCLIDE provides support for a Docker Service toolbar so the process developer could also use an external service;
- P49 - SmartCLIDE interface provides for the configuration of individual services;
- P51 - SmartCLIDE supports customisation of graphical colour scheme and generic graphical elements;
- P52 - SmartCLIDE is able to visualise existing services;
- P53 - SmartCLIDE provides a visualisation of services and dataflows;
- P54 - SmartCLIDE interface supports the grouping of services (e.g. per category);
- P55 - SmartCLIDE includes a built-in documentation capability.

The following requirements were identified by SmartCLIDE use case users as characteristics that the user interface should provide:

- P43 - SHOULD - SmartCLIDE provides support for integrating an online coding IDE for code formatting and syntax error highlighting and integration with external services;
- P44 - SmartCLIDE online coding IDE provides auto-complete functionality;
- P45 - A BPMN template toolbar can be included in the SmartCLIDE UI;
- P47 - SmartCLIDE provides the ability to create re-usable process templates (e.g. BPMN);
- P50 - SmartCLIDE provides facilities for searching for services and abstractions;

In order to provide most of those requirements, the workbench must use and expose to the user several services provide by the SmartCLIDE components. Those services aim at supporting developers during the different stages of the product development lifecycle. Therefore, the workbench must organise the provided services in a way that enables the SmartCLIDE user to quickly identify, which tool will provide the most benefits at which stage of the service development lifecycle.

At the time of writing of this deliverable, the best candidate to be the basis for of SmartCLIDE IDE frontend is Eclipse Theia²³. This IDE is highly customizable and extensible, either on the graphical aspect of the environment and on the modification and addition of functionalities and behaviours. A possible approach for integrating the SmartCLIDE tools is the development of SmartCLIDE plugin that would provide most of SmartCLIDE functionalities to the user.

One example of such case might be the Smart Assistant. This component aims to support the user during different stages of the lifecycle. While in some cases the Smart Assistant must be integrated along with other SmartCLIDE tools to support their usage and interaction with the tools, in other cases like the code autocompletion, the Smart Assistant needs to be running on the background while the user is coding, using the integrated text editor, in order to make adequate suggestions.

A key aspect of SmartCLIDE will be the capability to allow the user to specify the desired behaviour of specific tasks by providing examples of data operations that SmartCLIDE will use to infer and generalise the desirable behaviour. This coding-by-demonstration mechanism can possibly be integrated with the editor of BPMN diagrams, allowing the user to try do define the behaviour of specific tasks by launching a dedicated assistant. This “Coding-by-Demonstration Assistant” will allow the user to specify one or more data sets defining a collection of steps needed to reach the desired output. This information can then be used by the DLE and Service Composition components to define the generic behaviour of that task. The specification of the actions that users can apply to the initial data sets on their “demonstrations” will depend on the use case’s needs and on actions deemed feasible by the SmartCLIDE technologies.

As a Cloud IDE, SmartCLIDE will have to support multiple users, working simultaneous either in collaboration or in parallel with each other. To support this individual development processes, SmartCLIDE must support the assignment of separate workspaces to each SmartCLIDE user. To enable this mechanism, the user interface and SmartCLIDE backend need to interact with each other in order provide a user management system. This system must be complemented by a workspace management tool that manages the workspace(s) associated to each user and provide a mechanism to dynamic allocation of resources.

The current best candidate for the basis of SmartCLIDE IDE is Eclipse Theia and this IDE is composed by a frontend and a backend server that communicate using

²³ <https://theia-ide.org>

JSON RPC. Theia IDE is often used along Eclipse Che²⁴ to support the automatic deployment of new Theia backends, each of which is associated to a specific user workspaces. As shown in the figure below, a Che Server can manage user workspaces, which can include several user projects and be distributed across different machines. To support this Eclipse Che also provides a user authentication and authorization system to control the access to team and individual workspaces.

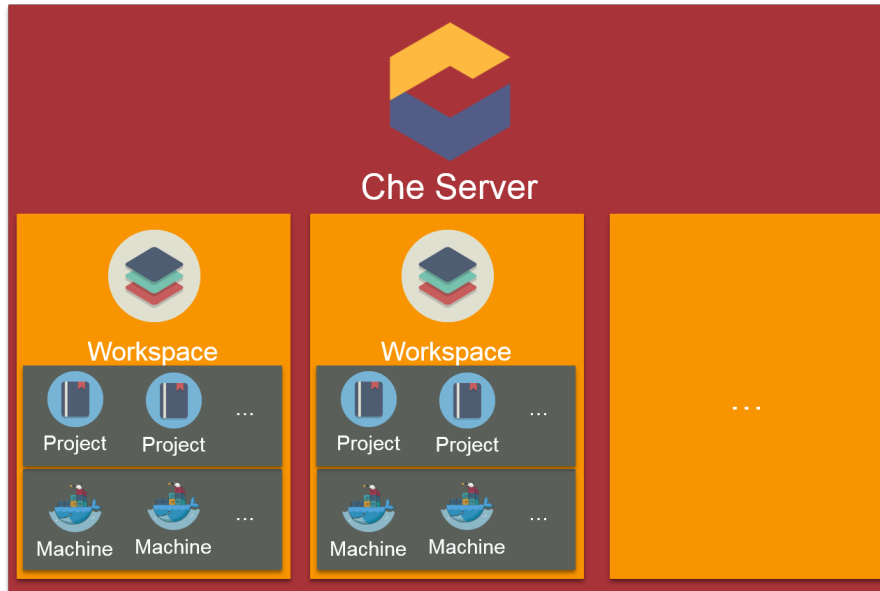
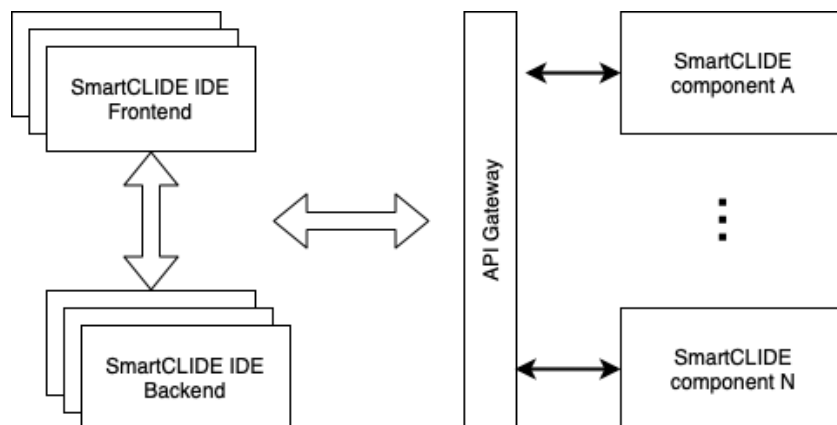


Figure 50: Overview of workspace organization in Eclipse Che

The SmartCLIDE IDE should function separated from the core components of SmartCLIDE backend. This would provide greater deployment flexibility and separate the core SmartCLIDE logic and functionalities from the IDE used. As the IDE needs to actively communicate with the several SmartCLIDE components to expose the SmartCLIDE functionalities to the user, as represented in the following Figure 51, the multiple instances of SmartCLIDE IDE will communicate with the other SmartCLIDE components through the API gateway that aggregates all APIS of each individual component, and manages the access to each one.



²⁴ <https://www.eclipse.org/che/>

Figure 51: Diagram of SmartCLIDE IDE and SmartCLIDE components communication

4.8.1 TRL 4 Lab Validations (Minimum Viable Product)

The User Interface and Workbench must take into consideration the entire user experience on SmartCLIDE ecosystem. It should include not only the direct experience of the user on the usage of the IDE, but also on the access to other information and functionalities provide by SmartCLIDE technologies.

The first point of contact of a user with the SmartCLIDE ecosystem is an authentication mechanism, which SmartCLIDE uses to determine which resources the user has access to, and identifies the previous work developed by this user. An example to a SmartCLIDE login page is shown in the Figure 52 below.

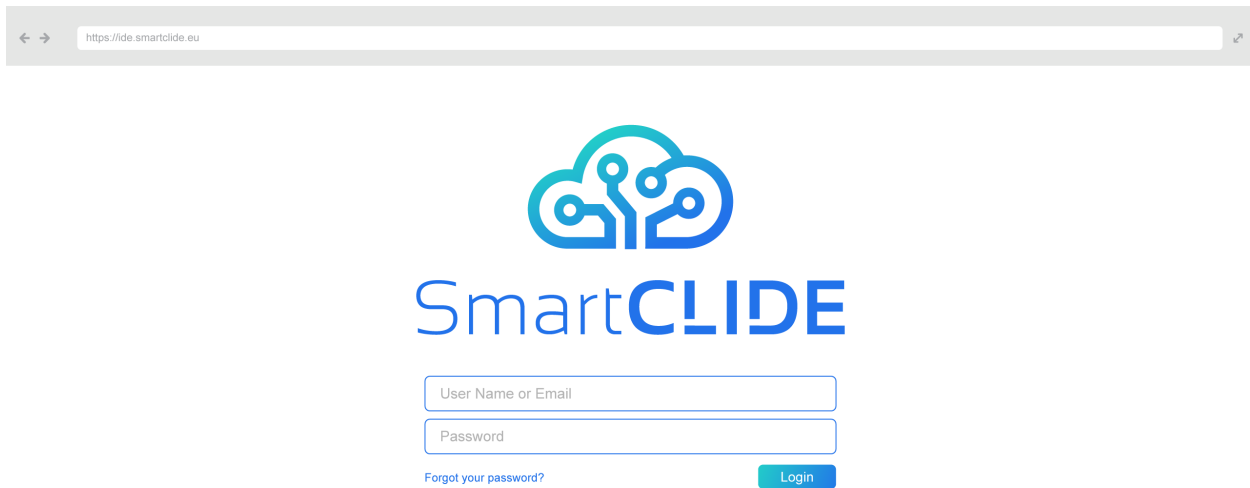


Figure 52: Login page for SmartCLIDE ecosystem

After a successful authentication the user should be provided with a Welcome dashboard, as shown in the Figure 53 below. The purpose of this dashboard is to provide a summary of information relevant for the user, identifying the services that were recently created or modified, the recent community on the considered services,

and the overview of the deployment status of the service deployments initiated by the user. Both the “Service” and “Deployment” sections are overviews and provide a connection to dedicated pages, where services and the deployments are presented in more detail. Also, in this page contains buttons “New Service” and “New Deploy” to provide the user with a quick access to common tools.

To facilitate the collaborative work on a team, this dashboard presents the information related to all the services the user is involved with or which are shared with the user. Consequently, the dashboard also shows the activity of other users on the services.

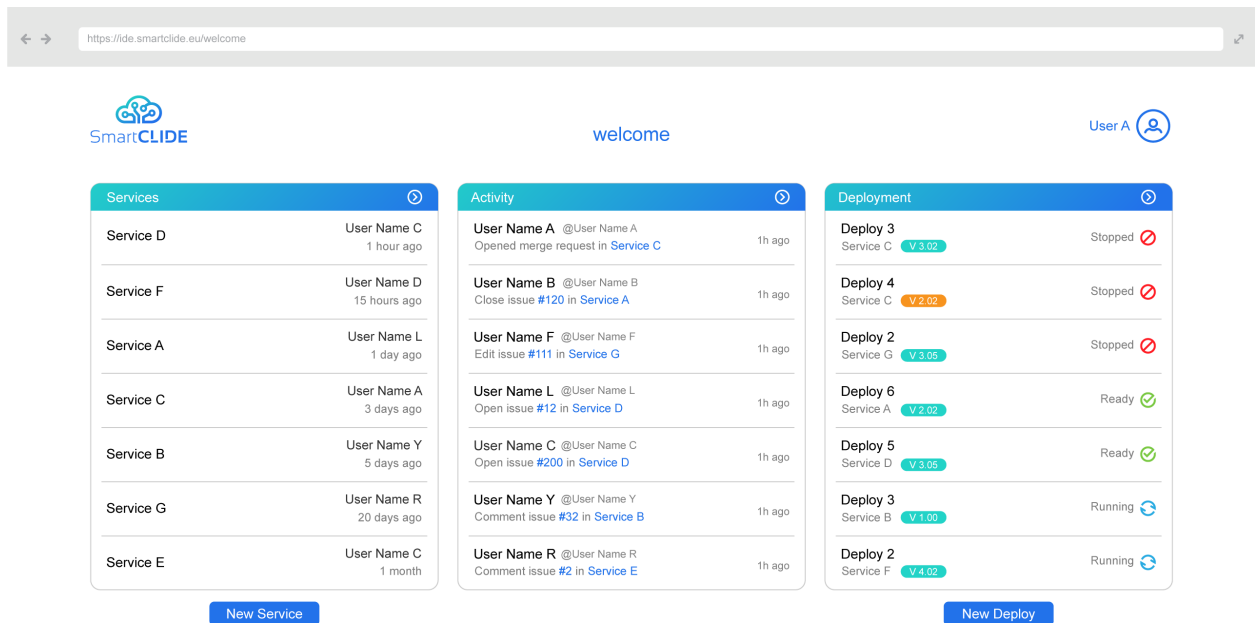


Figure 53: Welcome page when a user logs in

When the user wants to see more details about the services, he can access the page dedicated to the services. This page, presented in the following Figure 54, provides access to 3 lists of services, presented in 3 tabs:

- “My services” – presenting the list of services created by the user;
- “Shared with me” – showing the list of services created by other users (teammates), which were shared with the authenticated user;

- “Public services” - provides the user with a mechanism to search for other services available in the public domain.

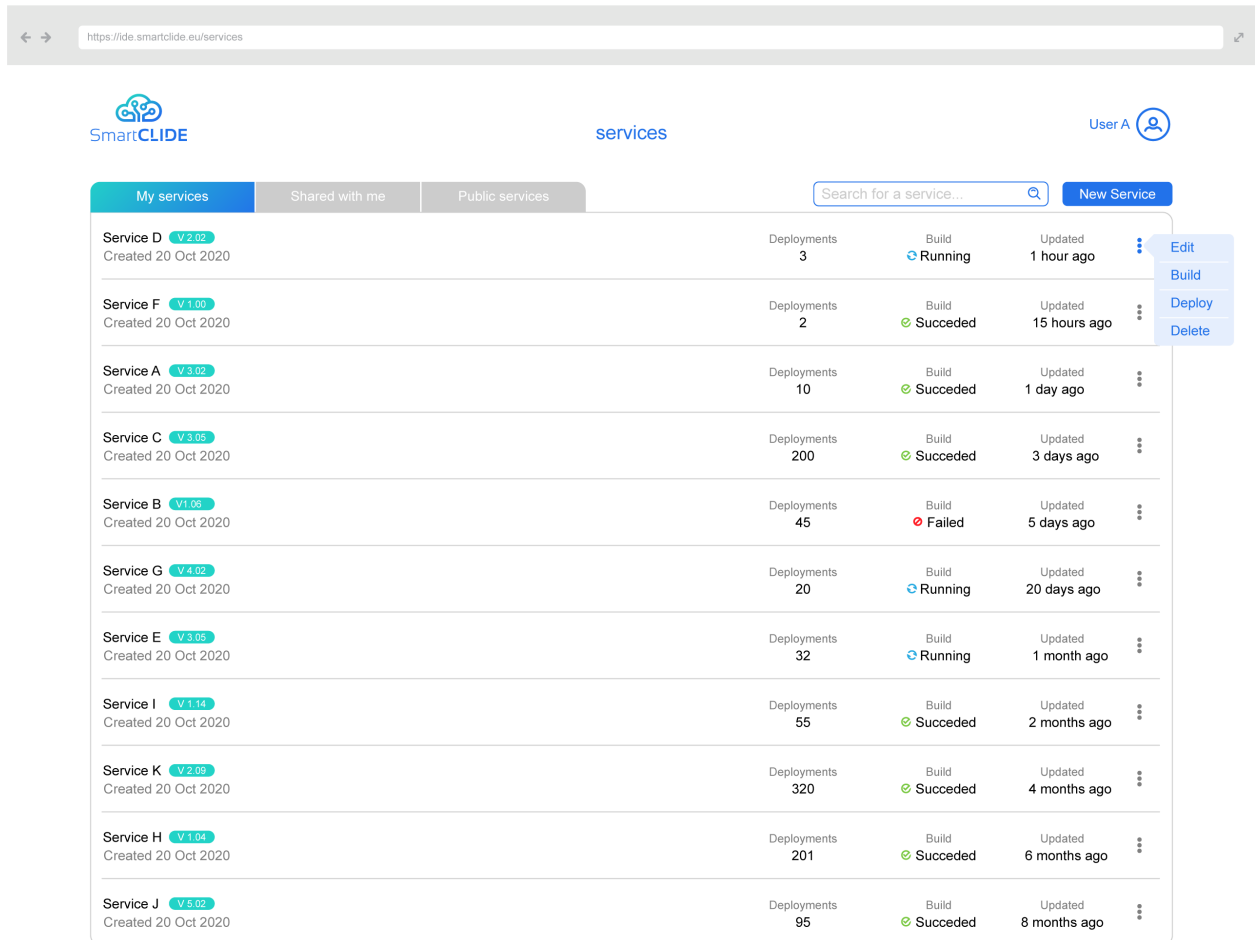


Figure 54: Services page

In those lists of services, the user can see relevant information about the services. Each service is characterised by its name, date of creation and date of last update, and the number of the last version. Moreover, if the service is configured to a CI/CD system connected to SmartCLIDE, the state of the building process can also be presented to the user. Service information is completed with the number of deployments of that service that SmartCLIDE is aware of.

For each service is provided a set of actions:

- Edit – Opens the SmartCLIDE IDE loaded with the selected service;
- Build – Starts the building process of the service, according to the building settings and CI/CD server configured;
- Deploy – Starts the deployment wizard for the definition of the deployment configuration and target infrastructure;
- Delete – Deletes de service from SmartCLIDE.

This page also provides a button “New Service ” that the user can use to start a new session on SmartCLIDE IDE in a clean state, ready for the implementation of a new service. An example of how this plugin could look like the SmartCLIDE IDE is shown in the Figure 55 below.

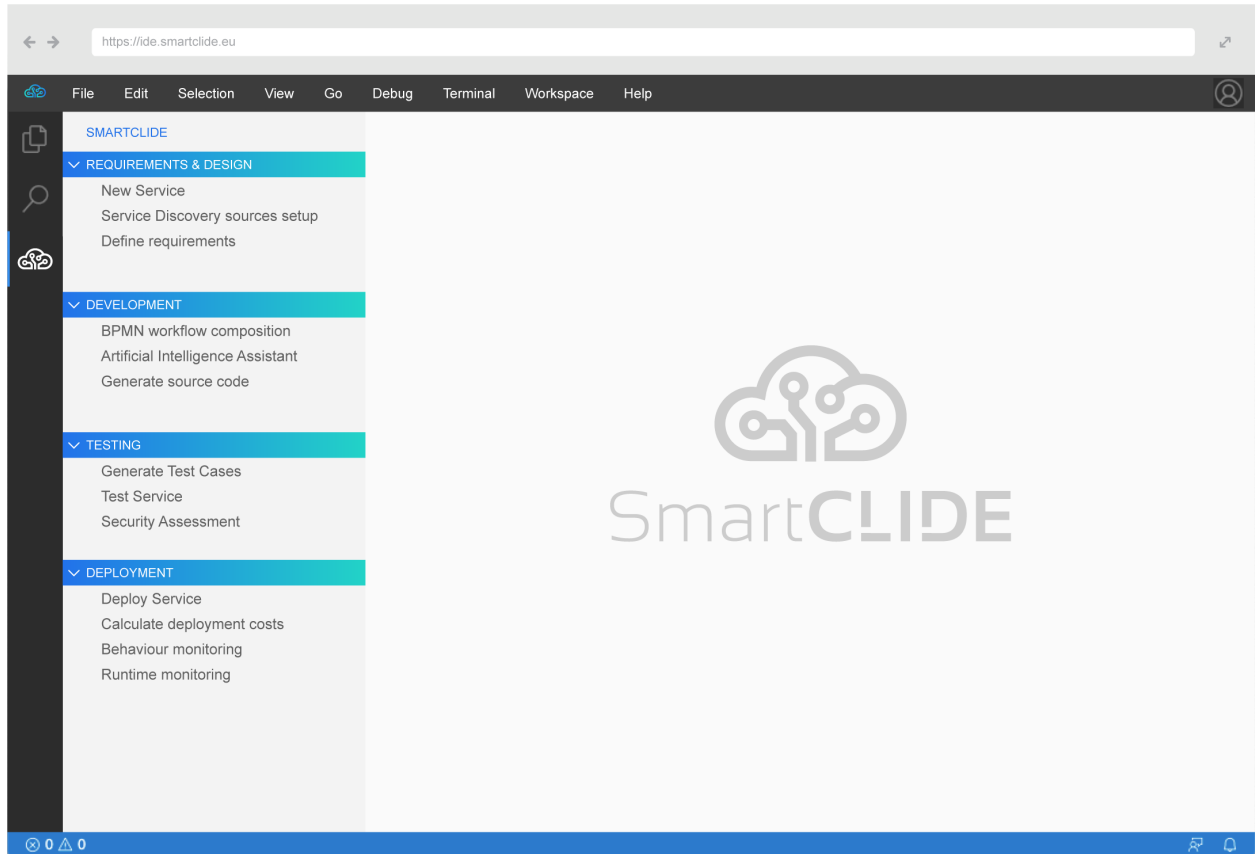


Figure 55: Example of SmartCLIDE plugin to support the development lifecycle

More details about the deployments can be accessed in the “deployments” page. In this page, represented by the following Figure 56, the user can see a brief description and status of all the deployments executed under the user’s instructions, and the lists of all Docker images that the user has access to.

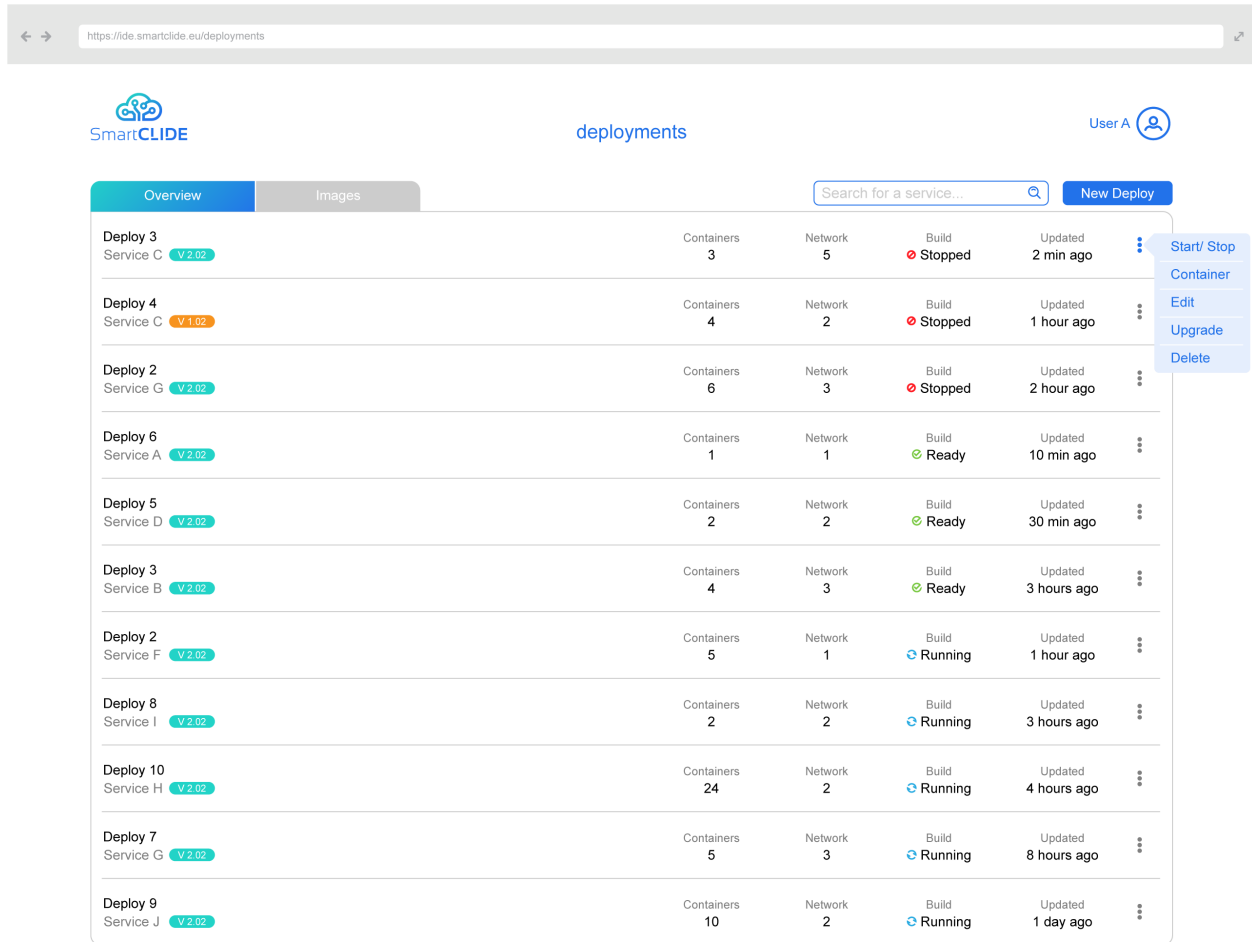


Figure 56: Deployments page

Each deployment is characterised by the name given to it (for easy identification), as well as which version of the service was deployed. It also provides information of how many containers and networks are deployed, the execution state of the deployed service, and for how long it has been in the current state. When the system detects that a deployment is not running the latest version of a service, the version “tag” changes its colour to orange, highlighting this situation to the user.

For each deployment, the user is provided with a set of actions that we can use to control the deployment get more information about it:

- Start/Stop – Used to start or stop a deployment;
- Containers – Allows to show more details about the state of containers in that deployment;
- Edit/Inspect – Opens SmartCLIDE IDE to allow the user to change deployment configurations and check deployment monitoring data;
- Upgrade – Re-deploys the service using the latest version of the images available;

- Delete – Stops the deployment and removes deployment information from pages.

The “New deploy” button starts the wizard for creating a new deployment. Most of the functionalities on this page will be supported by the Services Deployment component.

4.9 Smart Assistant

As stated, the goal of the Smart Assistant is to help both technical and non-technical users along with the Software Development Stages. To this effect, it will have to face a broad set of capabilities, being constantly present and giving different kinds of suggestions in the IDE interface.

A further explanation of the Smart Assistant potential suggestions is provided below:

- During the Requirements & Design phase:
 - Guidance to define requirements and user stories using the Gherkin syntax. Recommendations on the way requirements will lead to more accurate tests, allowing an automatic generation to be more performant based on NLP processing and existing tools.
 - Reusability of previous BPMN schemes or components needed in a task flow according to previous ones. This feature is supported by the DLE, which will learn from previous BPMN schemes and use the context to create behaviour patterns.
- During the Development phase:
 - Code analysis and syntax checking based on linters; potentially with AI enhancements to improve response time and highlight more suitable alternatives.
 - Autocompletion for different languages based on the developer's behaviour and context. Existing tools have to be checked, trying to extract generalizations in order to make the autocompletion suitable for different needed languages. The suggestion of parameters for completing code templates, based on existing code, will be tested too.
 - Guidance through the process of pushing changes into a version control repository. Typical checking behaviours or best practices can be suggested depending on the user development status.
- During the Testing phase:
 - Interpretations of the available tests. Once the tests are performed, subsequent actions to be conducted can be notified to the user depending on the results, or try helping to determine the degree of refinement needed to make a particular code successful.
 - Suggestion and offer of a set of acceptance tests, preferably based on initial Gherkin specifications. If Gherkin specifications are set, acceptance test generation tools are to be checked in order to provide a easy way to determine if desired functionalities are covered.
- During the Deployment phase:

- Suggestions or guidance through the deployment according to the Services Deployment component. Metrics to be observed and assessed during guidance can be: the suitability of different platforms, the degree of readiness of a service, the environment where a service is deployed.
- Suggestions on configurations for deployment based on services properties. Services will ideally have a set of deployment features (e.g. required resources, available environments) which can help to suggest a deployment setting (minimum, required, etc.) for their successful deployment.
- As a cross-wise helper, suggestions on the next steps to be taken based on the user context. User interactions with the IDE interface can trigger events, which properly managed along with the current action data may lead to appropriate suggestions based on typical behaviours. The common interface defined in Section 4.9.1 will make this suggestions visible to the user independently of the software development stage at which they are aimed, allowing the user to discard them or follow their indications.

All features have to be tested to determine their feasibility. A small subset, based on the use case requirements, will be prioritised.

To perform these tasks, the Smart Assistant is supported by the DLE and guided by the monitoring of users' actions, represented by the context abstractions. Both of these component connections will be performed through a REST API.

Something that needs to be discussed is where Smart Assistant capabilities will finally rest, as some of the suggestions can be provided by the DLE, and others don't seem to be necessarily supplied by the DLE in its role of AI provider. This means that maybe third parties' plugins can deal with particular tasks, or, in other words, that suggesting capabilities can be given without adding AI complexity or overloading the DLE with several different functionalities.

4.9.1 TRL 4 Lab Validations (Minimum Viable Product)

The Smart Assistant will rely on three basic items:

- The interface within the IDE
- The Context Monitoring
- The DLE intelligent capabilities

The **interface** will be integrated into the IDE as a constant, discreet helper, presenting a behaviour that highlights updates in a non-intrusive way, while stores and gives access to them anytime they are needed. It will consist of:

- A tab in the workspace of the IDE (see Figure 57). This tab will be bold in some way (colour, icon, etc.), noting when a new suggestion is generated. This tab will contain all the generated recommendations, split by the different phases of the software lifecycle. It will always be visible.

- Context menu, with a highlight of the location where the suggestion can be applied. It will provide options in case short actions or quick changes are required.
- Other possible visualisations, such as floating notifications when events susceptible to generate suggestions occur.

The Smart Assistant notifications system should be changeable both in a general way (simple suggestions/all suggestions) and in a more specific, layout and options degree of detail. That is, the grade of intrusion.

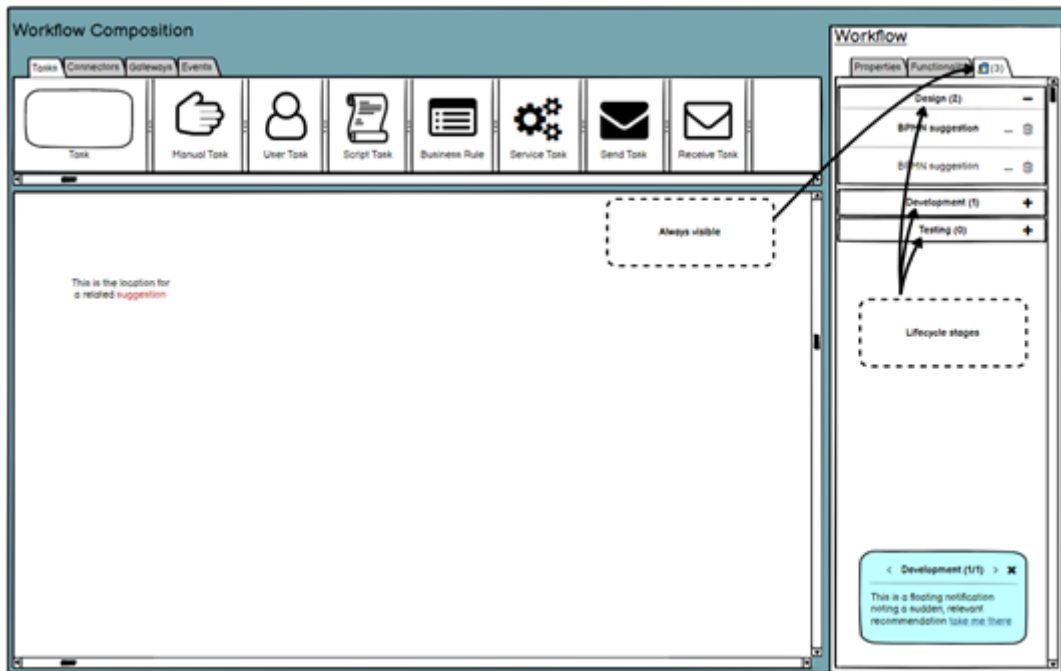


Figure 57: Smart Assistant - Workflow Composition

The **Context Handling** component. This item is crucial to acquire knowledge on which is the behaviour of the user, and therefore when the suggestions have to be created. Its abstractions are used by the DLE to predict behaviours.

The **DLE**. The Smart Assistant will rely on the predictions of the DLE models. Its role will depend on to which point AI is needed to support the suggestions, as some operations don't necessarily need from Machine Learning algorithms.

Initially, a basic set of suggestions will be leveraged for each phase, building a proof of concept for:

- The mechanisms of reaction to what the user is trying to do.
- The degree of assistance that can be expected for each software development phase.
- The accuracy and suitability of the recommendations.

These items will be adjusted depending on the needs observed by the partners and at the use cases, as well as other review actions, such as the success in the development of particular suggestion functionalities, e.g., adding programming languages.

4.10 Services Deployment

The “*Services Deployment*” component will provide a front-end for deploying new services in SmartCLIDE. This interface will allow the developer to visualize the status of deployed containers through a web-based interface accessible through a URL. Moreover, it will also allow the developer to deploy new services by providing a file that describes the application to deploy (i.e., stack).

To do so, the “*Services Deployment*” component will exploit the “*Service Creation, Composition and Testing*” component, which will allow the “*Services Deployment*” component to retrieve information about running containers, and to deploy new containers by using Docker Engine tools. This process is illustrated in the diagram below:

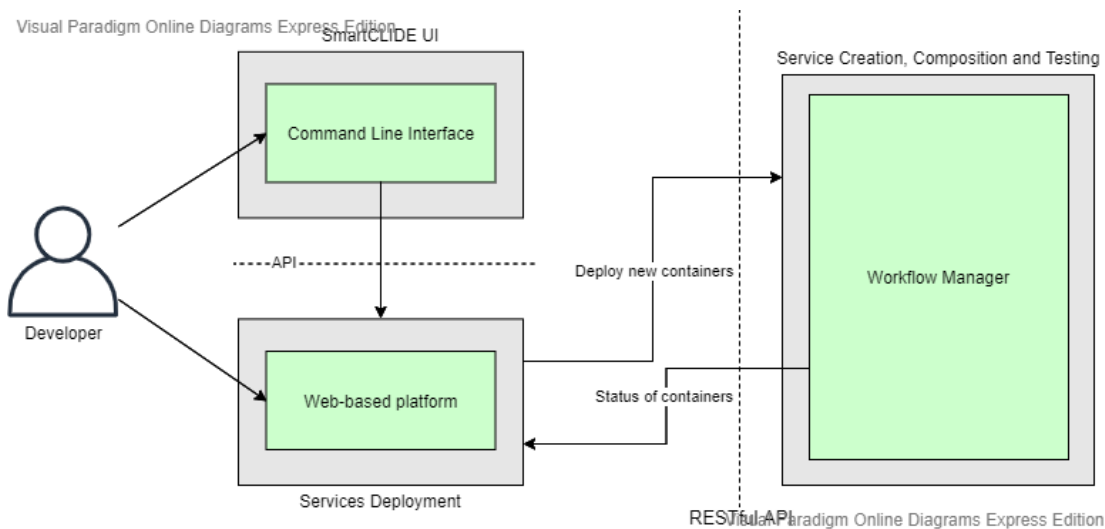


Figure 58: Process to deploy new Docker containers

4.10.1 TRL 4 Lab Validations (Minimum Viable Product)

After analysing the state of the art and initial requirements, we present the initial mock ups of the ‘*Services Deployment*’ component. This component provides a front-end for the deployment services and will also provide an API to allow other modules in SmartCLIDE to interact with the Composition Backend Services.

The User Interface will be accessible through a web URL, and will present a left vertical menu for choosing between different visualization options, such as overview, stacks, containers, images, or the deployment tool.

The first element in the menu is the Overview sheet (see Figure 59), which will show some general information about deployed stacks or containers. This information will be determined depending of what information is available through the backend services.

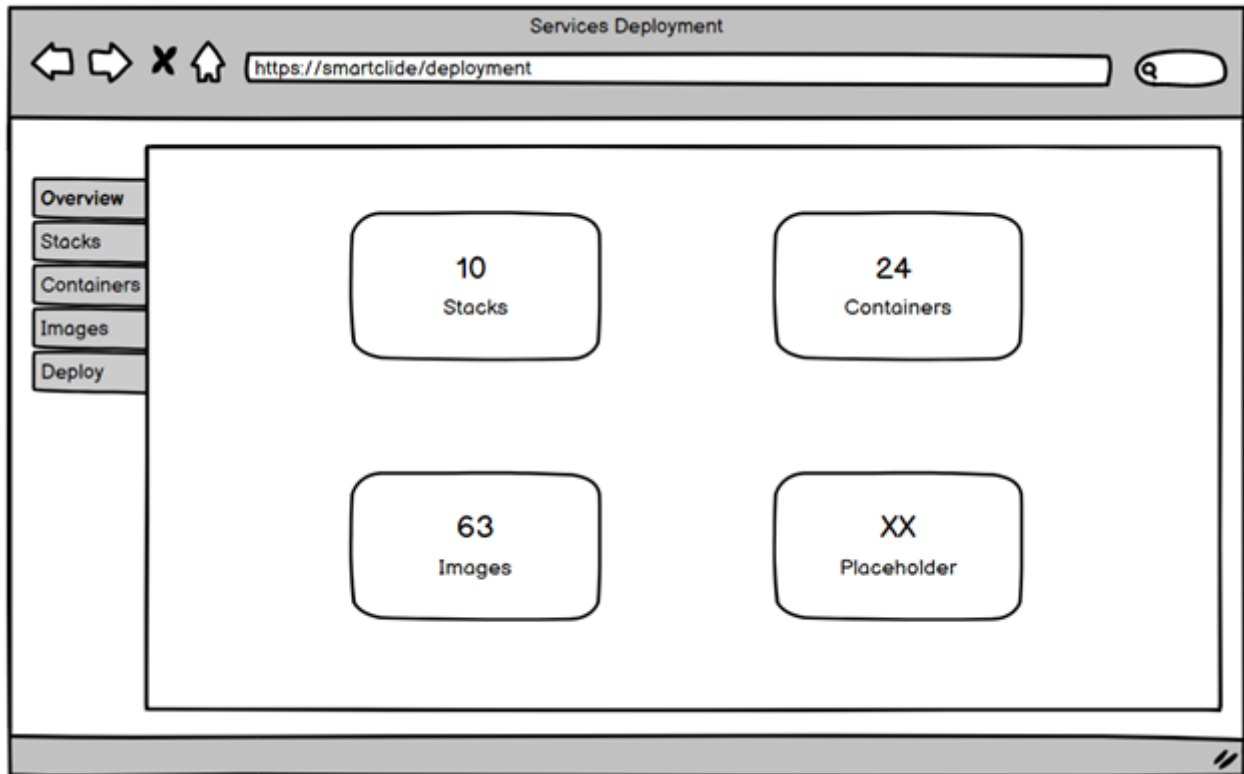


Figure 59: Service Deployment – Overview

Then, the next element in the menu is the Stacks sheet (see Figure 60), which will show the collection of Stacks recognised in the Docker environment. A Stack represent a complex application that is composed of a series of containers (e.g., a web server and a database system), so it can be deployed, started and stopped as a whole. The Stacks view will detail some information such as the stack name, the number of containers in the composition, the number of involved networks or its general state (e.g., running, paused...).

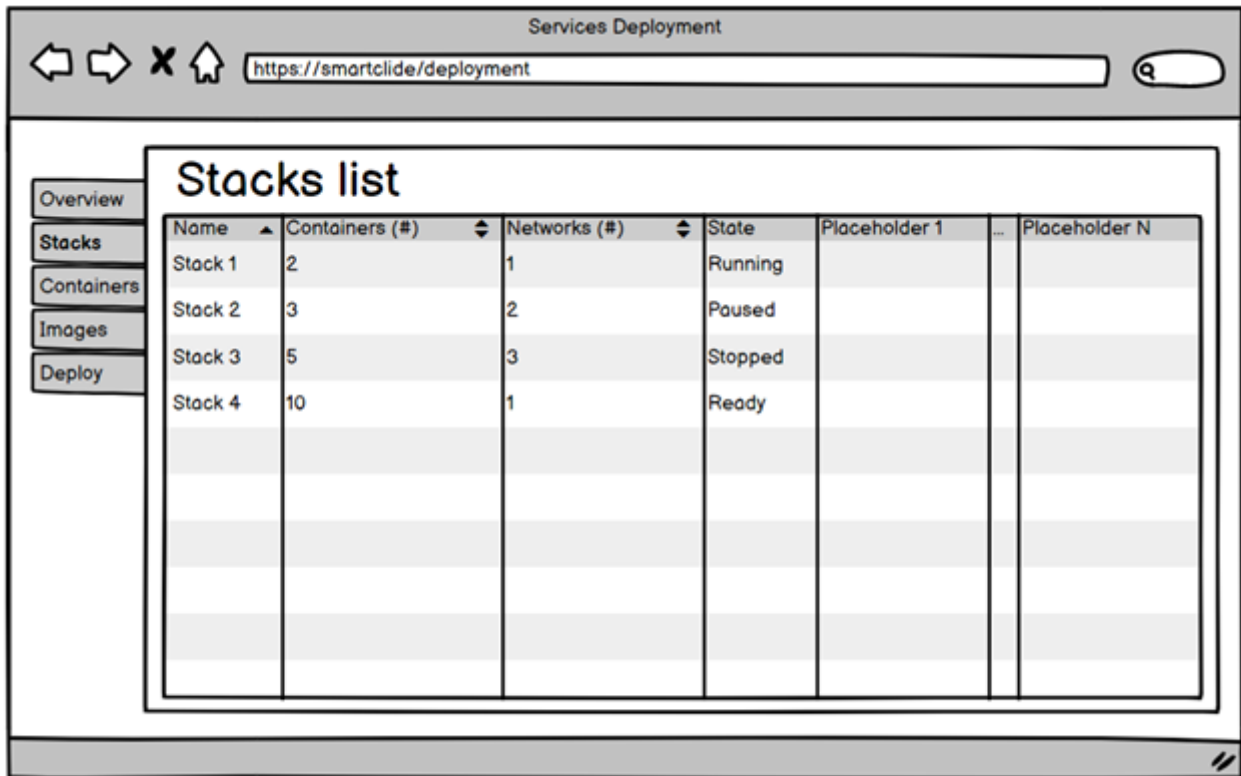


Figure 60: Service Deployment - Stacks

The next element in the menu is the Containers sheet (see Figure 61) in a similar way to the stacks one, but which will show relevant information about existing containers, such as the image, ports exposure, or its state.

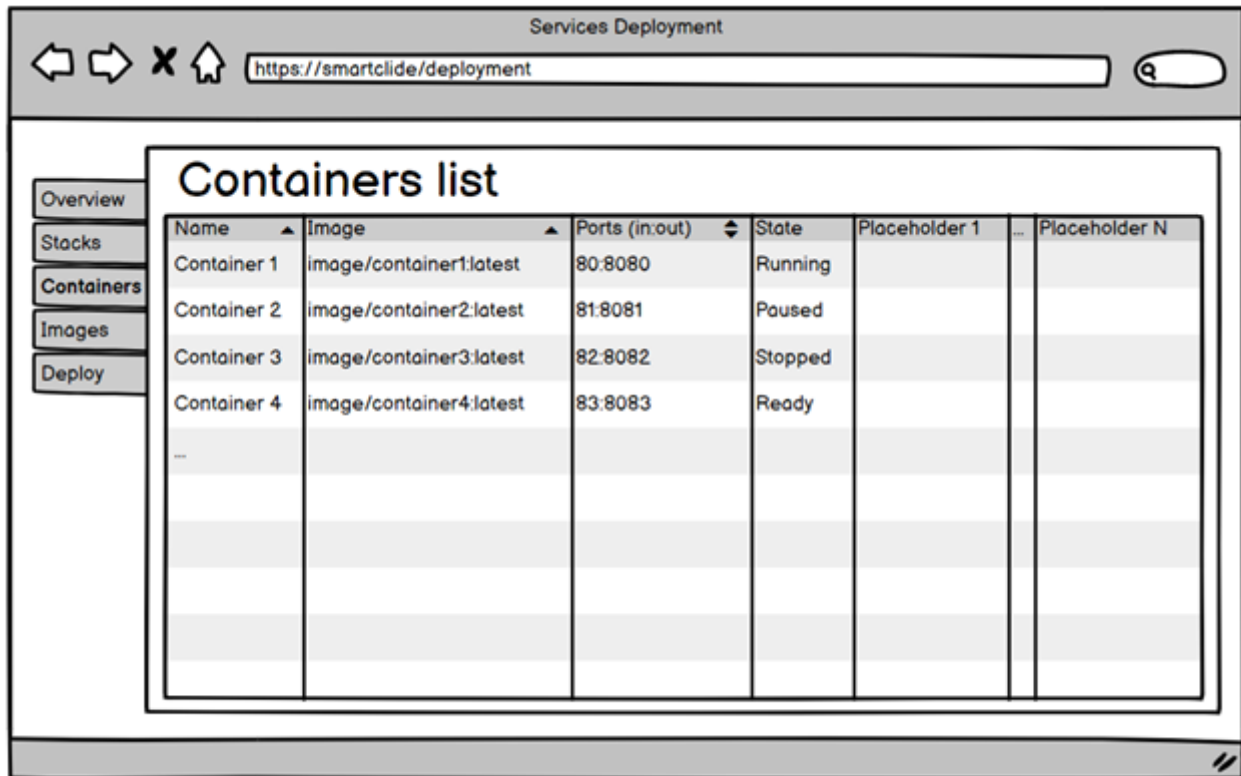


Figure 61: Service Deployment - Containers

Next sheet will include information about existing Images (see Figure 62), such as their name, time stamp, or size.

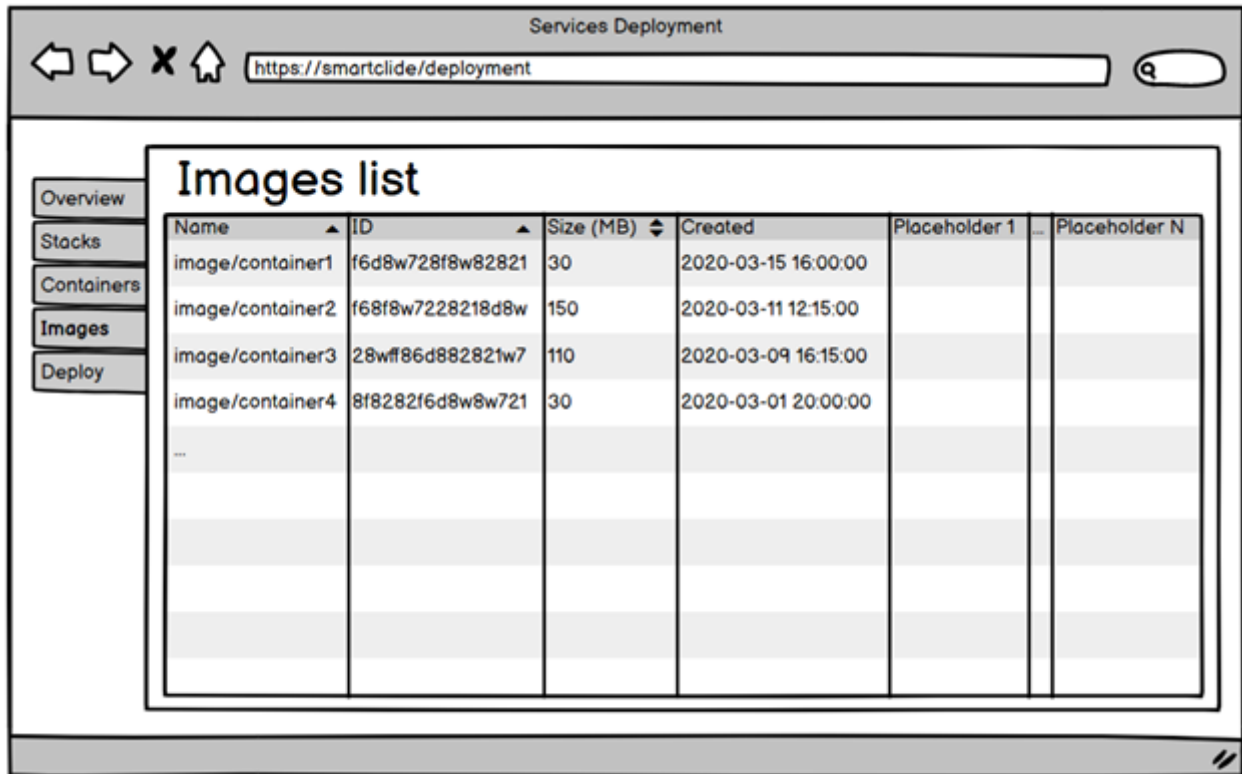


Figure 62: Service Deployment - Images

Finally, the last tab will allow the developer to Deploy a new stack of containers by providing the description file (see Figure 63). This functionality will also be accessible by a REST API.

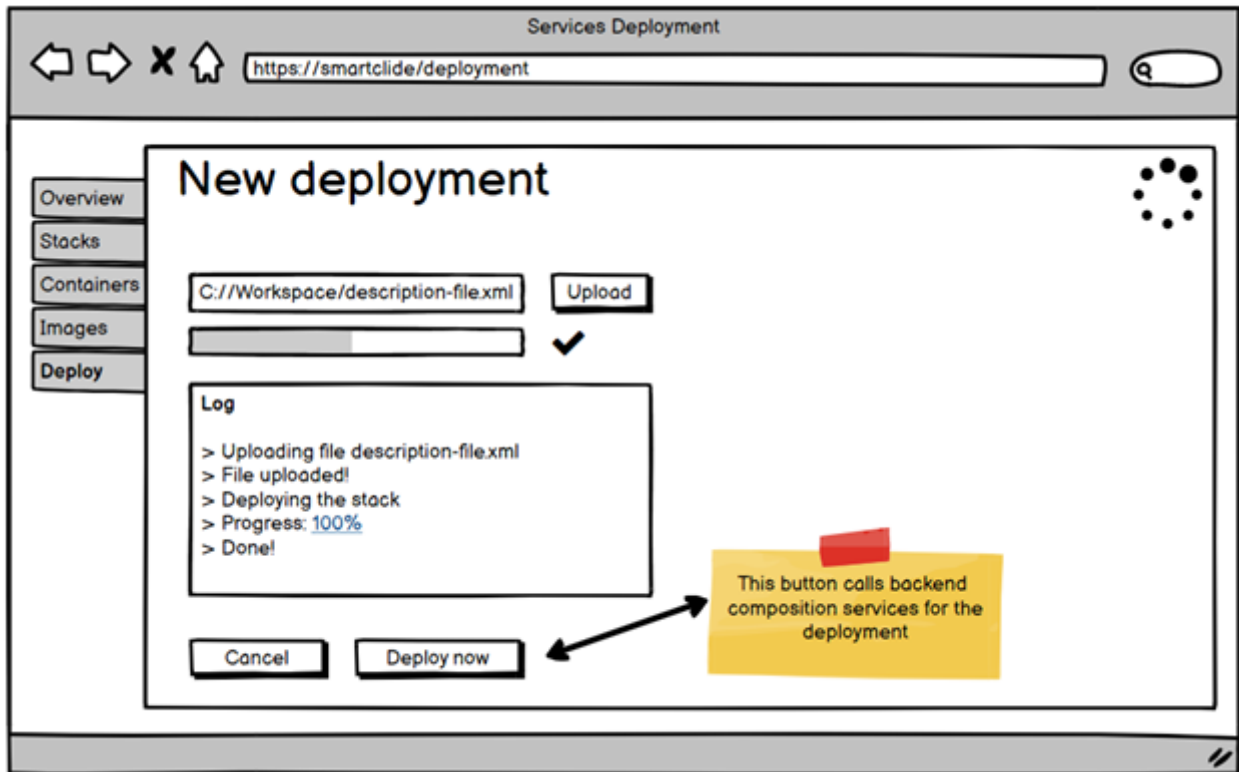


Figure 63: Service Deployment - New Deployment

5 Conclusions

This document presented a detailed description of the SmartCLIDE concept. A generic scenario was presented to explain the different aspects of the SmartCLIDE solution, and an architecture was chosen to describe the technical aspects, as well. Together with this, the main description of the different components of the SmartCLIDE solution was given.

The presented SmartCLIDE concept will serve as the main input for the succeeding work packages WP2 - Innovative approaches on Services Discovery, Creation, Composition and Discovery, WP3 - Implementation of SmartCLIDE framework components, and WP4 - Integration and Validation, which will specify, implement, and integrate the different SmartCLIDE components into the final SmartCLIDE solution.

6 References

1. Hommeaux, E. P., A. Seaborne (2008). SPARQL Query Language for RDF, *W3C Recomm.* [\url{http://www.w3.org/TR/rdf-sparql-query/}](http://www.w3.org/TR/rdf-sparql-query/).
2. LePendu, P., D. Dou (2011). Using ontology databases for scalable query answering, inconsistency detection, and data integration, *J. Intell. Inf. Syst.*
3. LePendu, P., D. Dou, G. A. Frishkoff, J. Rong (2008). Ontology database: A new method for semantic modeling and an application to brainwave data, In Proceedings *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.
4. Dou, D., H. A. N. Qin, P. LePendu (2010). Ontograte: Towards automatic integration for relational databases and the semantic web through an ontology-based framework, *Int. J. Semant. Comput.*
5. Reyes-Álvarez, L., M. del M. Roldán-García, J. F. Aldana-Montes (2019). Tool for materializing OWL ontologies in a column-oriented database, *Softw. - Pract. Exp.*
6. 2019. Python Eve, 2019 (2019). Available at: <http://docs.python-eve.org/en/stable/>, visited on 2019-10-16.
7. 2019. Flask, 2019 (2019). Available at: <https://palletsprojects.com/p/flask/>, visited on 2019-10-18.
8. OAuth 2.0 (2020). Available at: <https://oauth.net/2/>, visited on 2020-09-10.
9. 2019. Swagger, 2019 (2019). Available at: <https://swagger.io/>, visited on 2019-10-15.
10. Scrapy (2020). Available at: <https://scrapy.org/>, visited on 2020-09-11.
11. Beautiful Soup (2020). Available at: <https://www.crummy.com/software/BeautifulSoup/>, visited on 2020-09-10.
12. Apache Software Foundation2014. Apache Lucene Core, 2014 .
13. Apache Lucene Core (2020). Available at: <https://lucene.apache.org/core/>, visited on 2020-09-10.
14. PyLucene (2020). Available at: <https://lucene.apache.org/pylucene/>, visited on 2020-09-10.
15. Solr (2020). Available at: <https://lucene.apache.org/solr/>, visited on 2020-09-10.
16. Docker (2020). Available at: <https://www.docker.com/>, visited on 2020-03-11.
17. Kubernetes (2020). Available at: <https://kubernetes.io/>, visited on 2020-09-10.