Technical Requirements-Solution Architecture

Deliverable number: (D.2.1)

Author(s): Abdelhadi Belfadel (ISX), Ibad Kureshi (INLE), Dimitra Politaki (INLE)
Author'(s') affiliation (Partner short name): ISX, INLE

THE CIVITAS INITIATIVE IS CO-FINANCED BY THE EUROPEAN UNION

Deliverable No.		**2.1.**	
Work Package No.	**2**	Work Package Title	**TR – Solution Architecture**
Task No.	**2.1**	Task Title	**Technical Requirements – Solution Architecture**
Date of preparation of this version:		**21/12/2020**	
Authors:		**Abdelhadi Belfadel (ISX)** **Ibad Kureshi (INLE)** **Dimitra Politaki (INLE)**	
Status (F: final; D: draft; RD: revised draft):		**F**	
File Name:		**LEAD D2.1.TR–Solution Architecture_1.0.docx**	
Version:		**1.0**	
Task start date and duration		**01/06/2020 – 28/02/2020**	

This document is issued within the frame and for the purpose of the LEAD project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 861598.

The views represented in this document only reflect the views of the authors and not the views of the European Commission. The dissemination of this document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Revision History

Version No.	Date	Details
0.1	21/12/2020	1st draft version (ISX)
0.2	15/02/2021	Technology Architecture section (ISX)
0.3	04/03/2021	Update DDDAS and DSS sections (INLE)
0.4	05/03/2021	Merge content and add missing sections (3.2.3, 4.2.2) (ISX)
0.5	19/03/2021	Addressing reviewers' comments and improvements (ISX, INLE)
0.6	05/04/2021	Section 1.8 (INLE)
0.7	06/04/2021	Merge ISX and INLE updates – Final Draft (ISX)
1.0	12/04/2021	Update reviewers' minor observations (ISX) – Final document

Reviewers List

Name	Company	Date	Signature
Angel Batalla	LMT	01/02/2021	
Rodrigo Tapia	TUDELF	01/02/2021	

Contents

List of figures

List of tables

List of acronyms and abbreviations

Acronym /Abbreviation	**Meaning**
ABB	Architecture Building Block
DDDAS	Dynamic Data-Driven Application System
DSS	Decision Support System
DT	Digital Twin
GDPR	General Data Protection Regulation
KPI	Key Performance Indicators
LL	Living Labs
LSP	Logistics Service Providers
MMT	Multimodal Transport
PPP	Public-Private Partnerships
SBB	Solution Building Block
TLS	Transport Layer Security
UML	Unified Modeling Language

1. Introduction

1.1 LEAD project overview

LEAD will create Digital Twins of urban logistics networks in six cities, to support experimentation and decision making with on-demand logistics operations in a public-private urban setting. Innovative solutions for city logistics will be represented by a set of value case scenarios that address the requirements of the on-demand economy while aligning competing interests and creating value for all different stakeholders. Each value case will combine several measures (LEAD strategies): a) innovative business models, b) agile urban freight storage and last-mile distribution schemes, c) low-emissions, automated, electric, or hybrid delivery vehicles, and d) smart logistics solutions.

Cost, environmental and operational efficiencies for value cases will be measured in 6 Living Labs (LL). Evidence-proven value cases and associated logistics solutions will be delivered in the form of exploitable Digital Twins, incorporating the models that support adaptation to different contexts and that provide incentives for PPPs.

The LEAD consortium comprises 22 partners, all of whom are involved in the Living Labs, supported by 5 international partners for knowledge transfer. This structure incentivizes the co-creation of solutions by city authorities, logistics industry leaders, start-ups, and research experts in freight modelling, complex simulation and logistics optimization.

1.2 Deliverable purpose and scope

This report is developed in the context of WP2 - Digital Twin Model and Simulation Environment which designs and develops the technological core of the LEAD project. The purpose of this document, D2.1 Technical Requirements – Solution Architecture, is to specify the architecture of the LEAD digital twinning solution. As such, the deliverable starts with a reminder of the pertinent outcomes from the Knowledge Base – Reference Models (D1.2), and LEAD Value Case Scenarios (D1.4). It highlights how the outcomes of those documents influence the architecture of LEAD. Then, the document describes the LEAD components by showing the architecture in a graphical way. Based on this, Sections 2 to 5 describe each LEAD component in a detailed way.

1.3 Addressing the LEAD Description of Action

The following table maps the contents of this document to the requirements of the GA.

Table 1 Deliverable's adherence to LEAD objectives and Work Plan

LEAD requirements	Section of D2.1 addressing LEAD GA	Comments
D2.1 Technical Requirements - Solution Architecture System architecture and interfaces considering scalability, elasticity, availability and connectivity requirements and detailed development plan.		
-System architecture considering scalability, elasticity, availability and connectivity requirements. -LEAD subsystems and interfacing requirements -LEAD system components and data flows	2 3 4 5	
-Definition of the user interfaces allowing the interaction of human decision makers with the DDDAS -Models and algorithms	4.1	Further detailed in deliverable D2.4
-The interfaces between the various system components as well necessary API for orchestrating the Digital Twin	5.1	
-Interfaces between different models -Data / Sensor interfaces	4.1	Further detailed in deliverable D2.4
-Data Standardization and Data exchange requirements	4	
-Detailed development plan	3	
-The technical design requirements will take into account key industry requirements (system security, access security, data security and cybersecurity)	5	
-User interfaces and orchestration interfaces	4.1 4.2	Further detailed in deliverable D2.4

Task 2.1 - Technical Requirements - Solution Architecture Subtasks: ST2.1.1 Consolidation of smart city, transport and last mile logistics data standardisation and data exchange requirements. Analysis of existing standards (GTFS, TRANSMODEL, EuTravel Common Information Model) and ongoing standardization initiatives.	D1.8	
ST2.1.2 Requirements analysis and specification: LEAD subsystems and interfacing requirements. The technical design requirements will take into account key industry requirements such as overall system security (guarding access to the environment), access security (defining user roles and their data access along with outlining what data models and services can use), data security and cybersecurity (protecting data to internal and external standards in compliance with regulations), visibility security (reporting on where data came from, how it is put together and whom is consuming it). This subtask will also take into account the infrastructure of business stakeholders (distributed components) and will conclude in the specification of: a) LEAD system components and data flows b) Data / Sensor interfaces c) User interfaces and orchestration interfaces d) Interfaces between models, e) Models and algorithms	Sections 2-5	

1.4 Architecture development method

This architecture work is constructed and consolidated following the TOGAF standard and its guidelines, which is an open, industry consensus framework for Enterprise Architecture [1].

TOGAF provides an architecture development method and tools for assisting the acceptance and the production of architecture assets. It is based on an iterative process model supported by best practices and re-usable set of existing assets.

There are four architecture domains that are commonly accepted as subsets of an overall architecture work, all of which the TOGAF standard is designed to support.

The Business Architecture defines the business strategy, governance, organization, and key business processes. The Data Architecture describes the structure of an organization's logical and physical data assets and data management resources. The Application Architecture provides a blueprint for the individual applications to be deployed, their interactions, and their relationships to the core business processes. And finally, the Technology Architecture describes the logical software and hardware capabilities that are required to support the deployment of business, data, and application services; this includes IT infrastructure, middleware, networks, communications, processing, standards, etc.

As an architecture is a dynamic document throughout a project's lifecycle, this architecture development method is conducted in an agile way and can be iterated again to update the impacted components due to new constraints or requirements. These updates will be clearly defined in the next deliverables D2.3 Decision support system Interface and APIs, and D2.4 DDDAS (Looping Control) – Sensing – Data ingestion.

1.5 Target audience

This document is public and is aimed at the project partners, individuals in those organizations, the EU, EU Reviewers, and any individual who wishes to gain insight into the architecture of the LEAD Platform and the technical related work.

1.6 Deliverable context

This document is one of the cornerstones for establishing the research, and development baseline for the project. Its relationship to other documents is as follows, noting that some are used as a basis and others will derive from this document (see Figure 1):

- **Knowledge Base – Reference models (D1.2):** A report providing the reference guide for LEAD library of reference models for urban logistics, to be used by the partners to identify the most suitable model, or combination of models to develop their DTs.

- **Communities of Practice setup and Innovation Agenda - Value case scenarios and validation KPIs (D1.4):** Report providing the characterization of the Living Labs (LL) value case scenarios, and the definition of KPIs to target the impacts of the new LEAD scenarios based on a combination of different logistics measures.

- **Digital Twin Models Library (D2.2):** Report providing a set of open-source, case-specific software applications, for the specified models in D1.2, to be used in future Digital Twins. Together, these specifications and software become an open library from which the LL use case scenarios can source to populate their Digital Twins.

- **Decision Support System Interface and APIs (D2.3):** Report providing the implementation of the DSS and its related APIs enabling connection to external systems and inter-module communication for each of the modelling and simulation components. From a user perspective, this task reports the design of user interfaces and their related technical components where simulation scenarios can be configured and executed from.

- **DDDAS (Looping Control) - Sensing - Data ingestion (D2.4):** Report the implementation of the DDDAS enabling to ingest data from external systems to internal LEAD components, monitor data updates from physical twins, and the module that enables to manage the simulation process and the orchestration of models linked to the what-if scenario.

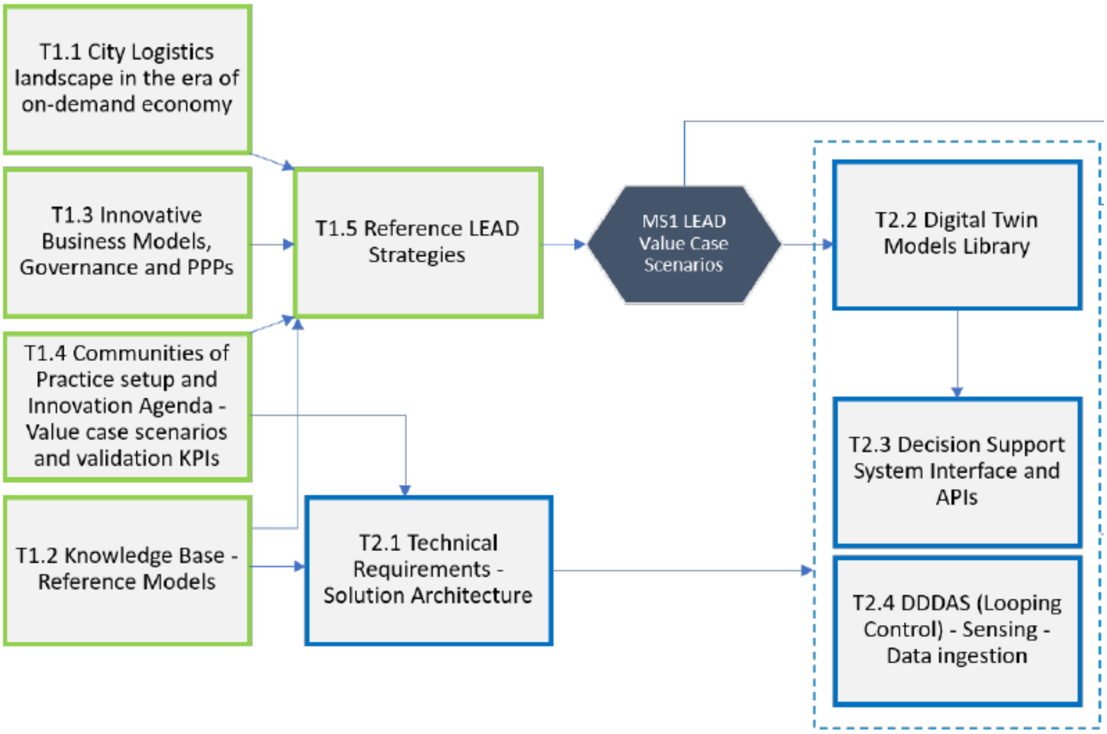

T1.1 City Logistics landscape in the era of on-demand economy
T1.3 Innovative Business Models, Governance and PPPs
T1.4 Communities of Practice setup and Innovation Agenda - Value case scenarios and validation KPIs
T1.2 Knowledge Base - Reference Models
T1.5 Reference LEAD Strategies
T2.1 Technical Requirements - Solution Architecture
MS1 LEAD Value Case Scenarios
T2.2 Digital Twin Models Library
T2.3 Decision Support System Interface and APIs
T2.4 DDDAS (Looping Control) - Sensing - Data ingestion

Figure 1 Deliverable (D2.1) dependencies

1.7 Document Structure

This deliverable is broken down into the following sections:

- **Section 2 (Architecture vision)**: This first phase ensures that the architecture is considered as a whole, by creating architecture content by cycling through the business, information system, and technology architecture. This enables to provide a big-picture of the targeted LEAD Platform, its main business process, the high-level architecture, and the implementability of the architecture. This phase helps as well to converge to a target and refine each level during the next iteration of the architecture development method.

- **Section 3 (Business Architecture)**: Based on the TOGAF recommendations [1], this phase enables to identify and analyze the Architecture Building Blocks (ABBs) that captures the architecture requirements to guide the development of the required solution building blocks. Each component in LEAD is considered as an ABB, considering, in the next architecture levels, several concepts such as the business, data, application, and technology requirements, the fundamental functionality and attributes (behaviour, interfaces provided, including security aspects), interoperability (where applicable) and dependence between building blocks.

- **Section 4 (Information System Architecture)**: This architecture level is focusing on the internal structure of the identified components during the business architecture, along with the required data structure and component interactions. It enables also to analyze technically the identified components that should cover the targeted features. We analyze the different solutions that exist in the market referred to as Solution Building Block (SBB) to implement the required component. Each SBB will be evaluated in the form of '+++' and '---' to cover the functionality, parameters, and security. The more '+', the better a solution will be covering such aspects, while more '- ', the less likely a solution is going to be selected for reuse. In terms of reusability, the tool evaluated are ensured to have an open-source and reuse/extend of source code open enough to allow the exploitation of such tool beyond LEAD.

- **Section 5 (Technology Architecture)**: This last phase enables to define the communication endpoints of the identified components in the IS Architecture to be used during the implementation. The focus is on API definitions and common data models that are provided for LEAD inter-communication, as well as the security aspects and the infrastructure that supports the LEAD platform.

1.8 Reference standardisation initiatives

The ultimate objective of introducing Digital Twins in last mile logistics is to improve the operation and efficiency of parcel delivery, reduce costs and externalities through forecasting and predictions of future states and support advanced decision making through the entire logistics lifecycle, while also fostering stakeholder participation via reliable real-life information. Technology enablers for building Digital Twins include modelling, predictive analytics and decision-making methods, and the use of lifecycle-oriented knowledge with historical and real-time operational and city data. The LEAD architecture design therefore considered key standardisation initiatives in logistics, transport data and digital twins to eventually enable the smooth flow of goods and the collaboration among the stakeholders across end-to-end logistics and supply networks.

Current standardisation initiatives suggest that LSPs handling the goods on their Seller-to-Buyer journey should use the same ID Key for the collection of goods despatched by the Seller as a shipment (in UN/CEFACT and GS1 terminology). That shipment ID Key can easily be passed down to any stakeholder involved in the journey of the goods. The ID Key may then be used by all stakeholders to share or retrieve the information necessary to handle the goods appropriately and to provide tracking information. The International Standards Organisation provides the standard (ISO 15459-6) intended exactly for that purpose. ISO 15459 ID Keys are unique and unambiguous regardless of the party that issues the ID Keys and assigns them to a specific object or entity.

More specifically, the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) propose a number of recommendations, standards, tools and resources that can be used to address the immediate and long-term challenges of supply chain and transportation logistics, also addressing the additional burden posed by the COVID-19 pandemic to facilitate global trade. There are a few UN/CEFACT projects currently active in related fields (track and trace, transport modal views of MMT directly relevant to last mile logistics). Relevant UN/CEFACT documents are:

- UN/CEFACT Multi Modal Transport Reference Data Model (UN/CEFACT SHIP/MMT-RDM) [2].
- UN/CEFACT Buy-Ship-Pay Reference Data Model BSP-RDM Version 1.0 (2019) [3].

In sequence, a number of UN/CEFACT processes and the related data exchanges and data elements as described in the UN/CEFACT Reference Data Models (RDMs) are being considered to align the last mile processes with existing UN/CEFACT standards:

- BOOKING: The booking process involving the booking requests and booking responses, covers space allocation, transport planning, and service requirements, and the details that appear in the waybill, and relates to the shipping instruction, the release of goods, the bill of lading etc.
- SHIPMENTS: The Shipping Instructions will typically follow the Booking from the Transport Service Buyer to the Transport Service Provider as a pre-cursor to the issuing of a waybill which then acts as the evidence of the transport contract.
- WAYBILL: The evidence of a contract between the Transport Service Buyer and the Transport Service Provider, usually issued on collection or receipt by the Transport Service Provider. The Waybill also shows who has the right of ownership of the goods being transported.

- REPORT/REQUEST: The status reporting processes cover ad-hoc or contractual reporting. The tracking of a shipments and goods or transport equipment is essential to a success of a supply chain operation, accurate and timely status updates allow the Seller and the Buyer to plan and manage the flow and timing and minimize the risk of disruption.

In terms of transport data standards, the following are considered:

- The General Transit Feed Specification (GTFS) defines a common format for public transportation schedules and associated geographic information. GTFS "feeds" let public transit agencies publish their transit data and developers write applications that consume that data in an interoperable way [4].
- Transmodel, the CEN European Reference Data model for public transport information [5], provides an abstract model of common public transport concepts and data structures that can be used to build many different kinds of public transport information system, including timetabling, fares, operational management, real time data, journey planning etc. Transmodel v6 is covering multimodal Public Transport, including flexible transport and also Demand Responsive Transport: most of the needs of bus, tramway, light-rail, metro, coach and long-distance rail are taken into account. The standard has been extended to cover alternative modes of transport, in particular vehicle sharing, vehicle pooling, vehicle rental, taxi (CEN TS 17413:2019).

In standards adopted in Digital Twins (DT), the Joint Technical Committee (ISO/IEC JTC 1) covers many areas including AI, automatic identification and data capture techniques, cloud computing, data usage, IoT. DTs is also a focus area of IEC Technical Committee 65, which develops international standards for industrial process measurement, control and automation. The Digital Twin - Reference Architecture (PWI JTC1-SC41-5) provides a standardized generic DT Reference Architecture using a common vocabulary, reusable designs and industry best practices. The document uses a top-down approach, beginning with collecting the most important characteristics of DT along its life cycle, abstracting those into a generic DT Conceptual Model, deriving a high-level system-based reference with subsequent dissection of that model into five architecture views from different perspectives [5]. Modelling work carried out in Tasks T2.2, T2.3 and T2.4 will build on the aforementioned initiatives and informed the following sections that present the LEAD project architecture and the key components of the envisioned platform.

2. Architecture Vision

This phase ensures that the architecture is considered as a whole, by creating architecture content by cycling through the business, information system, and technology architecture. This enables to provide a big-picture of the targeted LEAD Platform, its main business process, the high-level architecture, and the implementability of the architecture. This phase helps as well to converge to a target and refine each level during the next iteration of the architecture development method.

2.1 Generic Business Process Model

This section presents and analyses the targeted process by the LLs. To do so, we have first analyzed each LL value case scenario based on the inputs (description of the context, together with the actors, territories and companies involved, the main problematics they are facing and the targeted goals) provided in the deliverable D1.4 (Communities of Practice setup and Innovation Agenda - Value case scenarios and validation KPIs). Then we identified the common future workflow that is shared between all the LL value case scenarios.

Figure 2 is the outcome of the discussions occurred during the WP2.1 meetings, and depicts the generic business process that is designed and aligned with all LLs.

The LEAD generic business process is described below and in detail in Table 2 (from top to down). For each task, a **Who-What-Why** format is adopted to first identify the actor or component that manages or realizes this action. Then a description of the action is realized in the **What,** and finally the **Why** describes the objective or the outcomes of each task.

The first pool (tasks with blue colour) describes the workflow that manages the contextual information from the physical twins, and keeps the data updated during the whole lifecycle of the entire process. In addition, it enables to push an updated contextual data (if necessary) to the linked physical twins, in case a stakeholder needs to update the state of the physical twin.

The second pool (tasks with yellow colour) describes the workflow that manages the user tasks. These tasks are translated later on as User Interfaces where a stakeholder chooses a specific scenario or strategy, with a possibility to configure the model properties of the linked models before launching the selected scenario.

The third pool (tasks with green colour) describes the workflow managed by the DDDAS. It receives the selected scenario and model configurations from the user task. Then, sets the configuration for simulation, receives the required contextual data, initializes the data monitoring (if applicable) to check data threshold received from physical twins and restart the simulation if necessary, and finally prepare the execution environment to execute the targeted simulation.

The fourth and last pool (tasks with purple colour) described the workflow managed by the DSS. It receives the output of the simulation from the DDDAS as a set of successful scenarios. Then the decision system selects the best strategies among produced scenarios. Finally, the stakeholder that has launched the simulation scenario is notified when a generation of a detailed evaluation report is produced, to compare the produced KPIs with the ones calculated based on a refreshed contextual data of physical twins.

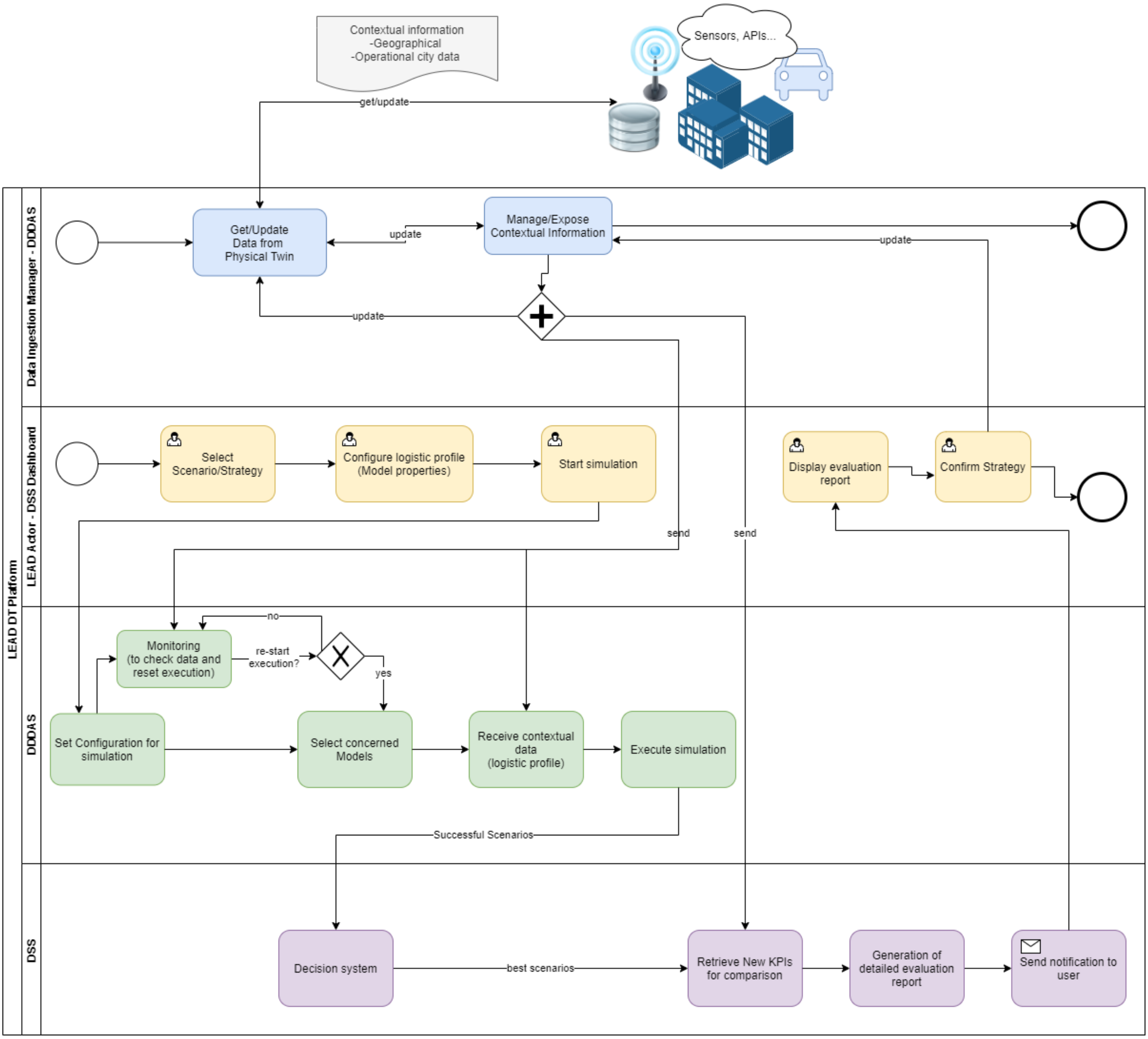

Contextual information
-Geographical
-Operational city data
Sensors, APIs...
get/update
LEAD DT Platform
Data Ingestion Manager - DDDAS
LEAD Actor - DSS Dashboard
DDDAS
DSS
Get/Update Data from Physical Twin
update
Manage/Expose Contextual Information
update
update
Select Scenario/Strategy
Configure logistic profile (Model properties)
Start simulation
Display evaluation report
Confirm Strategy
send
send
no
Monitoring (to check data and reset execution)
re-start execution?
yes
Set Configuration for simulation
Select concerned Models
Receive contextual data (logistic profile)
Execute simulation
Successful Scenarios
Decision system
best scenarios
Retrieve New KPIs for comparison
Generation of detailed evaluation report
Send notification to user

Figure 2 LEAD Generic Business Process

Table 2 Description of the Generic Business Process

LEAD Component / Stakeholder	Task	Description
Data Ingestion Manager	Get/Update Data from Physical Twin	**Who**: Data Ingestion Manager - Device Manager **What**: Manage interaction with heterogeneous environment of devices running different protocols **Why**: To exchange (get and set) data from device manager to physical twins
	Manage/Expose Contextual Information	**Who**: Data Ingestion Manager - Context Manager **What**: Manage interaction with internal LEAD components **Why**: To exchange (get and set) data from LEAD components to a context manager that manages contextual information
LEAD Dashboard	Select Scenario/Strategy	**Who**: LEAD Dashboard - LEAD Actor **What**: Select what-if scenarios **Why**: To select the required models and their order/sequences for each what-if scenario
	Configure logistic profile (Model properties)	**Who**: LEAD Dashboard - LEAD Actor **What**: Update model properties linked to the selected models of the scenario (urban freight characteristics, logistics needs…) **Why**: In case an actor needs to update/set model characteristics
	Start simulation	**Who**: LEAD Dashboard - LEAD Actor **What**: Launch the selected what-if scenario **Why**: To get the evaluation report
	Display evaluation report	**Who**: LEAD Dashboard - LEAD Actor **What**: Visualize the generated evaluation report after receiving notification of the execution of the selected scenario **Why**: To manage decisions and deal with changes
	Confirm Strategy	**Who**: LEAD Dashboard - LEAD Actor

		What: Confirm the simulation results **Why**: To record the results and push new data to physical twins where needed
DDDAS	Set Configuration for simulation	**Who**: DDDAS **What**: Set the preferences and new configurations to scenario models and system **Why**: To update default configuration based on user inputs (from configure logistic profile task)
	Monitoring	**Who**: DDDAS **What**: Monitor data updates from physical twins **Why**: To check data thresholds and reset simulation execution if needed
	Select concerned Models	**Who**: DDDAS **What**: Select the linked models to the what-if scenario **Why**: To prepare simulation execution
	Receive contextual data (logistic profile)	**Who**: DDDAS **What**: Receive contextual data from physical twins **Why**: To prepare data and model integration
	Execute simulation	**Who**: DDDAS **What**: Execute simulation **Why**: To produce paths to successful outcomes
DSS	Decision system	**Who**: DSS **What**: Select best scenarios among produced scenarios or predictive models to make decisions **Why**: To select best strategies
	Retrieve New KPIs for comparison	**Who**: DDDAS **What**: Retrieve New KPIs for comparison **Why**: To compare produced KPIs with calculated KPIs by real data of physical twins
	Generation of detailed evaluation report	**Who**: DSS **What**: Generation of detailed evaluation report **Why**: To generate evaluation report

	Send notification to user	**Who**: DSS **What**: Send notification to user **Why**: To read evaluation report and enable decision making

2.2 High-level LEAD architecture

Based on the scenario presented in the previous section, the following figures (Figure 3 to Figure 4) introduce the LEAD High-level Architecture diagram depicting the different LEAD components together with the major interactions they have as per their descriptions in the following in Section 2.2.1.

Figure 3 shows the legend for the LEAD architecture diagram of Figure 4. This legend shows the meaning of the symbols and stereotypes employed in the diagrams. Note that also shown are both the invocation arrows and the data flows, containing real data that is stored or used across components. Technical details like HTTP headers or response codes needed for command-like calls are considered to contain “no data”. For external components related to the describing component, the narrative focuses only on the external component use and interaction with the component rather than the internal features of the external component.

The boxes represented in Figure 3 have the following meaning:

- **Grey box**: This box represents internal modules of each LEAD component
- **Green box**: This box represents API modules that act as a bridge between the current component and any LEAD, or external to LEAD, component
- **Blue box**: This box represents any LEAD component that interacts with the current component
- **White box**: This box represents the user interface of the component.
- **Yellow box**: This box represents any external component or provider of certain tasks, e.g. external public API or sensor

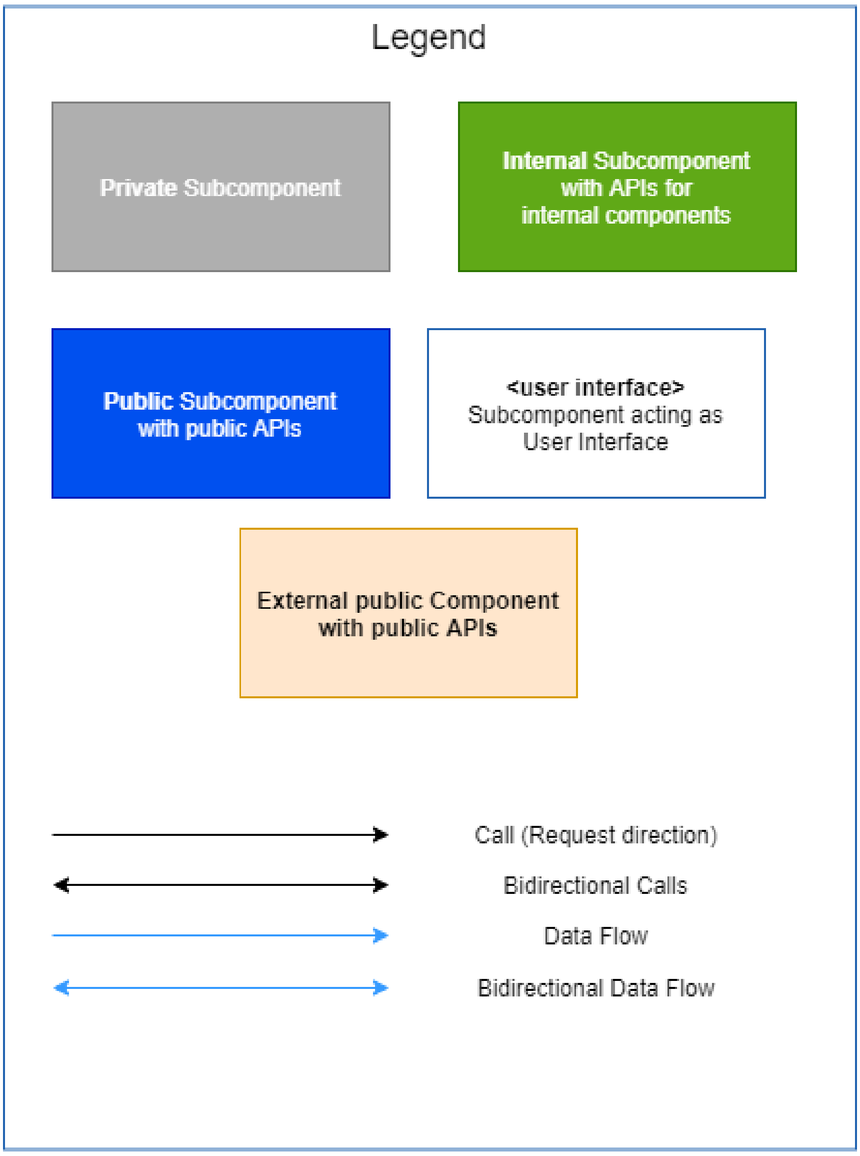

Legend
Private Subcomponent
Internal Subcomponent with APIs for internal components
Public Subcomponent with public APIs
<user interface> Subcomponent acting as User Interface
External public Component with public APIs
Call (Request direction)
Bidirectional Calls
Data Flow
Bidirectional Data Flow

Figure 3 Legend of the LEAD Components' Architecture Diagrams

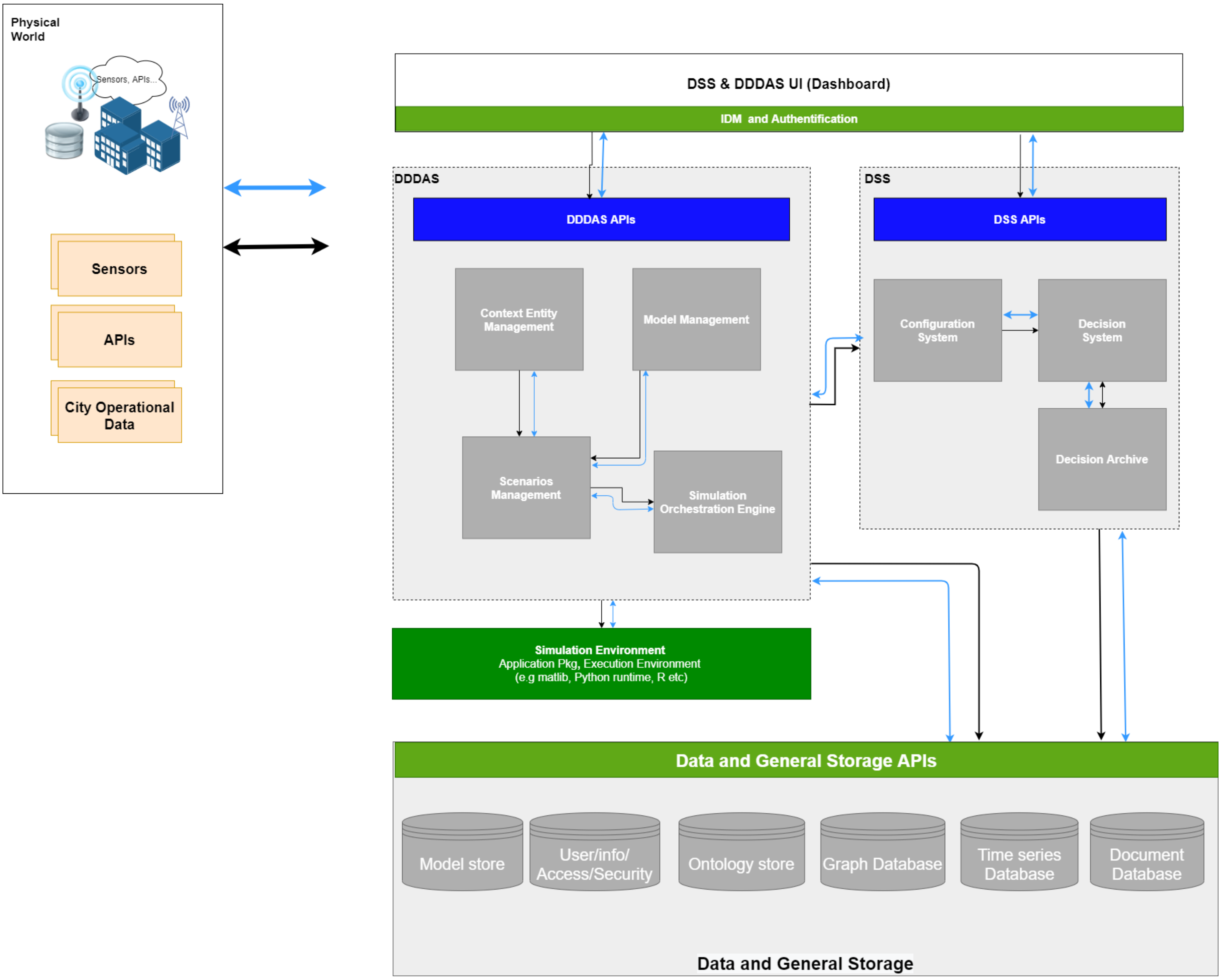


Figure 4 High-level LEAD Architecture Diagram

2.2.1 Main components description and interactions

LEAD platform consists of several components, which are described in this sub-section. The connections or communications between different components will be performed through APIs or direct calls inside the same component. The technology foundations as a first and high-level analysis are described in this section, however, specific technology selections for the components are decided in later sections.

As previously shown, the high-level architecture of the LEAD platform is depicted in Figure 4, and is composed of the following main components (from top to down):

- **Dashboard:** A user-friendly dashboard and interface, customizable for each LL to support logistics or city actors in running feasibility studies and exploring the results of alternative measures in a workflow-driven manner. The interface allows for the creation of simulation scenarios with specific win conditions or KPI's. The interface will allow the users to link the data sources to the models and define threshold parameters. As a digital twin, the simulation scenarios will be a pipeline of models (some running in parallel) feeding information into each other. The dashboard provides KPI metrics for both the physical and digital world. For the latter, the dashboard will show alignment of the simulations to actual ground conditions, as well as progress towards value scenario goals.

- **DDDAS:** The dynamic data driven application system will operationalize the user created scenarios by linking the data sources, models, and the simulation environment. At the first time step the DDDAS will read in and pre-process the data from the physical world, inject it into the simulation environment where the appropriate models have been instantiated. Each model is expected to be parameterized and therefore multiple instances will be created for parallel operation. To accomplish this the DDDAS will use containerizations, and virtualization to scale the system. At the end of the execution cycle, it is expected that the DDDAS will produce multiple predictions of the state of the physical world at the next time step based on the parameter sweeps of the various models in the scenario pipeline. These predictions will be sent to the DSS for further selection. For the lifetime of the scenario, the DDDAS will continue to monitor the data sources at each timestep and depending on established thresholds re-initiate the simulation scenario pipeline. Depending on the LL the DDDAS will also monitor and provide metrics to the user about the scenario. Situations, where the processing time of the simulation exceed the length of the time-step, will be raised with the user to ensure the DT can perform as close to real time as possible.

- **DSS:** The Decision Support System module employs Bayesian inference techniques to firstly decide which outcome of the simulation scenario is most likely achievable and addresses the defined scenario KPI's, and then recommends, to the city operators, the interventions required in the physical world to achieve the predicted outcome. At each time step the DSS continues to build its knowledge base of input parameters, model outputs, predicted outcome, and real-world readings. The later forming labels for the user interface, making simulation results readable and comprehensive to end-users.

- **Simulation Environment:** Every model will have its own execution and operational requirements. The Lead DT platform will use containerization, and virtualization to dynamically create these execution environments. These ephemeral environments will be created using the specifications provided by the model library and instantiated only when required. After execution at each time step these environments will be removed to conserve resources. Due to the parameterization of the models and temporal requirement of execution cycles, multiple instances will be dynamically created for each scenario being evaluated.

- **Data & General Storage:** This component stores the data managed in the LEAD Platform. This latter has different needs of data storage, e.g., model library, sensor data, events (time series), data files, log files or structured data. Each of them has its own requirements and constraints in terms of velocity of storage and querying, volume and updateability of the data, consistency or availability. It is not possible to think of a solution based on a single storage system. As a result, this component will offer several APIs to get access to several type of storage depending on the requirements and constraints of each component.

Each component has specific interactions with other LEAD components. However, at this high-level view, we focus only on the communication between main components. In the later architecture phases, and for each component, a more detailed interactions inside each main component are reflected and described.

These interactions are reflected in the form of a table with the following structure (columns):

- **Main Component:** This column defines the main component
- **Needs/Gives**: This column represents the following type of interactions:
  - **Gives**: The main component provides the component defined in column “With” the data described in column “What”
  - **Needs**: The main component needs from the component defined in column “With” the data described in column “What”
  - **Needs/Gives**: The “What” is exchanged on both directions

- **What**: This column describes the interaction or the exchanged data
- **With**: This column points at the component(s) that interacts with the main component

Table 3 LEAD main components' interactions description

Main component	Needs/Gives	What	With
Dashboard	Gives	User session data	DDDAS
Dashboard	Gives	User requests that need to be handled	DDDAS

Dashboard	Needs/Gives	User notification for displaying report	DSS
DDDAS	Needs	Definition of input data sources	Dashboard
DDDAS	Needs	Definition of model operating parameters	Dashboard
DDDAS	Needs	User defined scenarios with KPI's and operational thresholds	Dashboard
DDDAS	Gives	Selection of scenarios that would achieve the defined KPI's	DSS
DSS	Needs	All outcomes and parameters scenarios	DDDAS, Data Ingestion Manager – Context Entity Manager
DSS	Needs	KPIs	Data Ingestion Manager - DDDAS
DSS	Gives	Recommendation on best parameters to achieve scenario KPI's	Dashboard

2.2.2 Technical foundation

The Cloud Computing side of the LEAD Platform will be based on existing open source technology. Several cloud solutions exist in the market. The hybrid cloud stack provided by Cloudify[1], can integrate cloud and local environments. Other existing platforms are OpenStack[2], Cloud Foundry[3], and Apache CloudStack[4]. Classical approaches adopt the idea of deploying full virtual machines (KVM[5]). Based on the initial design phase, LEAD will follow

[1] https://cloudify.co/
[2] https://www.openstack.org/
[3] https://www.cloudfoundry.org/
[4] https://cloudstack.apache.org/
[5] https://www.linux-kvm.org/page/Main_Page

a more modern approach by applying containerization techniques like (Docker[6], Kubernetes[7]).

For the dashboard (end user portal side) implementation, LEAD aim to build this front-end with open source solution for creating customizable dashboards such as Grafana technology tools[8], that is widely used to compose observability dashboards with metrics, logs, and application data.

For the DDDAS component, a job scheduler will be employed over Docker/Kubernetes APIs, with a solution to handle data ingestion such as Kafka [9] backbone, along with an open source framework for creating and managing digital twins in the IoT such as Eclipse DITTO[10] for the Context Entity Manager. For the DSS component LEAD will employ Bayesian inference techniques coded in Python. Regarding the Simulation Orchestration Engine, solutions for IT automation to configure systems, deploy software and orchestrate IT tasks will be considered such as Ansible[11], SLRUM[12] and KVM[13].

For the Data and General Storage, it will be based on several existing open source technologies and must cover a very diverse storage need. Different storage will be considered, for instance, as sensor data can be generated very fast, a traditional database can reach its limits on processing speed and size quite fast which demands a different approach for the management of this kind of data. In this case, big data technologies are proposed to get the necessary speed and scalability. Time series database like InfluxDB[14], or document-oriented database such as MongoDB[15] will be used for the storage and querying of the data.

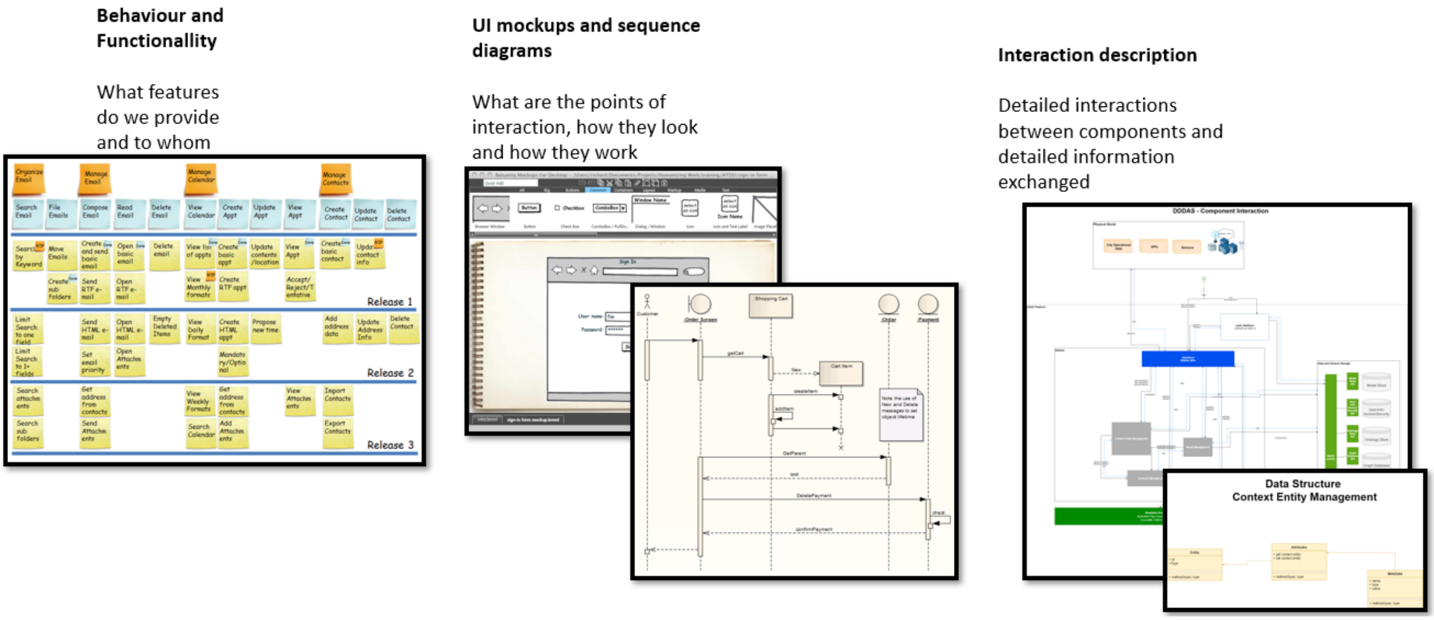
3. Business Architecture

This section describes the functional specification, and it describes how the LEAD platform will work from the user's perspective. As any functional specification, this section does not deal with the technical aspects on how the software is implemented. Instead, it explains the features provided by the components, specifying their features and interactions, including screens, menus, dialogs, etc.

[6] https://www.docker.com/
[7] https://kubernetes.io/
[8] https://grafana.com/
[9] https://kafka.apache.org/
[10] https://www.eclipse.org/ditto/
[11] https://www.ansible.com/
[12] https://slurm.schedmd.com/documentation.html
[13] https://www.linux-kvm.org/page/Main_Page
[14] https://www.influxdata.com/
[15] https://www.mongodb.com/

The functional analysis per each component is made from three perspectives (c.f. Figure 5):

- **Behaviour and Functionality**: Containing a story map with the features and functionality offered and the user stories that need to be developed to implement that functionality
- **Interaction descriptions**: describing for each component the set of interactions that it has with other LEAD components and users and describing the exchange of information flows that will be critical for a unified LEAD platform
- **UI mock-ups and sequence diagrams**: Describing, for each functionality, the interactions of the component with the user or with other LEAD components



Behaviour and Functionallity

What features do we provide and to whom

UI mockups and sequence diagrams

What are the points of interaction, how they look and how they work

Interaction description

Detailed interactions between components and detailed information exchanged

Figure 5 Business Architecture Artifacts

In terms of the behaviour and functionality, story maps describe the functionality on different levels of aggregation / abstraction. In order to define the story maps interactively, the online tool “Draw.io” has been used in the preparation of this deliverable. The three main elements are (see Figure 6):

- **Main activity/Component**: It provides a coarse definition of the behaviour of the component.

- **Tasks**: Activities are divided into tasks, which are features needed to complete an activity. Tasks are organized from left to right following a logical sequence to complete the activity. Tasks have related Sequence diagrams designed with UML (Unified Modeling Language) to support its description.

- **Subtasks** (User stories): Describe features of an application from the point of view of the subject who expects the new feature. The subject is not restricted to a LEAD user and can be any entity with a behaviour, e.g. the component being described, another component, etc.
  User stories follow a standard format: as a ***who***, I want ***what*** so that ***why***. This way, user stories capture in a simple sentence who wants what and how will the subject benefit from the new feature. To force this format, user stories are written in a schematic way, just specifying the who, what and why syntactical functions.
  User tasks should include acceptance criteria – a checklist that determines when the user story is considered as done. The acceptance criteria are also expressed from the point of view of the subject that formulates the user story and provides a detailed description of the criteria by which user stories should be evaluated and validated. User Stories have a unique ID per story (US001) in each story map. User stories are organized in releases in an incremental development plan. Thus, there are releases defined for the software deliverables of each component (e.g. M15 and M25).

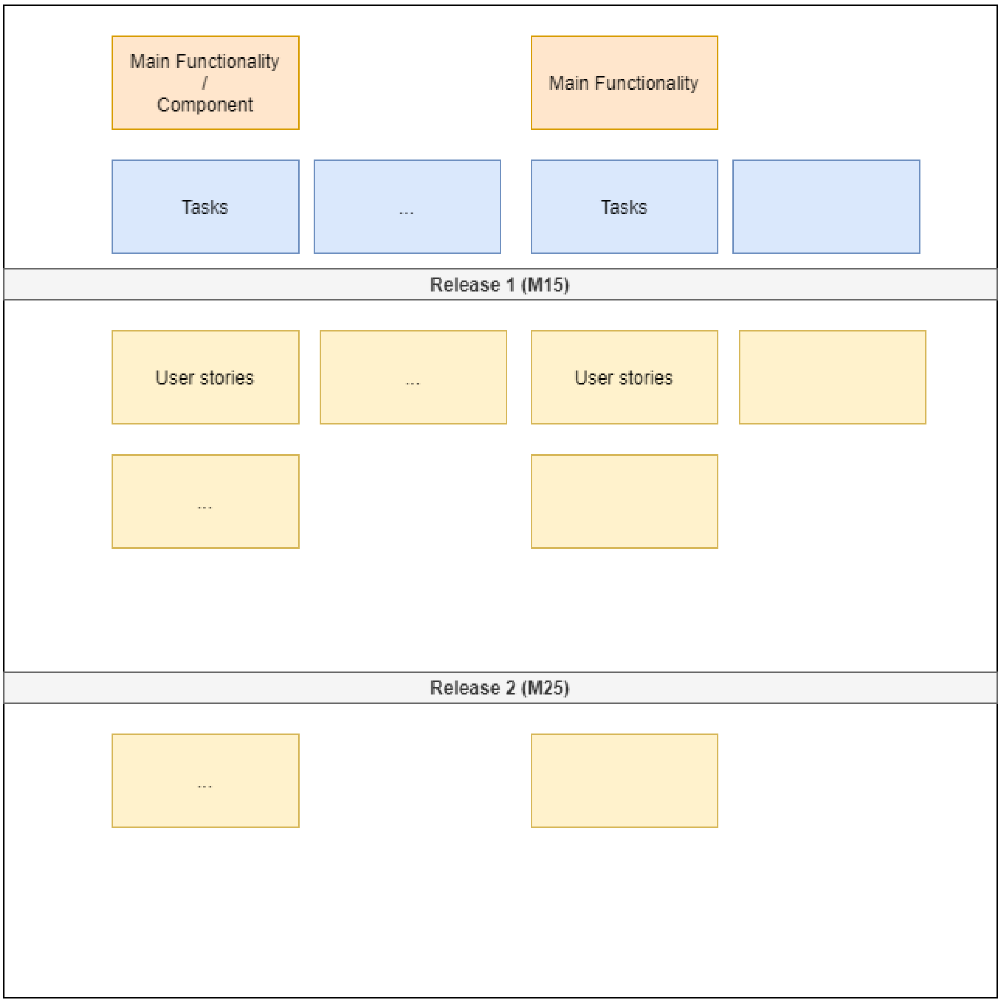


Figure 6 Behavior and functionalities artifact

This way, the functional specification of each component contains its story map, together with tables describing each user story.

As mentioned above, in the description of a feature, UML sequence diagrams are used to depict the interaction between the main classes and external components to the component under definition. Additionally, when a given functionality is initiated by a user, a UI mock-up has been provided so that a clear understating of the functionality is achieved. Thus, the functional specification of each component includes a subsection with the corresponding UI mock-ups and UML sequence diagrams.

In the next Information System Architecture, the interactions of the component are explained using a component interaction diagram and detailing the messages exchange through a UML class diagram.

3.1 DDDAS Component

3.1.1 Targeted behaviour and functionalities

The key role of the DDDAS is orchestrating data ingestion, the digital models, the simulation and optimisation environment as well as controlling the connection between the Digital Twin and the physical world environment.

More specifically, the main activities of the DDDAS component are the following:

- **Context entity management**: Manages the context entity registration, updates and data retrieval.
- **Model Management**: Manages the models from the model library.
- **Scenario Management**: Manages the registration of scenarios, their linked models and configuration of models.
- **Simulation Orchestration Engine Management**: Manages the configuration of simulation and the orchestration of the application packages.

An overview of activities, tasks and stories related to the DDDAS is shown in Figure 7.

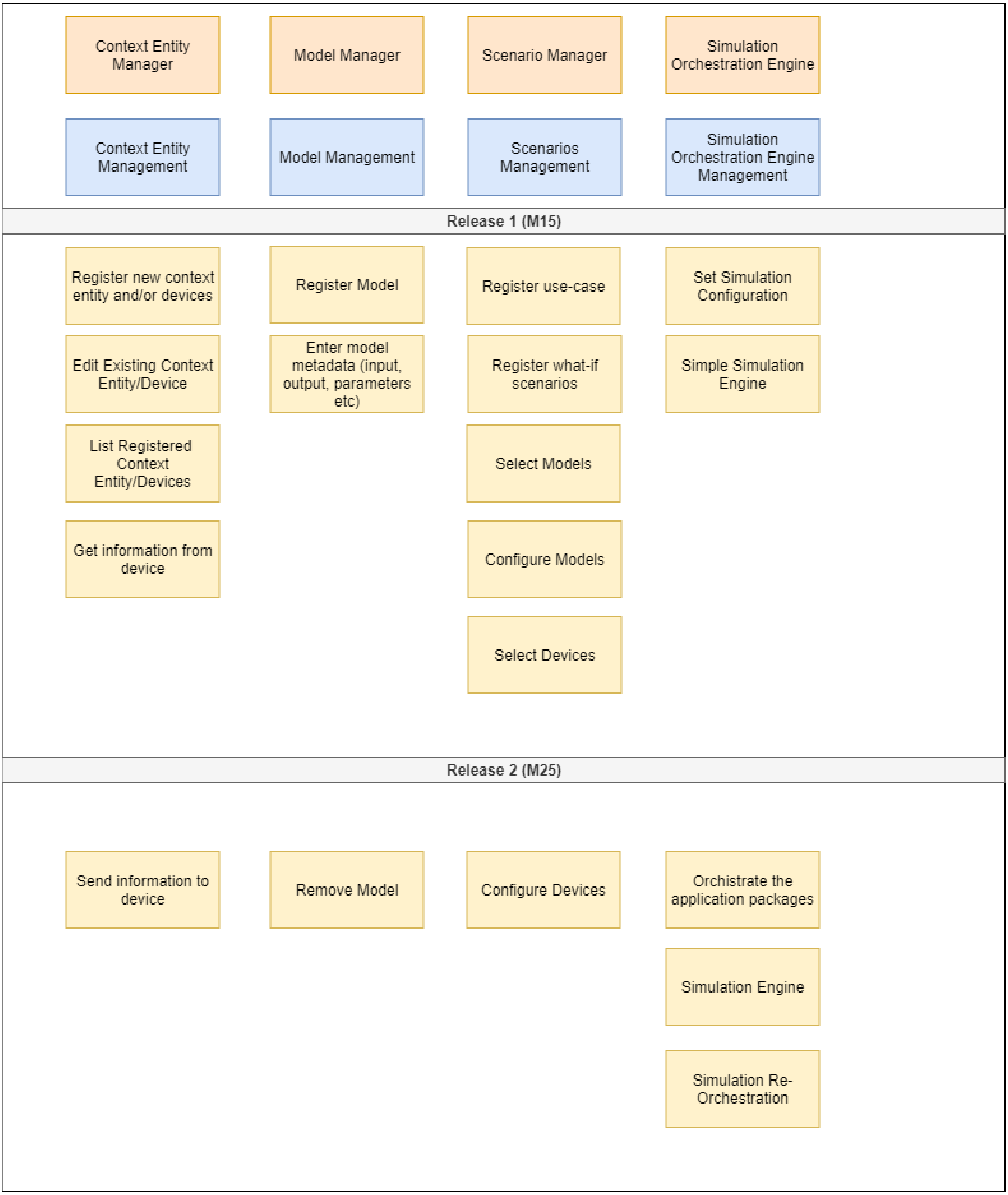

Context Entity Manager
Model Manager
Scenario Manager
Simulation Orchestration Engine
Context Entity Management
Model Management
Scenarios Management
Simulation Orchestration Engine Management
Release 1 (M15)
Register new context entity and/or devices
Register Model
Register use-case
Set Simulation Configuration
Edit Existing Context Entity/Device
Enter model metadata (input, output, parameters etc)
Register what-if scenarios
Simple Simulation Engine
List Registered Context Entity/Devices
Select Models
Get information from device
Configure Models
Select Devices
Release 2 (M25)
Send information to device
Remove Model
Configure Devices
Orchistrate the application packages
Simulation Engine
Simulation Re-Orchestration

Figure 7 DDDAS Activities, tasks and stories

3.1.2 User stories description

The textual description of each user story depicted in Figure 7 is as follows. The user privileges (admin, user) will be defined in the context of WP3 in collaboration with LL stakeholders.

Table 4 User Stories Description

User story	User story description
DDDASUS001 Register new context entity and/or devices	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Register a new context entity or device, including all the metadata and settings needed about this external entity **Why**: In order to retrieve contextual information from the endpoint of this external entity
	Acceptance Criteria: A new context entity with valid metadata and settings, after which it immediately be accessible to the LEAD components though its API to retrieve the exposed contextual data
DDDASUS002 Edit Existing Context Entity/Device	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Edit metadata of a registered device, including the deletion of this device **Why**: In order to provide the possibility to alter the information about a registered device and its settings
	Acceptance Criteria: After alteration of the settings of a device, the settings immediately become active. Deletion of a device is removed without checking the usage of this entity by another LEAD component
DDDASUS003 List Registered Context Entity/Devices	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Make a list of registered context entity/devices **Why**: In order to provide the possibility to list/see the registered context entity/devices.
	Acceptance Criteria:

	Registered Context Entity/Devices list is updated after adding and/or deleting a context entity/device.
DDDASUS004 **Get information from device**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Get information from the device. **Why**: In order to receive information from device.
	Acceptance Criteria: To get information from device, the information should be existing.
DDDASUS005 **Send information to device**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Send information to device. **Why**: In order to transfer information to device.
	Acceptance Criteria: To send information from device, the information should be existing and a request is required.
DDDASUS006 **Register Model**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Register a new model, including all the metadata (e.g. model parameters) and settings needed about this external entity. **Why**: In order to retrieve model information from the endpoint of this external entity.
	Acceptance Criteria: A new model with valid settings, after which it immediately be accessible to the LEAD components though DDDAS API to retrieve the exposed model data.
DDDASUS007 **Enter model metadata (input, output, parameters etc.)**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Enter model metadata (input, output, parameters etc) **Why**: In order to enter model metadata.
	Acceptance Criteria:

	Enter model metadata that have been given by Dashboard.
DDDASUS008 **Remove Model**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Remove a registered model **Why**: In order to remove/delete a registered model that is not used (anymore).
	Acceptance Criteria: After ensuring that the model is/will not being used by any current/future scenarios.
DDDASUS009 **Register use-case**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Register a use case. **Why**: In order to retrieve use-case information from the endpoint of this external entity.
	Acceptance Criteria: In order to simulate a use case, trying different models/scenarios, the first step is to register it in the DDDAS after being defined in the Dashboard.
DDDASUS0010 **Register what-if scenarios**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Register a use case. **Why**: In order to retrieve what-if scenarios information from the endpoint of this external entity.
	Acceptance Criteria: In order to simulate a what-if scenarios, trying different models, the first step is to register in the DDDAS after being defined in the Dashboard.
DDDASUS0011 **Select Models**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Select Models. **Why**: In order to select the appropriate model for each scenario.
	Acceptance Criteria: The selected models should be first registered.
DDDASUS0012 **Configure Models**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin

	What: Configure Models. **Why**: In order to configure models and adapt them for each user-case/scenario.
	Acceptance Criteria: After configuring a model, the system keeps the last configuration version.
DDDASUS0013 **Select Devices**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Select Devices. **Why**: In order to select the appropriate device(s) for each scenario.
	Acceptance Criteria: We select the appropriate device(s) for scenario if it is needed.
DDDASUS0014 **Configure Devices**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Configure device(s). **Why**: in order to configure device(s) and use them for each user-case/scenario.
	Acceptance Criteria: We configure a device if it is not used by any current simulation.
DDDASUS0015 **Set Simulation Configuration**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Set simulation configuration. **Why**: In order to set simulation configuration for each user case/scenario.
	Acceptance Criteria: We configure a simulation if a use-case/what-if scenario is registered
DDDASUS0016 **Simple Simulation Engine**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Set simulation configuration. **Why**: To start the simulation
	Acceptance Criteria: Simulation Engine is configured
DDDASUS0017 **Orchestrate the application packages**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Orchestrate the application packages. **Why**: In order to choose the appropriate application packages needed.
	Acceptance Criteria: Simulation engine is set and application packages registered

DDDASUS0018 **Simulation Engine**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Set simulate engine **Why**: in order to collect components, features and support functions that crucial to simulate model.
	Acceptance Criteria: Models are registered and simulation engine configured
DDDASUS0019 **Simulation Reorchestration**	**Description** **Who**: Admin of the Dashboard / DDDAS Admin **What**: Simulate reorchestration. **Why**: in order to orchestrate again application packages and rerun simulations.
	Acceptance Criteria: Models are registered and simulation engine re-configured

3.1.3 UI Mock-ups & Sequence diagrams

This sub-section shows sequence diagrams and UI mock-ups to clarify the stories sketched above and the LEAD internal interactions related to them.

3.1.3.1 Register new device, edit and list existing devices

Figure 9 shows the sequence diagram related to the registration of new device, edition of an existing device, and the retrieval of a list of existing devices, which primarily take place via the Dashboard.

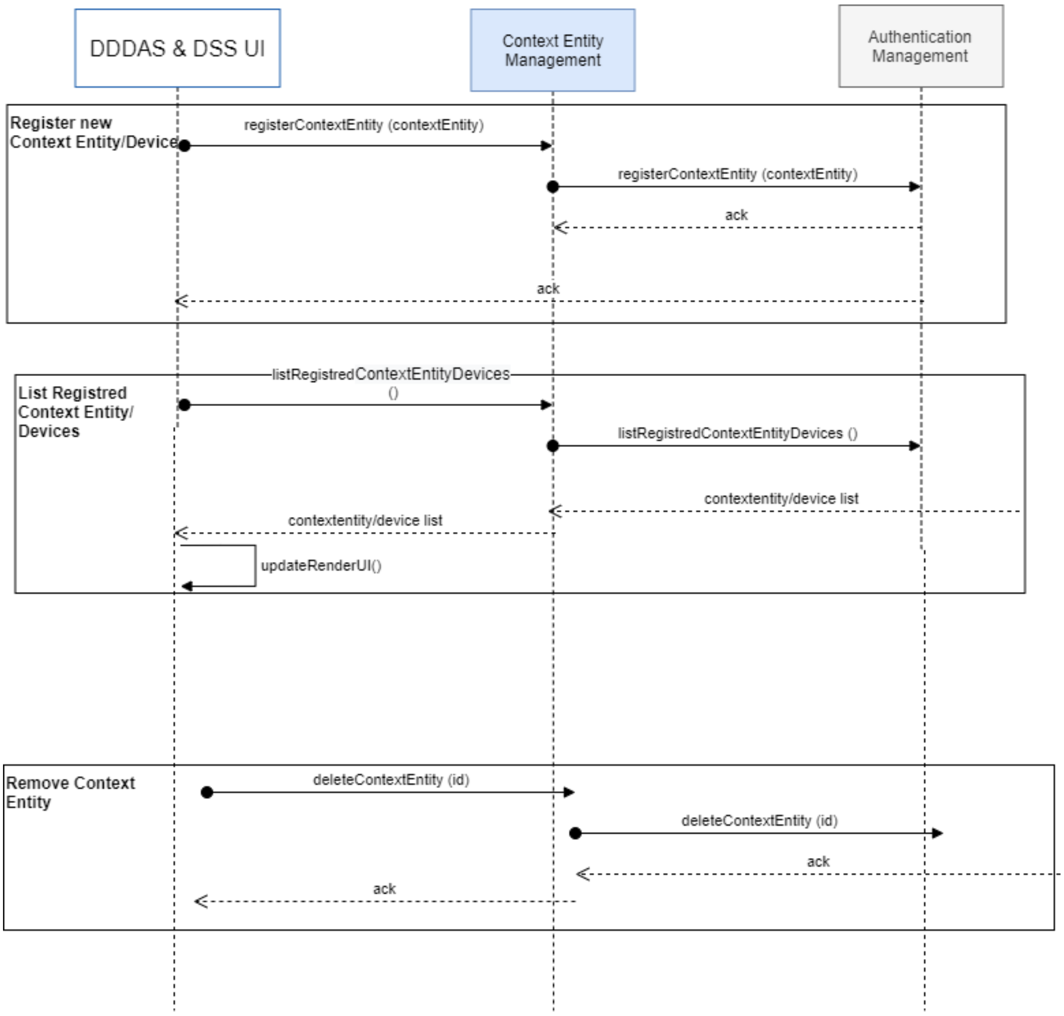

DDDAS & DSS UI
Context Entity Management
Authentication Management
Register new Context Entity/Device
registerContextEntity (contextEntity)
registerContextEntity (contextEntity)
ack
ack
List Registred Context Entity/ Devices
listRegistredContextEntityDevices ()
listRegistredContextEntityDevices ()
contextentity/device list
contextentity/device list
updateRenderUI()
Remove Context Entity
deleteContextEntity (id)
deleteContextEntity (id)
ack
ack

Figure 8 Register a new device Sequence Diagram

The main steps are:

- Register a new device (DDDASUS001)
- Edit a registered device (DDDASUS002)
- List registered devices (DDDASUS003)

The user interface for the registration of new device is shown as part of the Dashboard component. It is a basic registration form in which device name, URL, and params are requested. These details can easily be changed later by means of another interface.

3.1.3.2 Get and send information from/to device

Figure 10 shows the sequence diagram related to send information to device and get information form the device which take place in Context Entity Management.

The main steps are:

- Send information to device (DDDASUS005)
- Get information to devices (DDDASUS006)

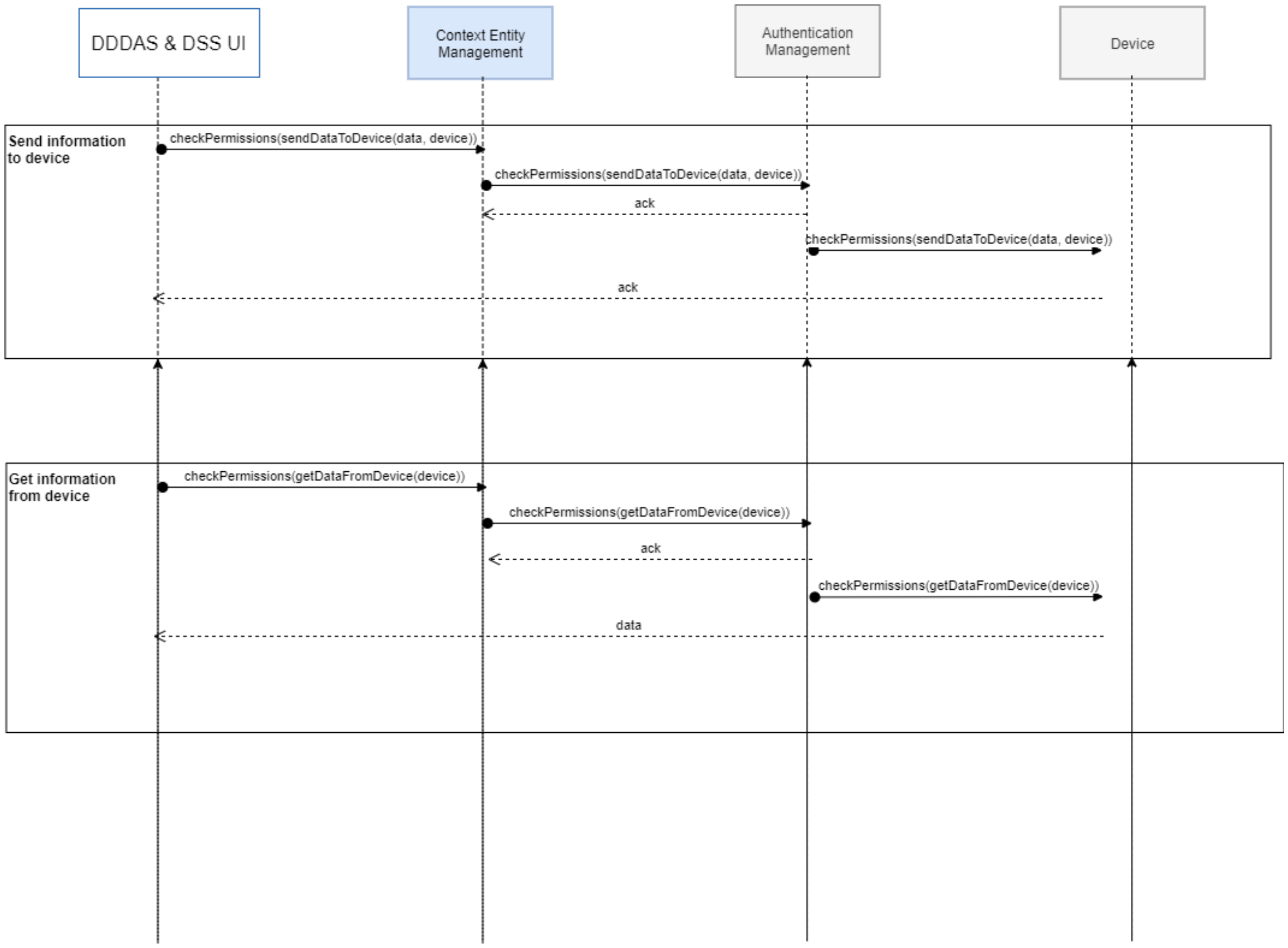

DDDAS & DSS UI
Context Entity Management
Authentication Management
Device
Send information to device
checkPermissions(sendDataToDevice(data, device))
checkPermissions(sendDataToDevice(data, device))
ack
checkPermissions(sendDataToDevice(data, device))
ack
Get information from device
checkPermissions(getDataFromDevice(device))
checkPermissions(getDataFromDevice(device))
ack
checkPermissions(getDataFromDevice(device))
data

Figure 9 Send/Get information from/to device Sequence Diagram

3.1.3.3 Scenario Management (register user case, register what-if scenarios, select model, select device and, configure device)

Figure 10 shows the sequence diagram related to register user case, register what-if scenarios, select model, select device and, configure device which primarily take place in Dashboard and then in Scenario Management.

The main steps are:

- Register user case (DDDASUS009)
- Register what-if scenarios (DDDASUS010)
- Select model (DDDASUS011)
- Select device (DDDASUS012)
- Configure device (DDDASUS013)

The user interface (see Figure 13) for the registration of user case, what-if scenarios are shown as part of the Dashboard component. It is a basic registration form in which user case, what-if scenarios, model and device names, and params are requested.

3.1.3.4 Simulation Configuration Management (set simulation configuration, orchestrate the application packages, simulate engine and simulate re-orchestration)

Figure 11 shows the sequence diagram related to set simulation configuration, orchestrate the application packages, simulate engine and simulate re-orchestration.

3.1.3.5 Model Management (register model, enter model metadata and remove model)

Figure 12 shows the sequence diagram related to register model, enter model metadata and remove model which primarily take place in Dashboard.

The main steps are:

- Register model (DDDASUS6)
- Enter model metadata (DDDASUS7)
- Remove model (DDDASUS8)

As depicted in Figure 13, the user interface for the model registration, enter model metadata, remove model are shown as part of the Dashboard component. It is a basic registration form in which model and model metadata are requested.

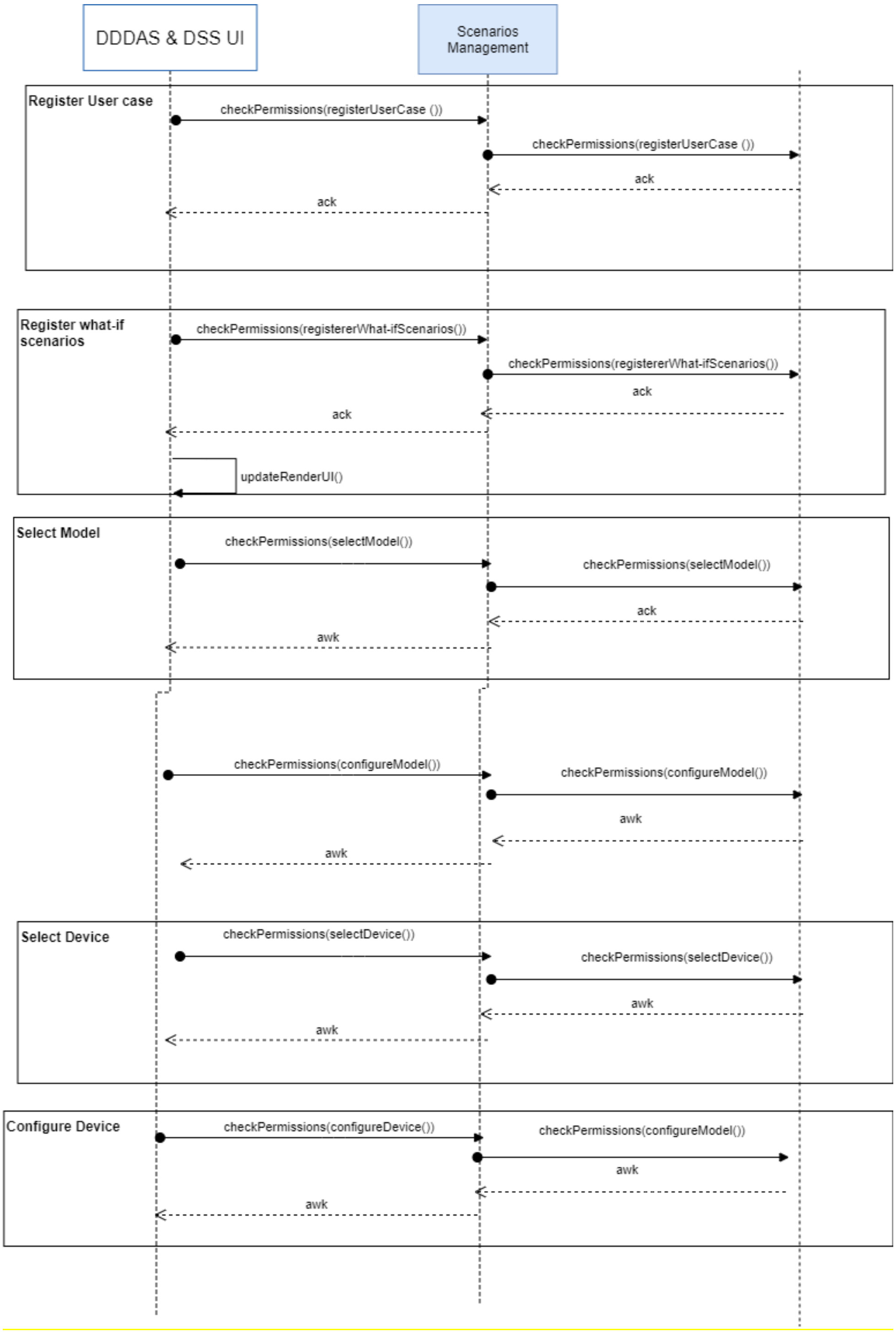

DDDAS & DSS UI
Scenarios Management
Register User case
checkPermissions(registerUserCase ())
checkPermissions(registerUserCase ())
ack
ack
Register what-if scenarios
checkPermissions(registererWhat-ifScenarios())
checkPermissions(registererWhat-ifScenarios())
ack
ack
updateRenderUI()
Select Model
checkPermissions(selectModel())
checkPermissions(selectModel())
ack
awk
checkPermissions(configureModel())
checkPermissions(configureModel())
awk
awk
Select Device
checkPermissions(selectDevice())
checkPermissions(selectDevice())
awk
awk
Configure Device
checkPermissions(configureDevice())
checkPermissions(configureModel())
awk
awk

Figure 10 Scenario Management Sequence Diagram

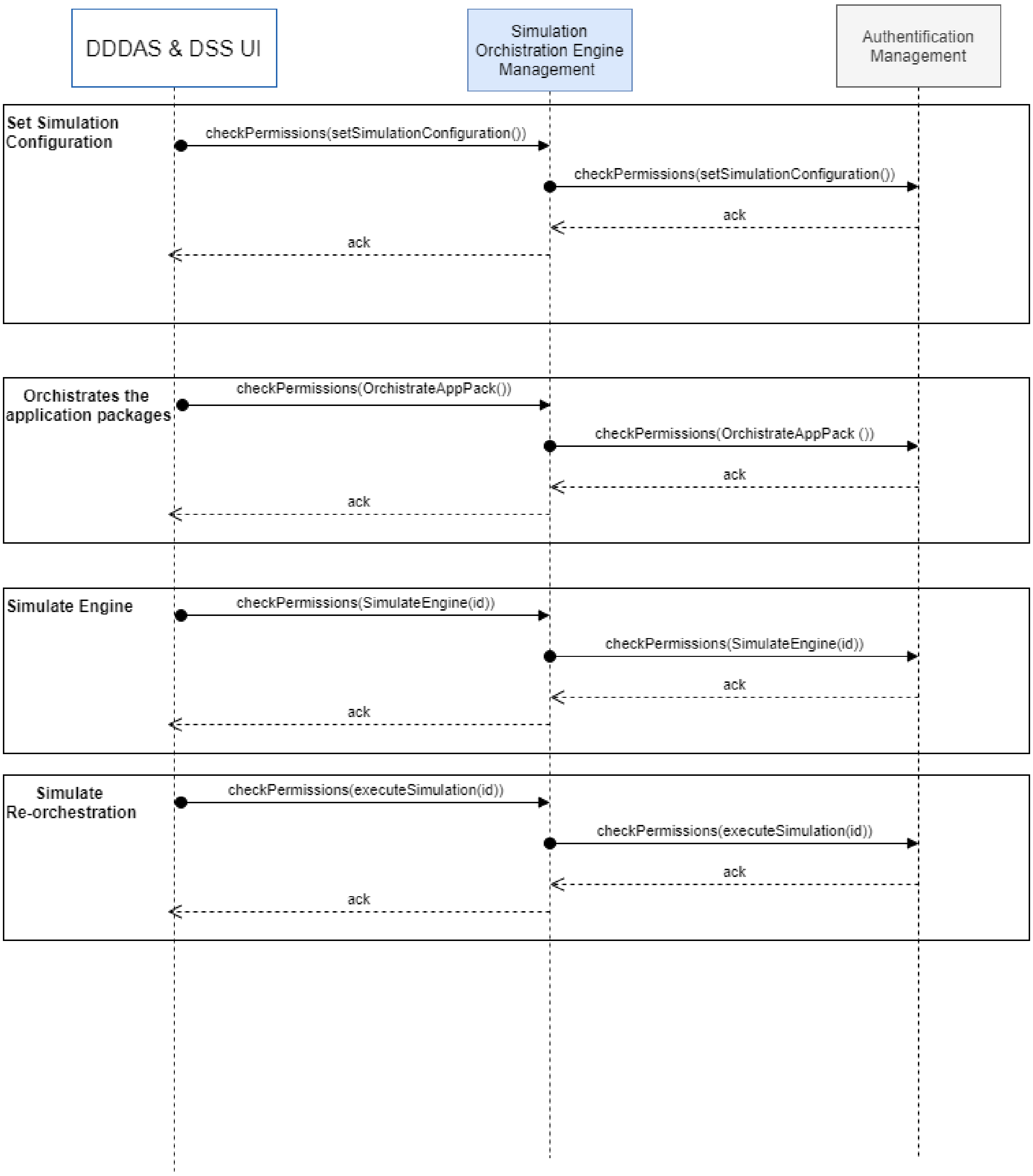

DDDAS & DSS UI
Simulation Orchistration Engine Management
Authentification Management
Set Simulation Configuration
checkPermissions(setSimulationConfiguration())
checkPermissions(setSimulationConfiguration())
ack
ack
Orchistrates the application packages
checkPermissions(OrchistrateAppPack())
checkPermissions(OrchistrateAppPack ())
ack
ack
Simulate Engine
checkPermissions(SimulateEngine(id))
checkPermissions(SimulateEngine(id))
ack
ack
Simulate Re-orchestration
checkPermissions(executeSimulation(id))
checkPermissions(executeSimulation(id))
ack
ack

Figure 11 Simulation Orchestration Sequence Diagram

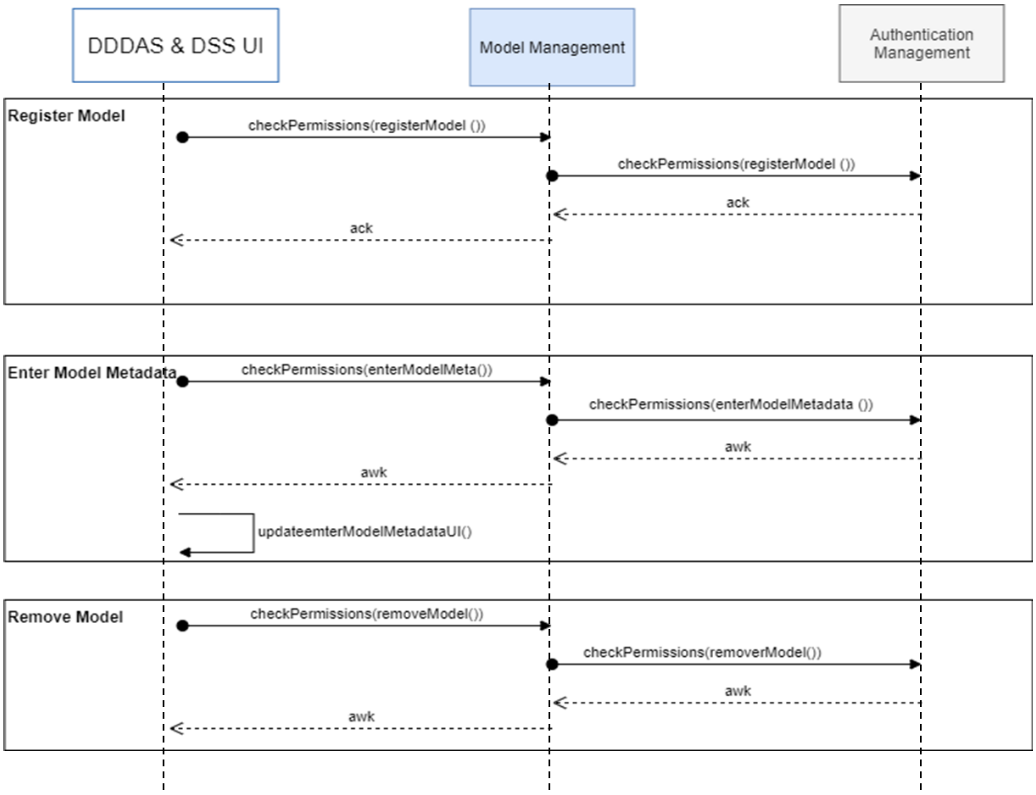
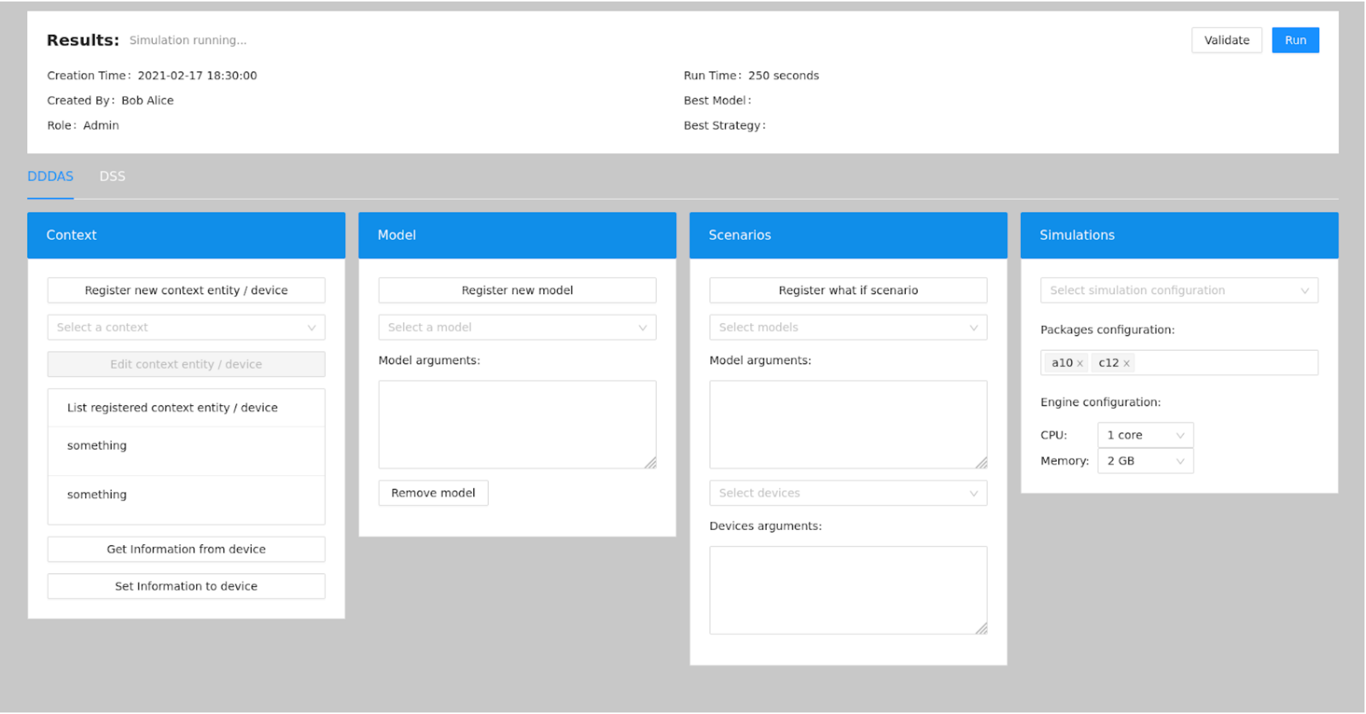

DDDAS & DSS UI
Model Management
Authentication Management
Register Model
checkPermissions(registerModel ())
checkPermissions(registerModel ())
ack
ack
Enter Model Metadata
checkPermissions(enterModelMeta())
checkPermissions(enterModelMetadata ())
awk
awk
updateemterModelMetadataUI()
Remove Model
checkPermissions(removeModel())
checkPermissions(removerModel())
awk
awk

Figure 12 Model Management Sequence Diagram



Results: Simulation running...

Creation Time : 2021-02-17 18:30:00

Created By : Bob Alice

Role : Admin

Run Time : 250 seconds

Best Model :

Best Strategy :

Packages configuration: a10 x c12 x

CPU: 1 core

Memory: 2 GB

Figure 13 DDDAS Mock-up

3.2 DSS Component

3.2.1 Targeted behaviour and functionalities

The main activities of the DSS component are the following:

- **Configuration System**: It defines the models, scenarios and devices parameters.
- **Decision System**: Select predictive models to make decisions regarding with the best scenarios/strategy. To choose the best scenarios among the successful ones that have been provided by DDDAS.
- **Decision Archive**: It is essentially a data base that keeps all the relative information regarding with the best scenario that has been chosen by Decision System.

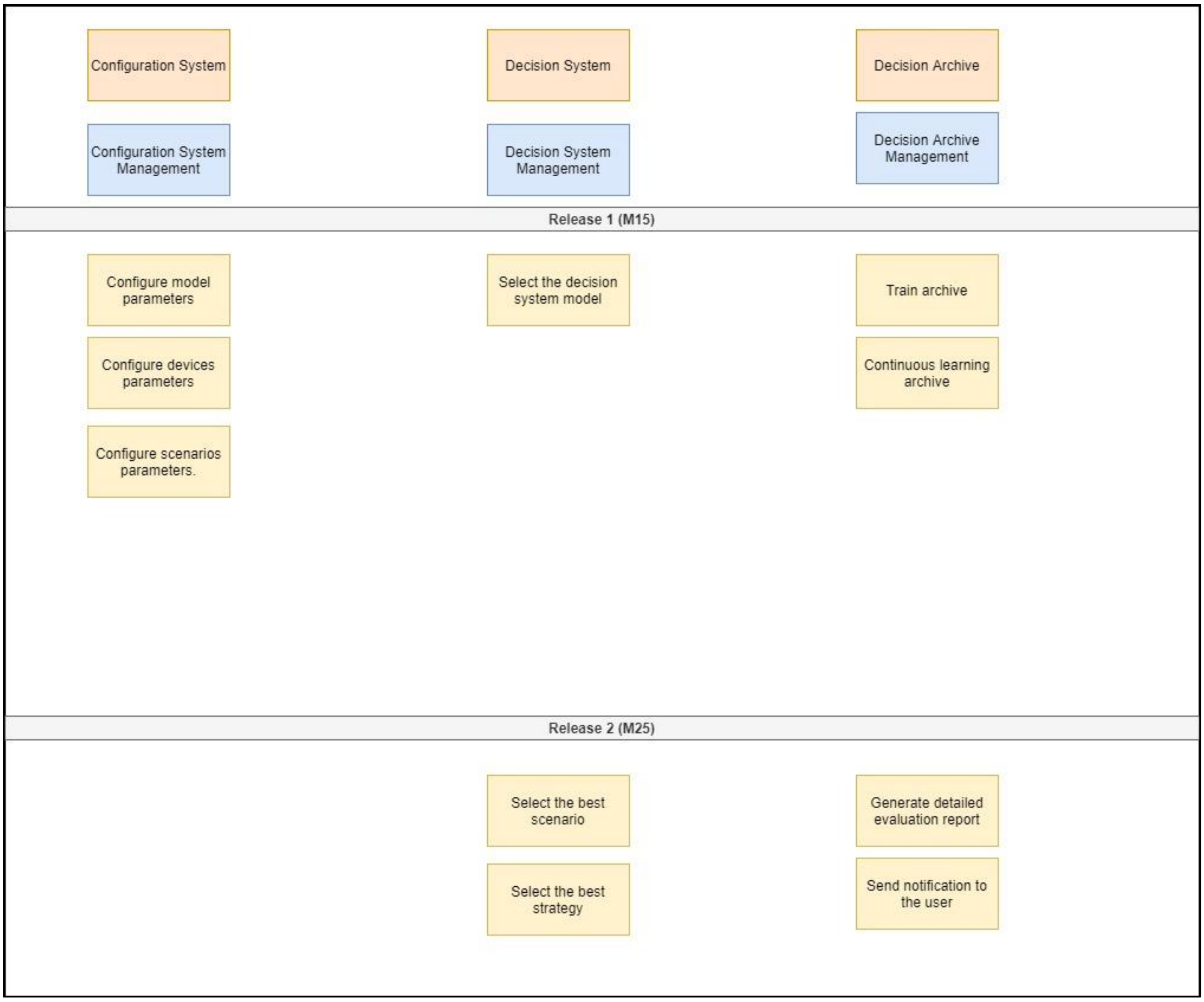

Configuration System
Decision System
Decision Archive
Configuration System Management
Decision System Management
Decision Archive Management
Release 1 (M15)
Configure model parameters
Select the decision system model
Train archive
Configure devices parameters
Continuous learning archive
Configure scenarios parameters.
Release 2 (M25)
Select the best scenario
Generate detailed evaluation report
Select the best strategy
Send notification to the user

Figure 14 DSS Activities, tasks and stories.

3.2.2 User stories description

The textual description of each user story depicted in Figure 14 is as follows:

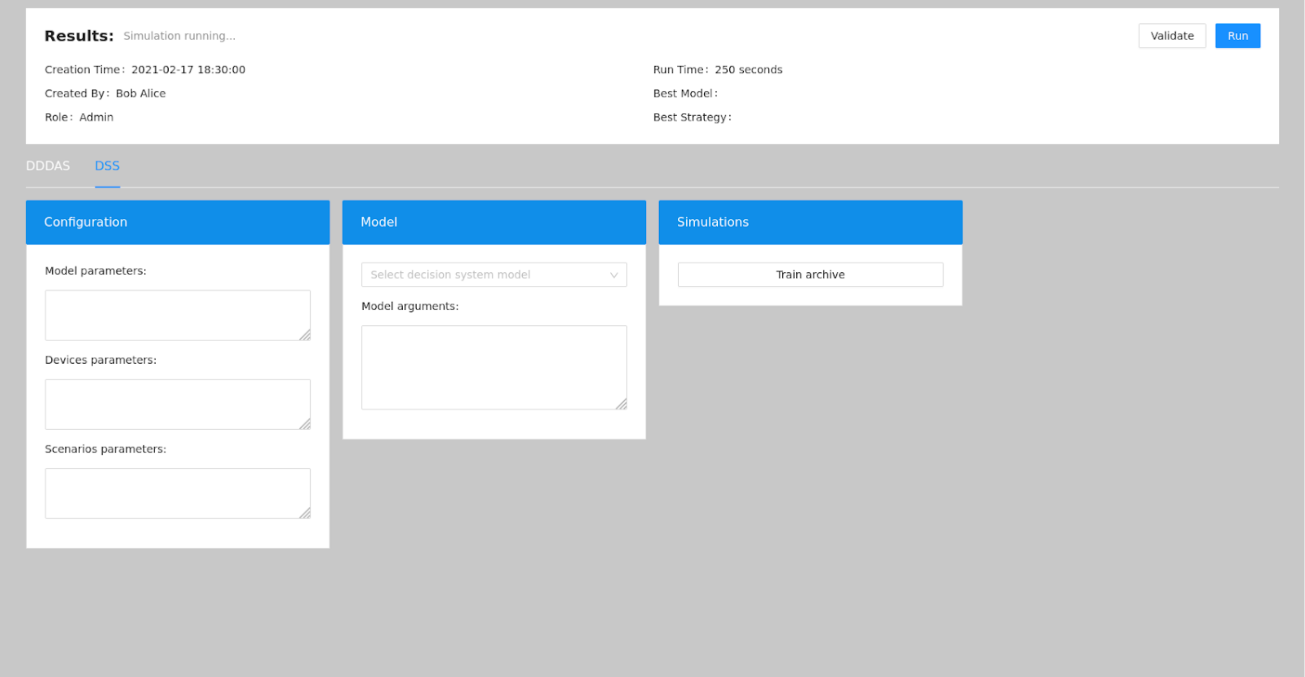
Table 5 User Stories Description

User story	User story description
DSSUS001 Configure model parameters	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Configure model parameters **Why**: in order to provide the appropriate model parameters to decision system given the expected simulation.
	Acceptance Criteria: Ability to define model parameters whose validity can be programmatically assessed by querying the model library
DSSUS002 Configure scenarios parameters	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Configure scenarios parameters **Why**: in order to provide the appropriate scenario parameters to decision system.
	Acceptance Criteria: Front end linked with backend successfully making API calls.
DSSUS003 Configure device parameters	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Configure device parameters **Why**: in order to provide the appropriate device parameters to decision system.
	Acceptance Criteria: Specify input thresholds that correspond to the input devices operational capacity.
DSSUS004 Select the decision system model	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Select the decision system model **Why**: in order to choose the best scenario/strategy

	Acceptance Criteria: Availability of DSS models on the platform.
DSSUS005 Select the best scenario	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Select the best scenario **Why**: in order to take the simulation result.
	Acceptance Criteria: Trade off analysis being successfully carried out and explained by the DSS.
DSSUS006 Select the best strategy	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Select the best strategy **Why**: in order to take the simulation result.
	Acceptance Criteria: Optimisation models available on the platform and model parameters definition (To be elaborated in Task T2.3 Decision Support System Interface and APIs)
DSSUS007 Train Archive	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Train Archive **Why**: in order to choose best scenario.
	Acceptance Criteria: KPIs defined and simulations being successfully carried out and explained by the DSS (To be elaborated in Task T2.3 Decision Support System Interface and APIs).
DSSUS008 **Continuous learning archive**	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Continuous learning archive **Why**: in order to choose the best scenario based on some KPIs.
	Acceptance Criteria: Baseline KPIs defined (To be elaborated in Task T2.3 Decision Support System Interface and APIs)
DSSUS0010 Generate detailed evaluation report	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Visualize the generated evaluation report after receiving notification of the execution of the selected scenario **Why**: To manage decisions and deal with changes

	Acceptance Criteria: Simulation results analysis compared to baseline KPIs (To be elaborated in Task T2.3 Decision Support System Interface and APIs)
DSSUS0011 Send notification to user	**Description** **Who**: Admin of the Dashboard / DSS Admin **What**: Send notification to user **Why**: in order to generate the detailed evaluation report.
	Acceptance Criteria: User registered and authorised (To be elaborated in Task T2.3 Decision Support System Interface and APIs)

The following diagram shows an early UI mock-up of the DSS that will be designed and developed in the context of Task T2.3 Decision Support System Interface and APIs. Deliverable D2.3 will include more detailed designs and user stories capturing the Living Labs requirements. The interface will provide tools for end users to configure the models, input parameters and run what-if scenario simulations. The overall solution will be scalable and adaptable to new requirements, able to serve new scenarios beyond the ones elaborated in the context of the LEAD project.



Results: Simulation running...

Creation Time: 2021-02-17 18:30:00
Created By: Bob Alice
Role: Admin

Run Time: 250 seconds
Best Model:
Best Strategy:

Figure 15 DSS UI early mock-up

4. Information System Architecture

This section focuses on the application and data architecture levels. Each LEAD component is described in detail with its internal structure and the interactions between internal or external LEAD components. First, an internal detailed description of sub-components and their interactions is realized based on the same legend as for the LEAD architecture diagram presented early in Figure 3. Then, a presentation of the main data entities that is needed on those components is presented depending on the level of details of each component (meta-model, or specific class diagram). Finally, an analysis of each component is realized to define solution building blocks (SBBs) that we are considering at this level of detail for the implementation of each component and its sub-components.

Each SBB will be evaluated in the form of '+++' and '---' to cover the functionality, parameters, and security. The more '+', the better a solution will be covering such aspects, while more '-', the less likely a solution is going to be selected for reuse. In terms of reusability, the tool evaluated are ensured to have an open-source and reuse/extend of source code open enough to allow the exploitation of such tool beyond LEAD.

From this, an extra column has been added showing the final decision of the LEAD consortium, as to whether the solution evaluated is re-used or not for developing the component. The values of this column should be interpreted as follows:

- **Reuse as is**: the SBB is taken as an initial
- **Partial**: some of the functionality from the SBB is taken as a basis, but the development of the component needs further manipulation
- **Research**: the SBB is not known enough and, thus, more research is needed on this SBB to make a final decision
- **Buy**: the SBB, or the functionality of the SBB, is bought into the project in the form of re-use or minimal development on top of it
- **Don't use**: the SBB is disregarded for the development of the component

The listed SBB solutions are collected with the objective to maximize the overlap with components' functionalities. For the mapping between the components and the SBBs, the following architecture principles have been followed:

- Whenever possible, each SBB should cover ALL the targeted component functionalities
- Whenever possible, each SBB should present reliable non-functional properties to be implemented within LEAD platform
- Whenever possible, each SBB should offer interoperability enablers to get access to all its functionalities

- Other principles that could be targeted could be: reuse, coherence, plug and play, etc.

Finally, after the evaluation of each SBB, a final statement will be provided in the future deliverables for each component (D2.3 and D2.4) justifying the final decision for the development of the respective LEAD component

4.1 DDDAS Component

4.1.1 Component Interactions description diagram

The following diagram (see also full view in Annex B) presents the main interactions of the DDDAS.

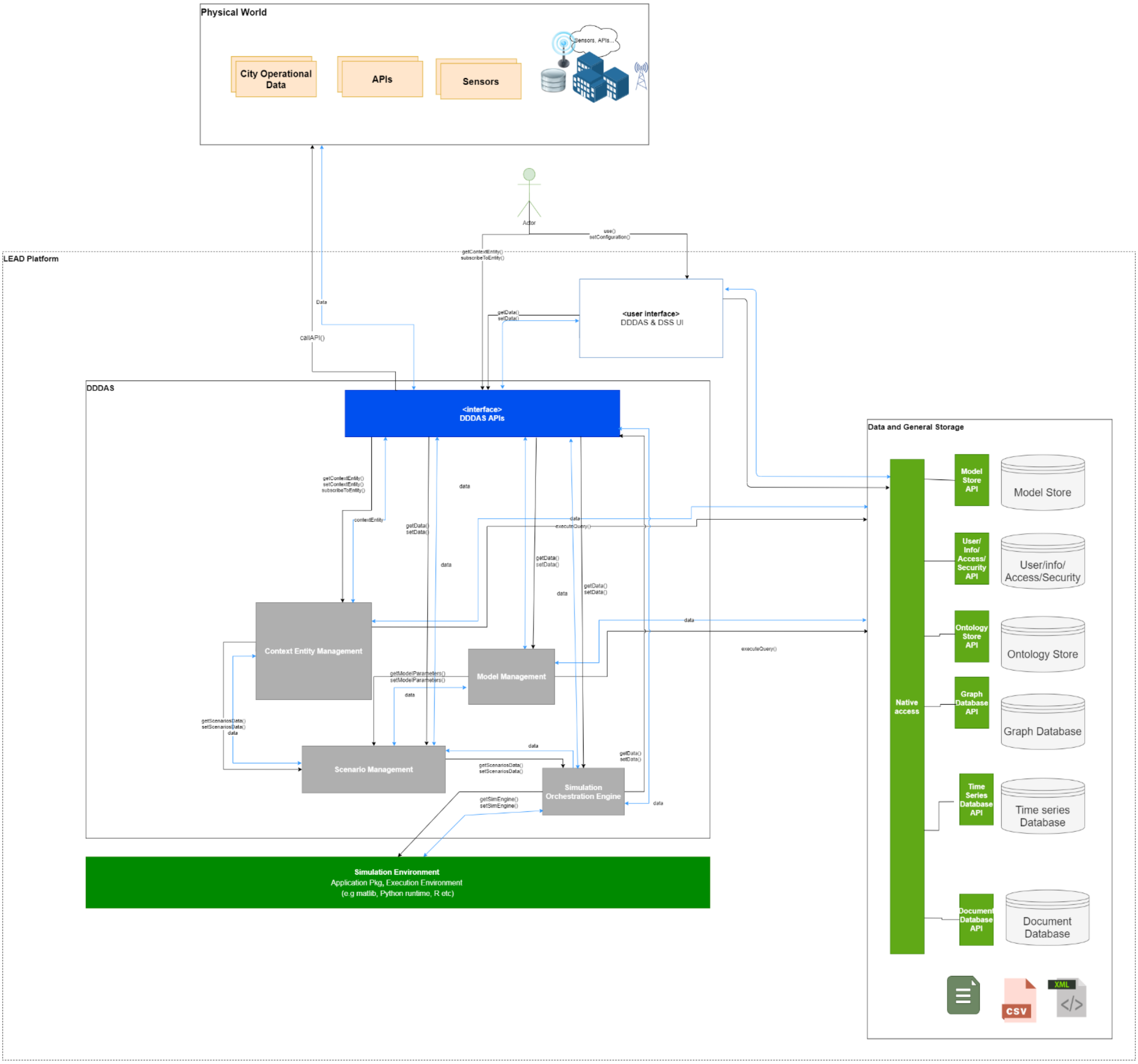


Figure 16 DDDAS Interaction Diagram

The DDDAS component consists of several sub-components, which are described in this sub-section. The connections or communications between sub-components will be performed through direct class calls. And the communications between internal sub-components (in gray colour) and the other components (green for internal or yellow colour for the external) is realized through the exposed Restful APIs.

As depicted in Figure 16, the DDDAS is composed of the following sub-components:

- **Context entity management**: Manages the context entity registration, updates and data retrieval.

- **Model Management**: Manages the models from the model library

- **Scenario Management**: Manages the registration of scenarios, their linked models and configuration of models

- **Simulation Orchestration Engine Management**: Manages the configuration of simulation and the orchestration of the application packages

- **DDDAS APIs:** Exposes the APIs to get access to the functionalities of the sub-components described earlier.

- **Simulation Environment:** This component is outside of the DDDAS; however, it is important to include it in this description since it has a strong link to the simulation orchestration engine. This component enables to execute the prepared simulation in the DDDAS and initialize a complete environment (such as virtual machine, libraries like python, R…) to execute the targeted simulation and retrieve the outputs.

4.1.2 Component Interactions definition

DDDAS sub-component have several interactions. These interactions are reflected in the form of a table with the following structure (columns):

- **Sub-Component:** This column defines the sub-component or the sub-function
- **Needs/Gives**: This column represents the following type of interactions:
  - **Gives**: The sub-component provides the sub-component defined in column “With” the data described in column “What”
  - **Needs**: The sub-component needs from the sub-component defined in column “With” the data described in column “What”
  - **Needs/Gives**: The “What” is exchanged on both directions

- **What**: This column describes the interaction or the exchanged data
- **With**: This column points at the sub-component(s) that interacts with the sub-component defined in column “Sub-Component”

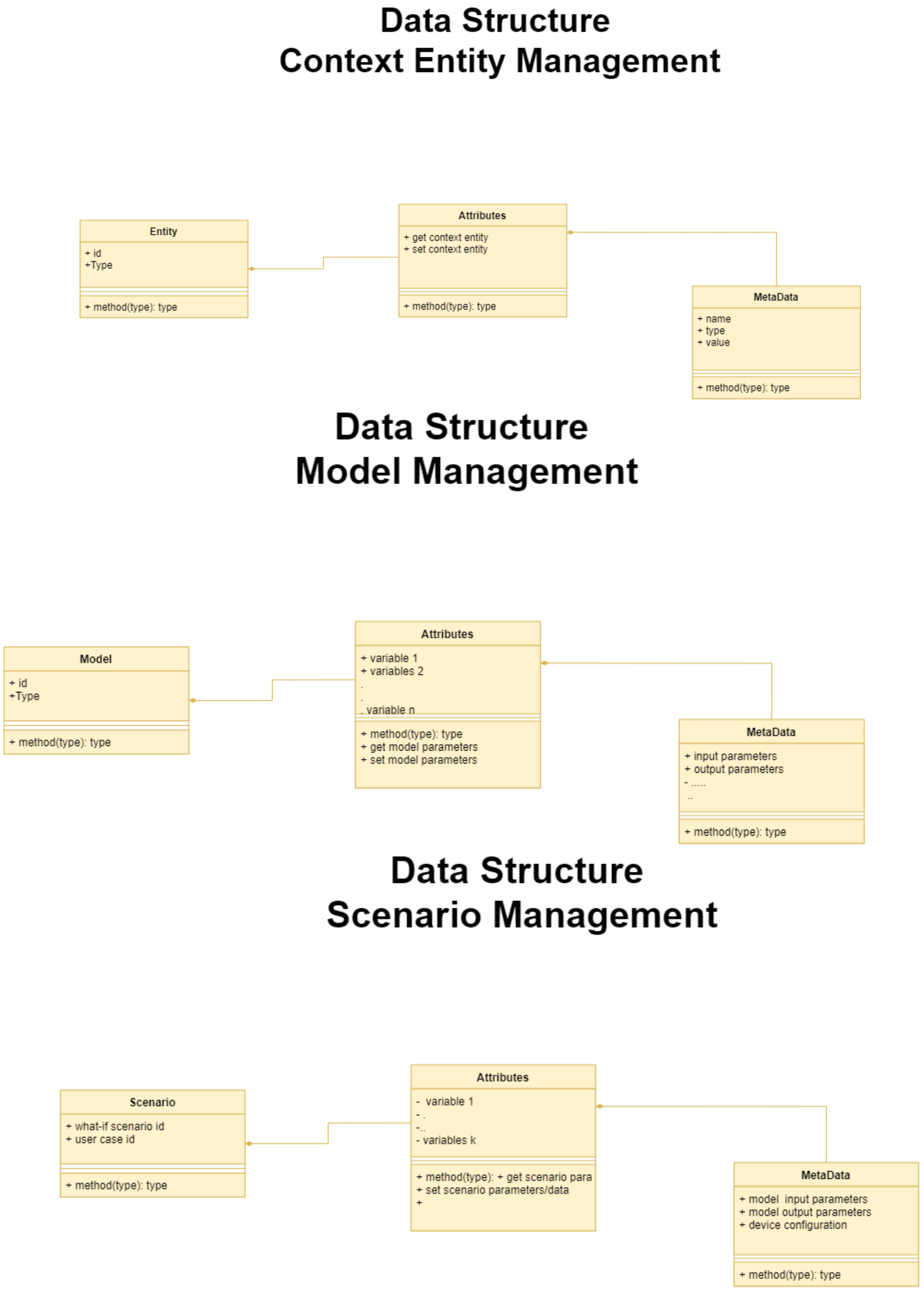
Table 6 DDDAS Interactions Description

Sub-Component	Needs/Gives	What	With
API	Needs	Sensor/DB data	Real World
API	Gives	Sensor/DB data	Context Entity Management
API	Gives	Sensor/DB data	Orchestration Engine
Context Entity Management	Needs	Sensor/DB data	API
Context Entity Management	Needs	Scenario Thresholds	Scenario Management
Context Entity Management	Gives	Sensor/DB data	Storage
Context Entity Management	Gives	Sensor/DB data	Scenario Management
Model Management	Needs/Gives	Data parameters/Model definitions	Simulation Management /Model Library
Model Management	Needs/Gives	Model definitions/Data parameters	Model Library/ Simulation Management
Scenarios Management	Gives	Scenario Data	Simulation Orchestration Engine
Scenarios Management	Needs	Sensor/DB data	Context Entity Management
Scenarios Management	Needs	Model Information	Model Library
Scenarios Management	Needs	User parameters	API (Dashboard)
Scenarios Management	Needs	Simulation status	Simulation Orchestration Engine
Scenarios Management	Gives	Simulation bundle	Simulation Orchestration Engine
Scenarios Management	Gives	Scenario thresholds	Context Entity Management

Scenarios Management	Gives	Simulation results	DSS

4.1.3 Component Classes and Information Exchanged

Figure 17 shows the associated data model for the described interactions in section 4.1.2. At this level of detail, we defined a meta-model for information exchanged with the context entity manager, the model manager and the scenario manager. In case of the context entity manager, the main class is the “Entity” class, that reflects the contextual data entities that are ingested from external APIs (e.g. sensor data). This data entity can have several attributes, augmented with more metadata (if needed), enabling to have a dynamic structure to fit different kind of contextual data entities. Regarding the model or scenario manager, the same concept is applied to fit to different kinds of models and scenarios and let the DDDAS sub-components generic enough to gather all the required data for different contexts. These class diagrams will be further detailed in the context of Tasks T2.3 Decision Support System Interface and APIs and T2.4 DDDAS (Looping Control) - Sensing - Data ingestion to define relationships between the components, the models and their metadata, considering also the work in T2.2 Digital Twin Models Library and specifically subtask ST2.2.3 LEAD Meta-model definition.



Data Structure
Context Entity Management

Data Structure
Model Management

Data Structure
Scenario Management

Figure 17 DDDAS Components Data Structures

4.1.4 Definition of Solution Building Blocks: Reuse vs Make vs Buy

Table 7 below analysis the DDDAS to identify solution building blocks (SBBs) that we are considering at this level of detail for the implementation of each DDDAS component and its sub-components. This evaluation is made based on the methodology described in the introduction of section 4.

The final decision about the choice of the solution building blocks will be presented and justified in D2.4 deliverable for the development of the DDDAS.

Table 7 DDDAS Solution Building Blocks Analysis

Component	Solution	Level (Generic ->Specific)	Functionality	Evaluation	Level
Context Entity Manager	Orion Context Broker	Generic	All	+++	Reuse as is
	Eclipse DITTO	Generic	All	+++	Partial
	Kafka	Specific	Data pipelines	++	Partial
DDDAS Dashboard	Angular JS	Generic	All	+++	Reuse as is
Model Manager	Python	Generic	All	++	Partial
Scenarios Management	Python	Generic	Programming Logic	++	Partial
Simulation Orchestration Engine	SLRUM	Specific	Workflow Orchestration	+++	Reuse as is
	Drop & Compute	Specific	Workflow Management	++	Partial
	Python	Generic	Programming Logic	++	Partial
	Docker, Kubernetes	Generic	Virtual Environment, Environment Configuration	+++	Reuse as is
	KVM	Specific	Virtual Environment	+++	Reuse as is
	Ansible	Specific	Environment Configuration	+++	Reuse as is

4.2 DSS Component

4.2.1 Component Interactions description diagram

The following diagram (see also full view in Annex B) presents the main interactions of the DSS.

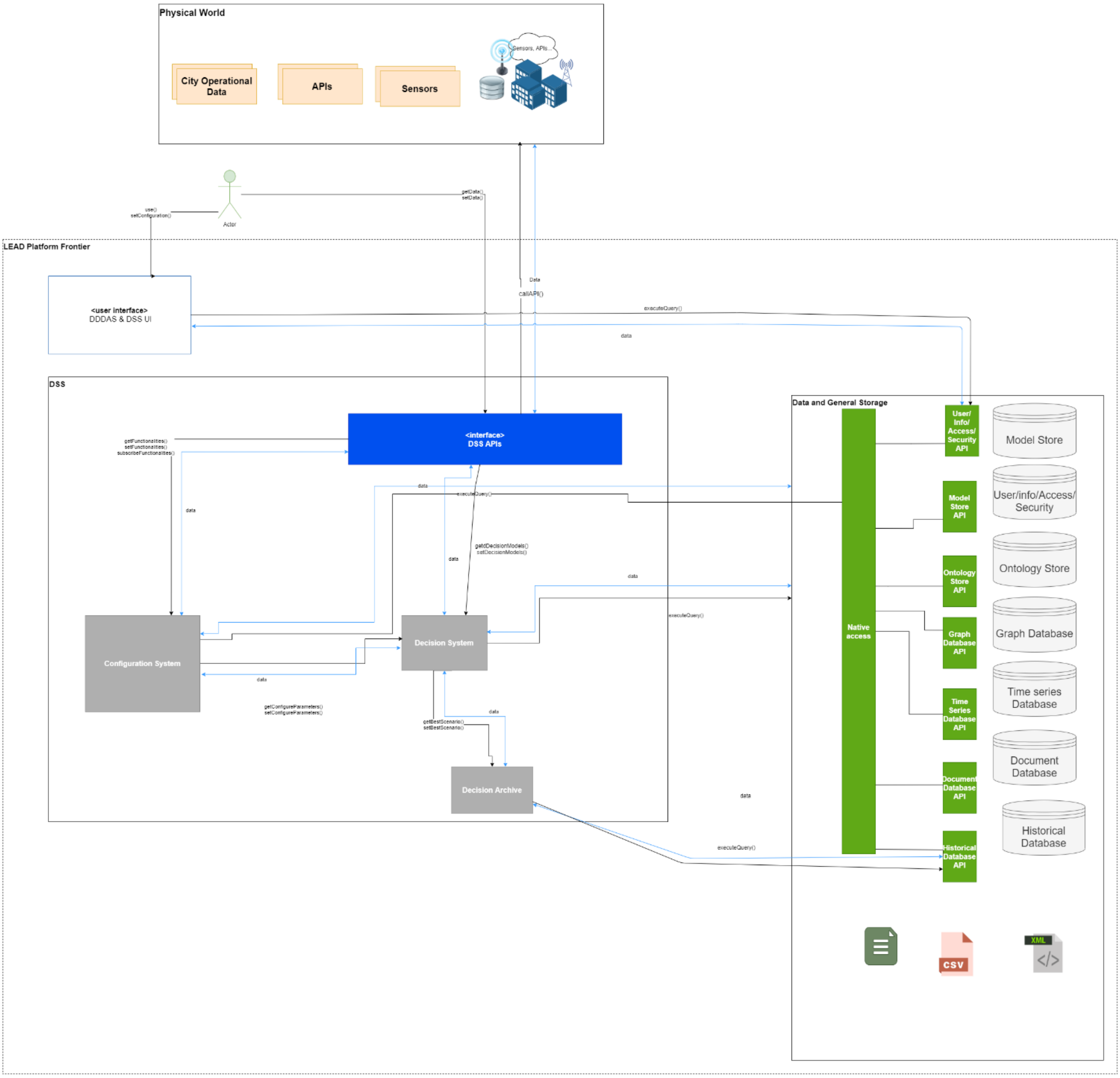


Figure 18 DSS interaction diagram

The DSS component consists of several functions, which are described in Section 4.2.2. The connections or communications between functions will be performed through direct class calls. And the communications between internal functions (in gray colour) and the other components (green for internal or yellow colour for the external) is realized through the exposed Restful APIs.

As depicted in Figure 13, the DSS is composed of the following sub-components:

- **Configuration System**: It configures model, scenarios and devices parameters.
- **Decision System**: Select predictive models to choose the best scenarios among the successful ones that have been provided by DDDAS.
- **Decision Archive**: It is essentially a data base that keeps all the relative information regarding with the best scenario that has been chosen by Decision System.
- **DSS APIs:** Exposes the APIs to get access to the functionalities of the sub-functions described earlier.

4.2.2 Component Classes and Information Exchanged

The figure below is indicative of the LEAD platform underlying data structure (entities, attributes and metadata definition). The data structure will be developed in the context of Task T2.4 DDDAS (Looping Control) - Sensing - Data ingestion.

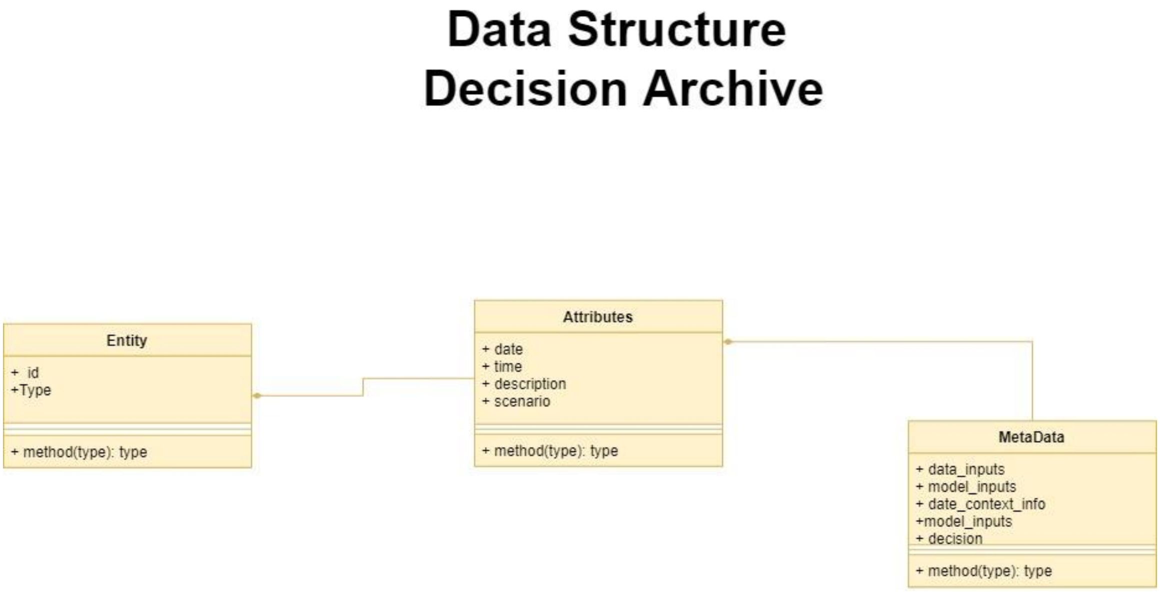


Figure 19 DSS Components Data Structures

4.2.3 Definition of Solution Building Blocks: Reuse vs Make vs Buy

Table 8 below analyses the DSS to identify solution building blocks (SBBs) that we are considering at this level of detail for the implementation of each DSS component and its sub-components. This evaluation is made based on the methodology described in section 4.

The final decision about the choice of the solution building blocks will be justified in D2.3 deliverable for the development of each component.

Table 8 DSS Solution Building Blocks Analysis

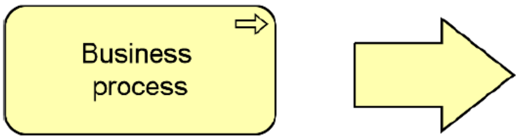
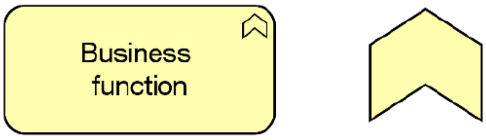
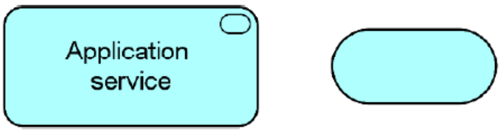
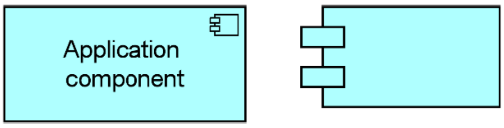
Component	Solution	Level (Generic ->Specific)	Functionality	Evaluation	Level
Configuration System	Python/JSON	Generic	All	++	Partial
Decision System	Python	Specific	Bayesian & Other Model	++	Partial
	R	Specific	Bayesian & Other Model	++	Partial
Decision Archive	MongoDB	Generic	Storage	+++	Reuse as is
DSS API's	Soap API	Generic	Communications	+++	Partial
	Swagger	Generic	Communications	+++	Partial
	Postman	Generic	Communications	+++	Partial
	Kafka	Generic	Communications	+++	Partial

Task 2.3 will elaborate the details of the DSS interactions with the user (choosing models and scenarios) and the dashboard will be co-designed with the Living Labs stakeholders.

4.3 LEAD usage viewpoint

This section describes the relationships between components in terms of the information flows between them or in terms of the services they offer, and how the identified SBBs are used to support the execution of the generic business process. Note that in this viewpoint, we focus only on the specific functionalities to execute the targeted generic processes. However, more functionalities are provided for the DDDAS and DSS to manage more actions in design time or run time such as the "continuous learning" or "Train archive". Please refer to section 3 for a detailed description of all provided functionalities.

Figures from (Figure 20 to Figure 23) illustrate the application usage in LEAD platform. These Figures are designed based on the ArchiMate 3 specification [2] as follows:

Design	Description
	A business process represents a sequence of business behaviours that achieves a specific result.
	A business function represents a collection of business behavior based on a chosen set of criteria (typically required business resources and/or competencies). There is a potential many-to-many relation between business processes and business functions.
	An application service represents an explicitly defined exposed application behaviour.
	An application component represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable.

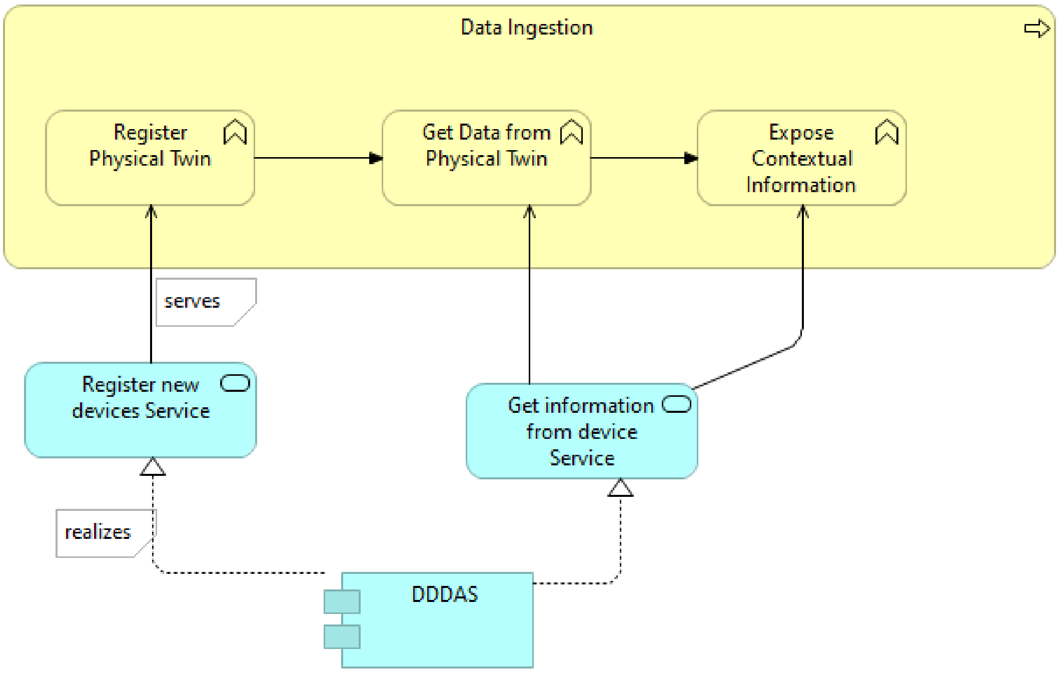

Data Ingestion
Register Physical Twin
Get Data from Physical Twin
Expose Contextual Information
serves
Register new devices Service
Get information from device Service
realizes
DDDAS

Figure 20 LEAD usage view point - Data ingestion

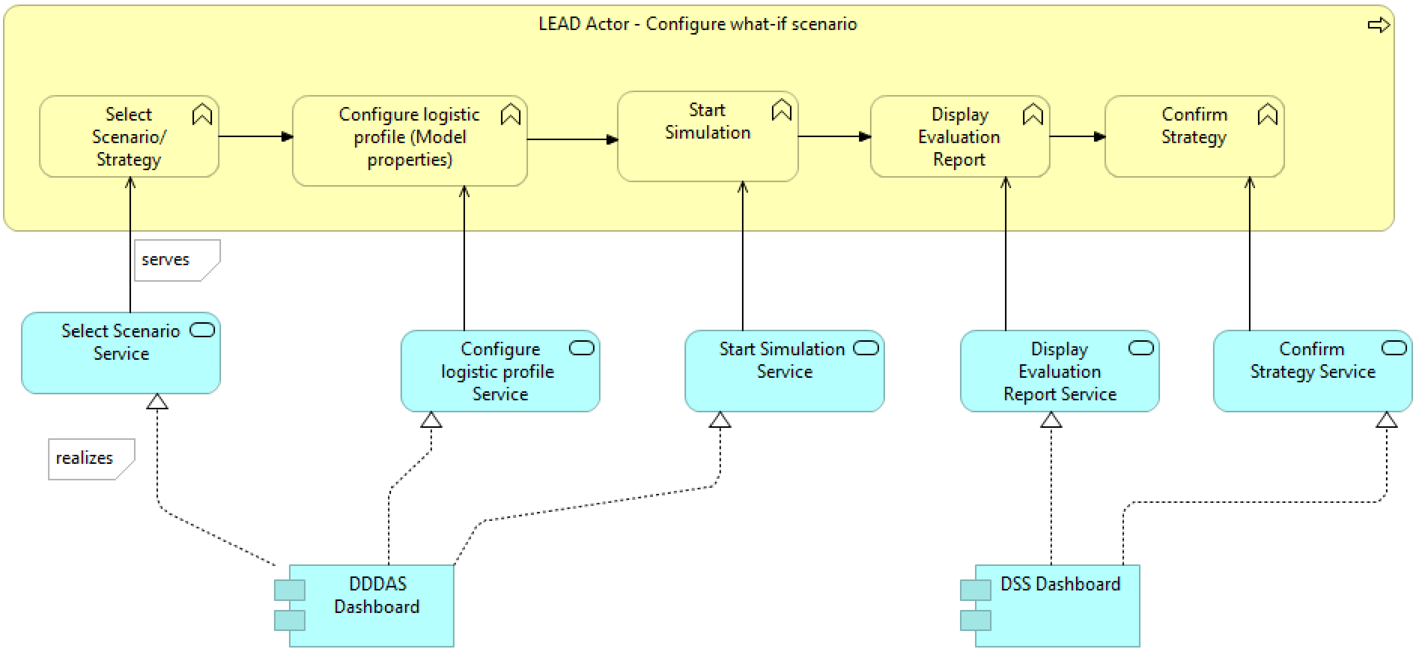

LEAD Actor - Configure what-if scenario
Select Scenario/ Strategy
Configure logistic profile (Model properties)
Start Simulation
Display Evaluation Report
Confirm Strategy
serves
Select Scenario Service
Configure logistic profile Service
Start Simulation Service
Display Evaluation Report Service
Confirm Strategy Service
realizes
DDDAS Dashboard
DSS Dashboard

Figure 21 LEAD usage view point - Configure what-if scenario

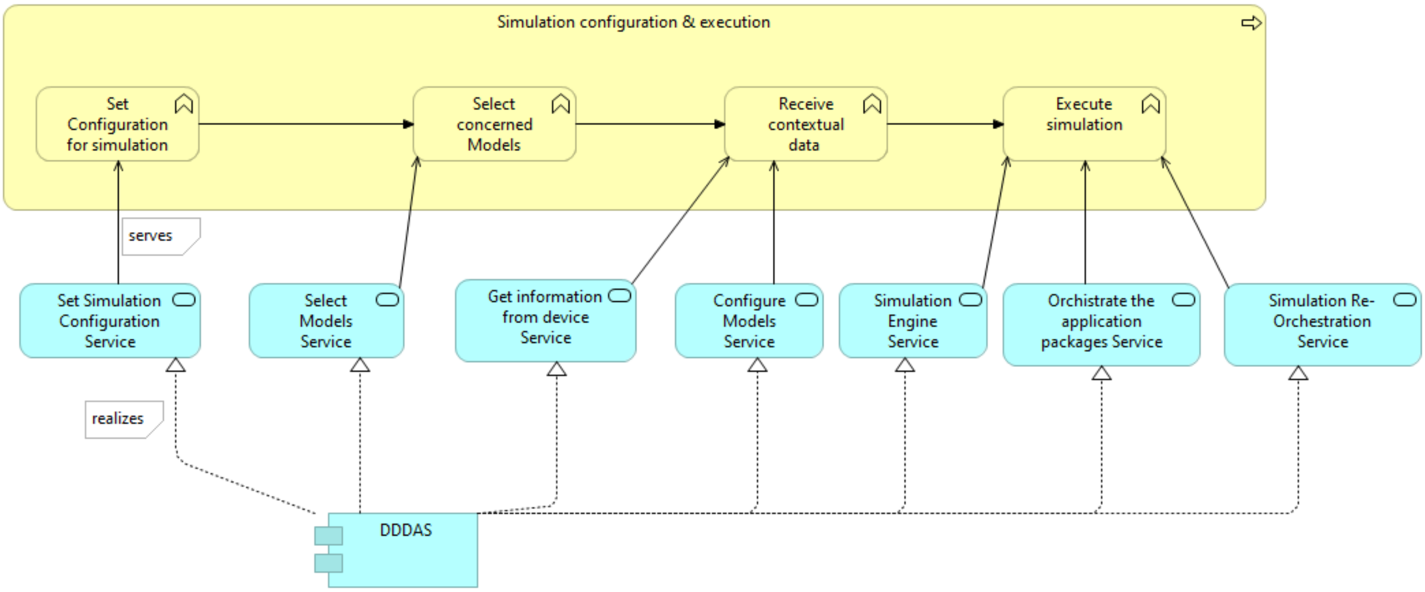

Simulation configuration & execution
Set Configuration for simulation
Select concerned Models
Receive contextual data
Execute simulation
serves
Set Simulation Configuration Service
Select Models Service
Get information from device Service
Configure Models Service
Simulation Engine Service
Orchistrate the application packages Service
Simulation Re-Orchestration Service
realizes
DDDAS

Figure 22 LEAD usage view point - Simulation configuration & execution

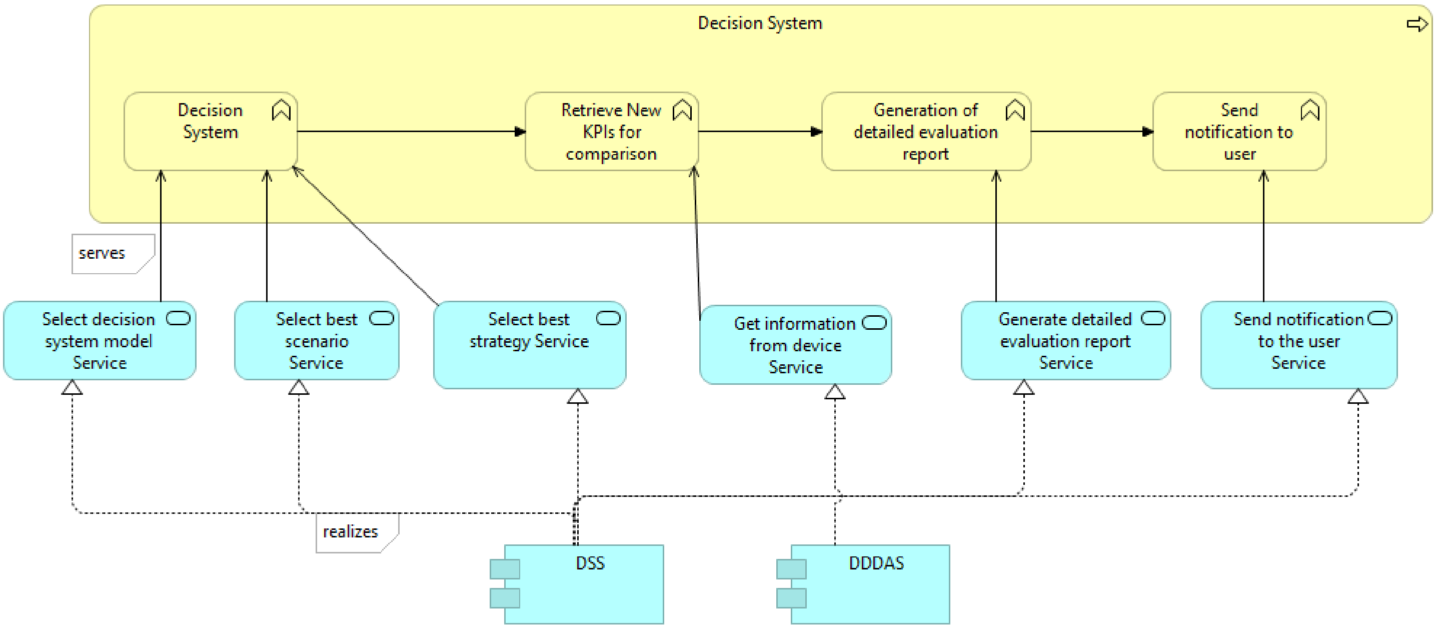

Decision System
Decision System
Retrieve New KPIs for comparison
Generation of detailed evaluation report
Send notification to user
serves
Select decision system model Service
Select best scenario Service
Select best strategy Service
Get information from device Service
Generate detailed evaluation report Service
Send notification to the user Service
realizes
DSS
DDDAS

Figure 23 LEAD usage view point - Data ingestion - Decision System

5. Technology Architecture

This last phase enables to define the communication endpoints of the identified components in the IS Architecture to be used during the implementation. The focus is on API definitions and common data models that are provided for LEAD inter-communication, as well as the security aspects and the infrastructure that supports the LEAD platform.

5.1 APIs Definition & Documentation

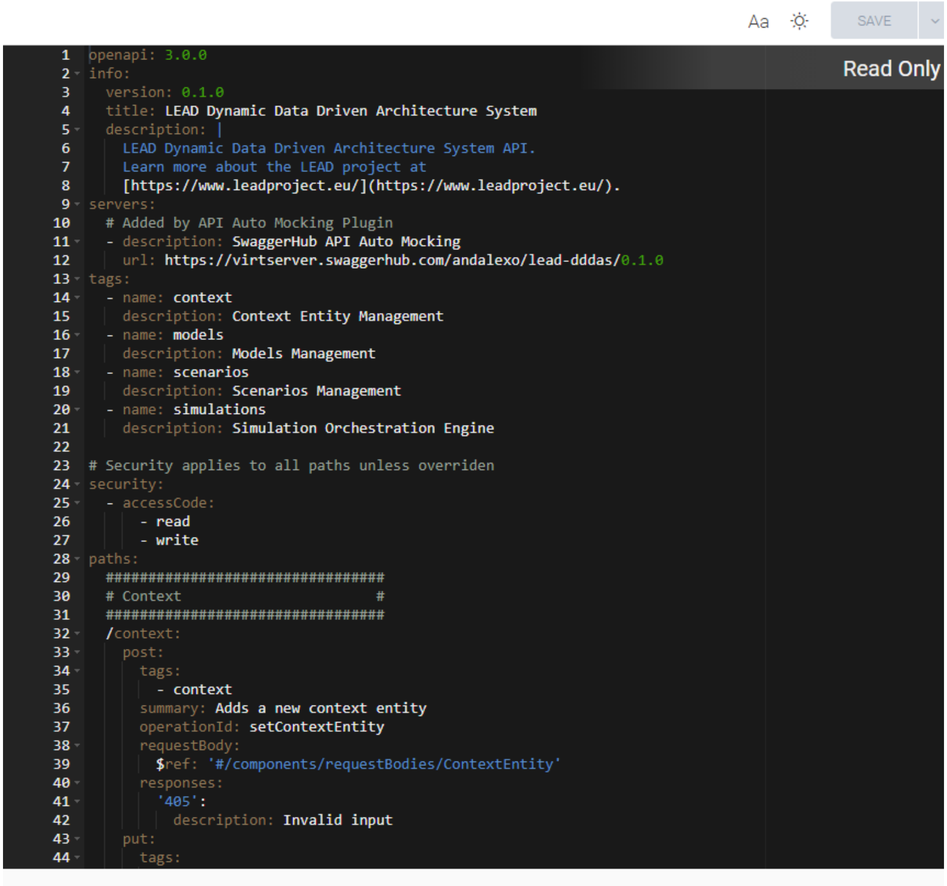
This section describes the technical specification and how the public interfaces will provide access to the functionality of the different LEAD components and Assets.

The focus is on interface definitions and common LEAD data models that are provided for LEAD inter-communication. As such, the deliverable mainly consists of online documentation, which is acting as a living document, and can be updated as soon as an endpoint changes. This living document allows only a little effort to communicate changes to all partners, developers and general public. This document itself represents an introduction to the on-line work.

OpenAPI Specification[16] emerged as an approach to building APIs and became one of the most popular frameworks providing a blueprint for API behaviour. OpenAPI Specification is the largest framework for designing APIs using a common language and enabling the development across the whole API lifecycle, including documentation, design, testing, and deployment. The framework provides a set of tools that help programmers generate client or server code and install self-generated documentation for web services.

For our purpose of documenting the exposed APIs, it has been decided to adopt Swagger Hub[17] that integrates the core Swagger tools (UI, Editor, Codegen, Validator) to document and share the APIs within LEAD partners and with the interested public. Each API is described using YAML[18] structure and is following the OpenAPI 3.0 specification[19] (see Figure 24 for an example).

[16] https://swagger.io/
[17] https://app.swaggerhub.com/s
[18] https://en.wikipedia.org/wiki/YAML
[19] https://swagger.io/docs/specification/about/



```yaml
openapi: 3.0.0
info:
  version: 0.1.0
  title: LEAD Dynamic Data Driven Architecture System
  description: |
    LEAD Dynamic Data Driven Architecture System API.
    Learn more about the LEAD project at
    [https://www.leadproject.eu/](https://www.leadproject.eu/).
servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/andalexo/lead-dddas/0.1.0
tags:
  - name: context
    description: Context Entity Management
  - name: models
    description: Models Management
  - name: scenarios
    description: Scenarios Management
  - name: simulations
    description: Simulation Orchestration Engine

# Security applies to all paths unless overriden
security:
  - accessCode:
      - read
      - write
paths:
  #################################
  # Context                       #
  #################################
  /context:
    post:
      tags:
        - context
      summary: Adds a new context entity
      operationId: setContextEntity
      requestBody:
        $ref: '#/components/requestBodies/ContextEntity'
      responses:
        '405':
          description: Invalid input
    put:
      tags:
```

Figure 24 OpenAPI specification example

5.2 Online Document Reference

The API documentation for each component is referenced via a web link shown in the table below. The complete online documentation is accessible at https://app.swaggerhub.com/search?query=leadproject.eu and will be updated continuously.

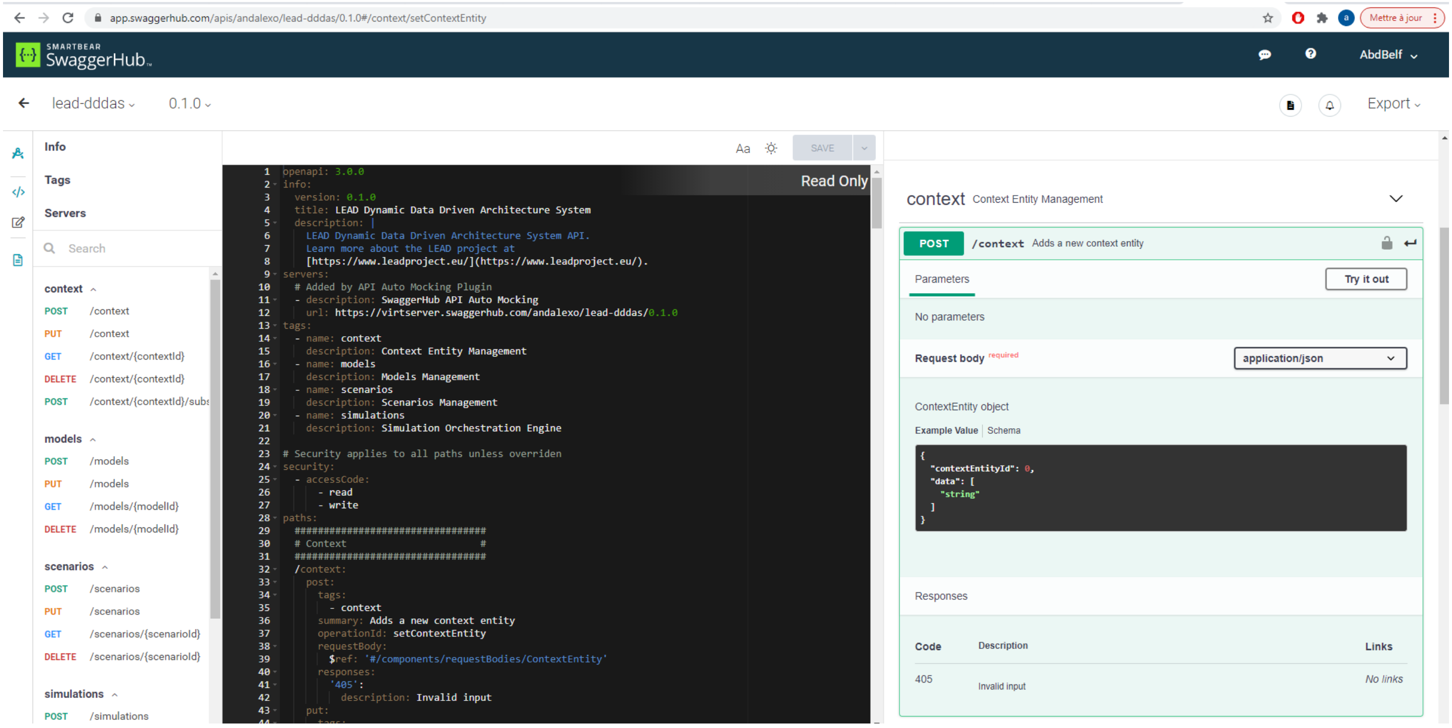
This online documentation is quite comprehensive and should be fully explored. An example for one part, of one component, is described in this section for illustrative purposes. We provide in Annex A an offline copy of the current description of each API.

Table 9 APIs Description - Online Documentation

Component	Reference
DDDAS	
Context Entity Manager	https://app.swaggerhub.com/apis/andalexo/lead-dddas/0.1.0#/context

Models Manager	https://app.swaggerhub.com/apis/andalexo/lead-dddas/0.1.0#/models
Scenario Manager	https://app.swaggerhub.com/apis/andalexo/lead-dddas/0.1.0#/scenarios
Simulation Orchestration Manager	https://app.swaggerhub.com/apis/andalexo/lead-dddas/0.1.0#/simulations
DSS	
Configuration System	https://app.swaggerhub.com/apis/andalexo/lead-dss/0.1.0#/config
Decision System	https://app.swaggerhub.com/apis/andalexo/lead-dss/0.1.0#/decision-system
Decision Archive	https://app.swaggerhub.com/apis/andalexo/lead-dss/0.1.0#/decision-archive

The documentation structure consists of 3 columns as depicted in Figure 25. The left column lists the entire functions of each component. The middle column describes the operations, requests, and security information. The right column presents a user interface to read the description in a visual manner, including a possibility to test the APIs.



```yaml
openapi: 3.0.0
info:
  version: 0.1.0
  title: LEAD Dynamic Data Driven Architecture System
  description: |
    LEAD Dynamic Data Driven Architecture System API.
    Learn more about the LEAD project at
    [https://www.leadproject.eu/](https://www.leadproject.eu/).
servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/andalexo/lead-dddas/0.1.0
tags:
  - name: context
    description: Context Entity Management
  - name: models
    description: Models Management
  - name: scenarios
    description: Scenarios Management
  - name: simulations
    description: Simulation Orchestration Engine

# Security applies to all paths unless overriden
security:
  - accessCode:
      - read
      - write
paths:
  #################################
  # Context                       #
  #################################
  /context:
    post:
      tags:
        - context
      summary: Adds a new context entity
      operationId: setContextEntity
      requestBody:
        $ref: '#/components/requestBodies/ContextEntity'
      responses:
        '405':
          description: Invalid input
    put:
```

context Context Entity Management

POST /context Adds a new context entity

Parameters

No parameters

Request body required application/json

ContextEntity object

Example Value | Schema

```json
{
  "contextEntityId": 0,
  "data": [
    "string"
  ]
}
```

Responses

Code	Description	Links
405	Invalid input	No links

Figure 25 API Description - Component overview

Each API endpoint is presented with all required and optional parameters inside the URL and also the http body (Figure 26). The parameters tab lists all parameters and provides information regarding the parameter type, object type, and a description of purpose of each parameter. In this documentation, three kinds of parameters types are used:

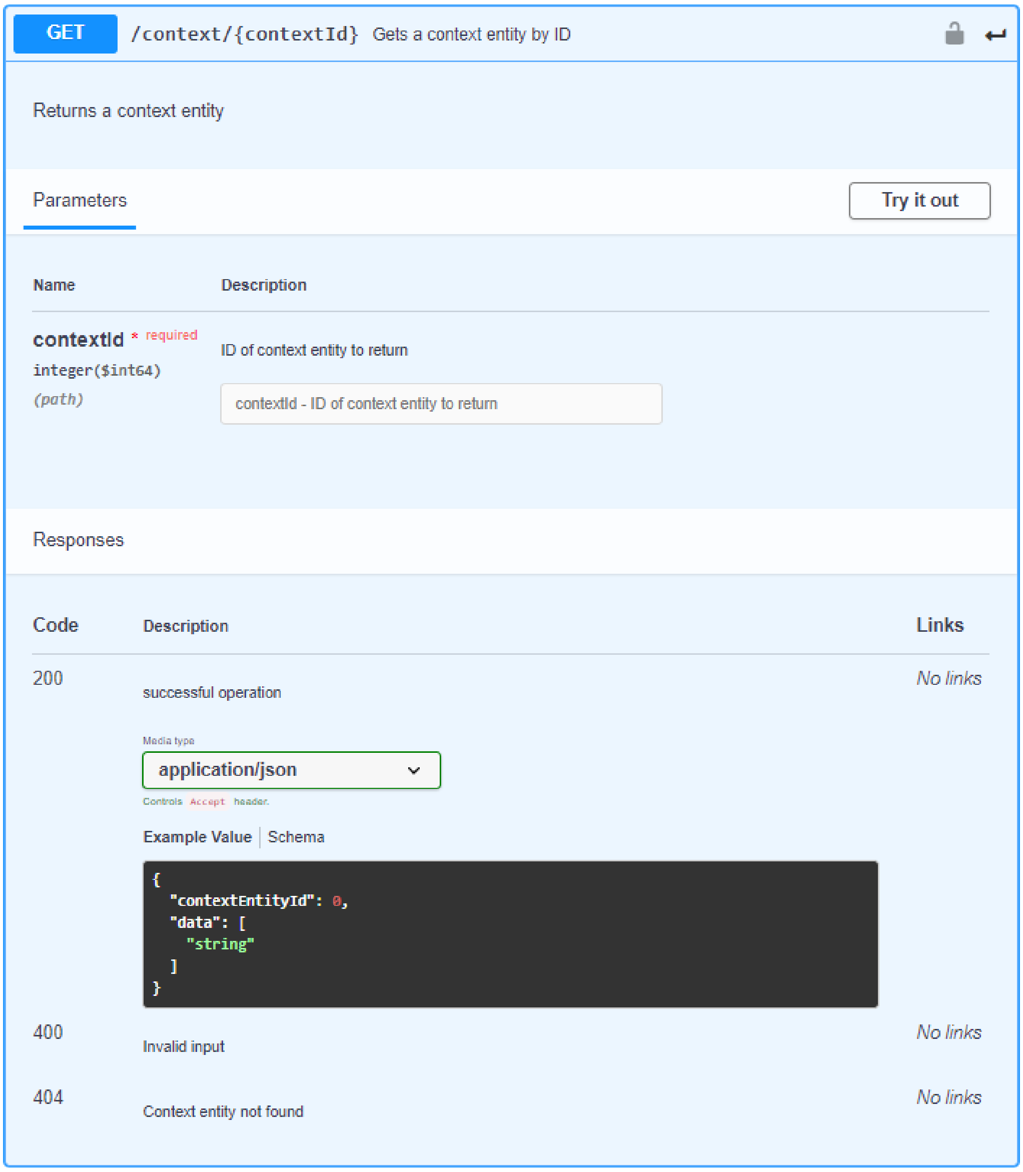
- **Resource parameters**, that are used to access specific resources via unique identifiers in order to retrieve information (GET), update properties of an object or to remove a resource, e.g. /v1/context/1234 where 1234 identifies the context entity

- **Query parameters** that enable the extension of a request with a string. This string contains named parameters that can then be evaluated by a Web application, e.g. /v1/assets?type=device, where a list is retrieved with only assets of type device.

- **Body content**, that contains a JSON object or a list of JSON objects that can be evaluated by a web application. Body contents are normally only contained in requests that create (POST) or edit (PUT) a resource.

The object type, on the other hand, describes the expected value of a parameter. This object type must be adhered to or the query will not be executed correctly and an error will be returned.

Finally, the response describes the returned result from the request. The response always returns an http status code to indicate the acknowledgment. In the case of the example (Figure 26), a representation of an asset in JSON format is returned combined with an http status code 200 (OK), 400 (invalid input) or 404 (resource not found). A complete list of all standardized http status codes can be seen at https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.



GET `/context/{contextId}` Gets a context entity by ID

Returns a context entity

Parameters

Name	Description
contextId * required integer($int64) (path)	ID of context entity to return contextId - ID of context entity to return

Responses

Code	Description	Links
200	successful operation Media type application/json Controls `Accept` header. Example Value | Schema	No links
400	Invalid input	No links
404	Context entity not found	No links

```
{
  "contextEntityId": 0,
  "data": [
    "string"
  ]
}
```

Figure 26 API endpoint - Get Context Entity

5.3 Online Documentation Updates

The online documentation will be continuously updated during the development work. Data models can change at any time and therefore the APIs endpoint definition has to be updated.

The partners are responsible for their own content and are able to update their components on their own.

5.4 Privacy and Security Concepts

All LEAD components will implement a security protocol to give access to its resources as described in the sequence diagrams of section 3 and APIs description in section 5.1.

The authentication manager component will be based on the OAuth protocol[20] that addresses this issue by introducing an authorization layer and separating the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server and is issued a different set of credentials than those of the resource owner.

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token (a string denoting a specific scope, lifetime, and other access attributes). Access tokens are issued an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

Regarding the data governance and exchange between the LLs and the LEAD Platform, it will be secured based on the TLS (Transport Layer Security) [21] protocol, and each partner is responsible to make its models and components supporting the GDPR compliance.

5.5 Deployment Diagram

This section shows how one or more applications are realized on the infrastructure. This comprises the mapping of applications and components onto artifacts, and the mapping of the information used by these applications and components onto the underlying storage infrastructure.

Figure 27 depicts the Legend (based on ArchiMate 3 specification [3]) for the deployment diagram presented in Figure 28. The Cloud Computing side of the LEAD Platform enable to deploy one instance for all LL partners as depicted in Figure 28. The main location of the cloud infrastructure is located in the EU and can be reached from any other location. LLs will need only a Web browser to get a secured access to LEAD Platform through the secured channel connection (via the HTTPS protocol) and an authentication that is required to get access to the exposed components of LEAD Platform.

[20] https://swagger.io/docs/specification/authentication/oauth2/

[21] https://en.wikipedia.org/wiki/Transport_Layer_Security

Regarding the deployment of the components (such as the DDDAS, DSS or storages), and after the evaluation of the SBBs as presented in section 4, it has been decided to adopt Docker as a containerization technology for the deployment and management of the software systems lifecycle.

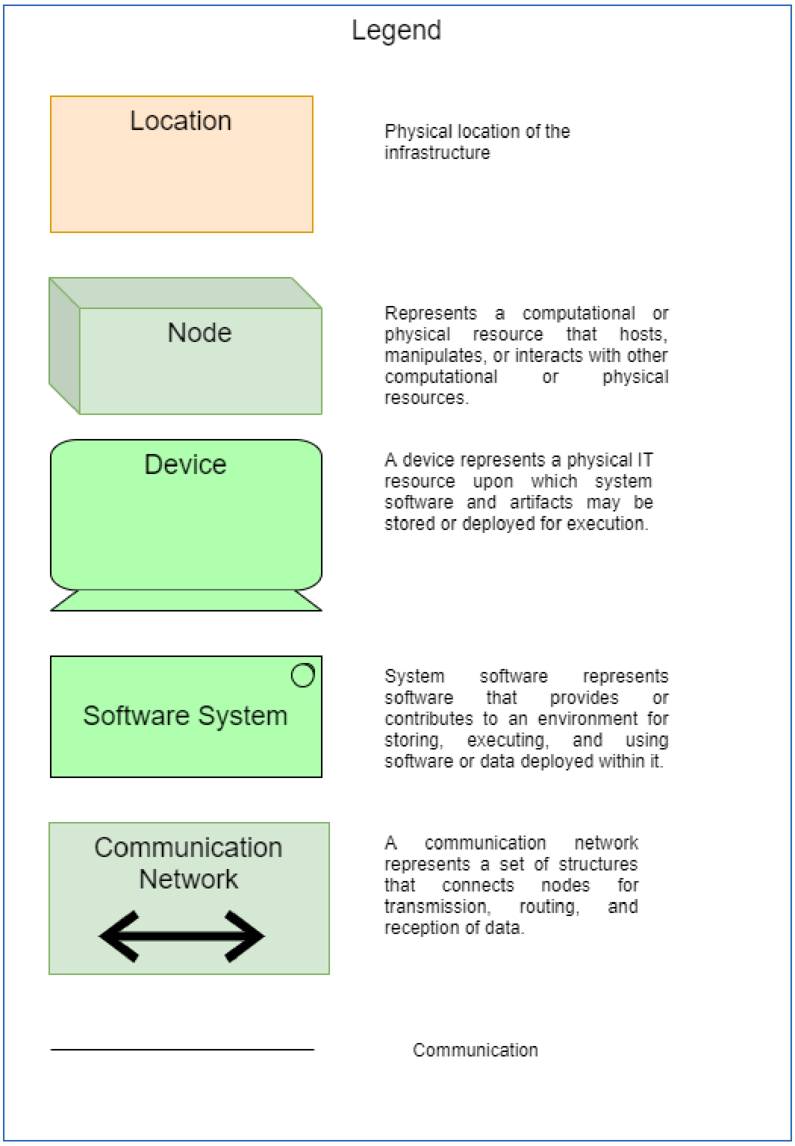

Legend
Location
Physical location of the infrastructure
Node
Represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.
Device
A device represents a physical IT resource upon which system software and artifacts may be stored or deployed for execution.
Software System
System software represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.
Communication Network
A communication network represents a set of structures that connects nodes for transmission, routing, and reception of data.
Communication

Figure 27 Legend for the Deployment Diagram

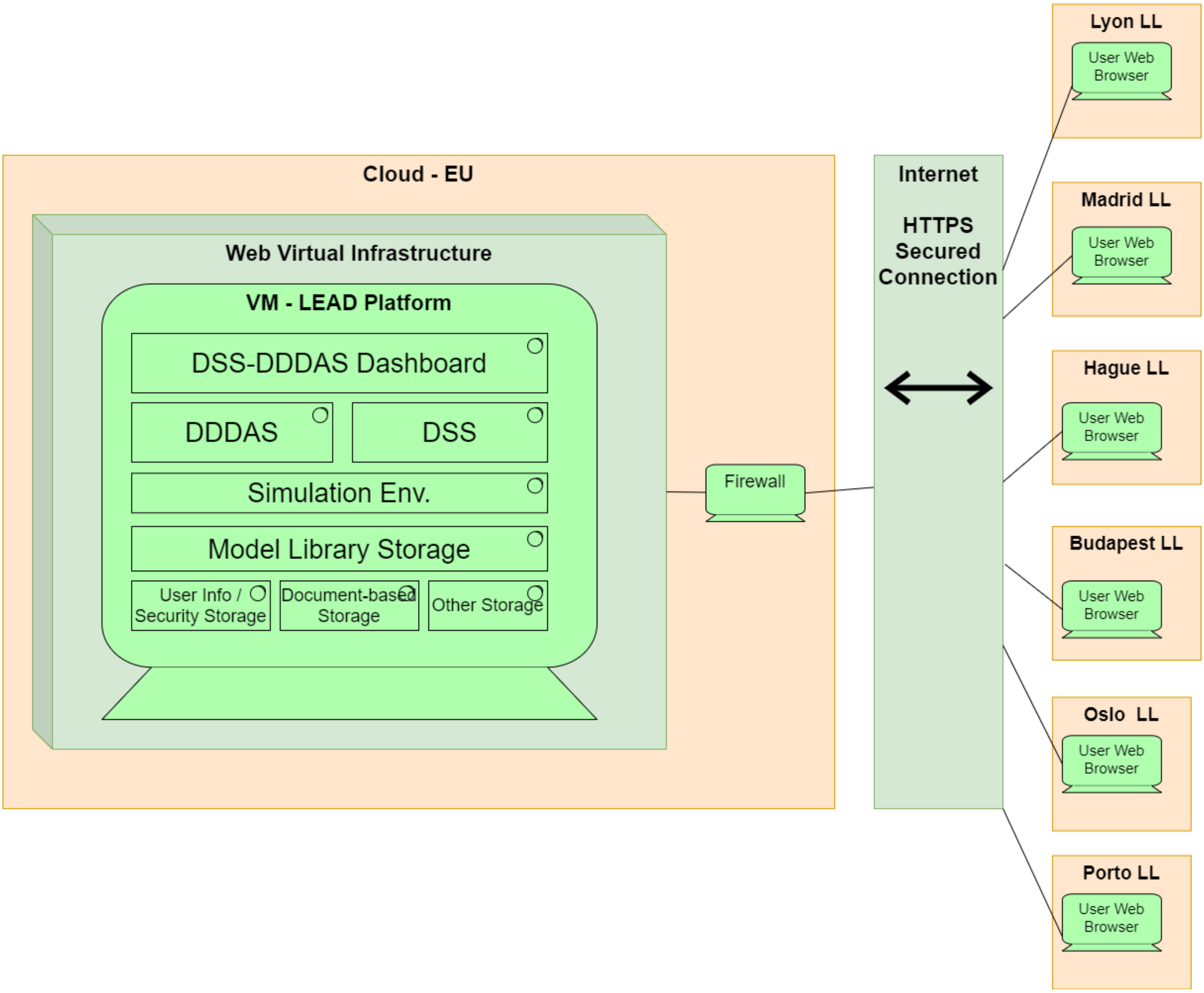


Figure 28 Deployment Diagram

6. Conclusion

This deliverable has defined the architecture of LEAD and its main generic process that is shared between all LLs. The global architecture defines all LEAD components and their interconnection. It is based on the main components defined in the description of action but at a more detailed level including their main interactions. Then a specific and detailed view of each component is described, along with its behaviour and functionalities, technical foundations and infrastructure details.

This deliverable will drive the work carried out in Tasks T2.2 (Digital Twin Models Library), T2.3 (Decision Support System and APIs), and T2.4 (DDDAS – Sensing – Data ingestion). Indeed, for T2.2, a detailed definition of its main consumer client in LEAD platform is defined (DDDAS) with a detailed specification about its targeted functionalities, data structure, and interactions with the model library. This will help to better design and consolidate the library of open source models that will be used in the LLs digital twins. Regarding T2.3 and T2.4, a detailed specification is initiated and provided in this deliverable, along with the UIs, APIs, components interactions, and a development plan that will enable the understanding, development and continuous improvement of the components.

It will invariably happen that due to further work during the project, it will lead to a revisiting of the architecture in an agile and practical approach. In this case, the changes will be clearly identified and recorded in the coming deliverables.

7. References

[1] The Open Group, TOGAF® Version 9.1. A Pocket Guide. The Open Group. December 2011.

[2] UN/CEFACT, "Multi Modal Transport Reference Data Model (UN/CEFACT SHIP/MMT-RDM)".

[3] UN/CEFACT, "UN/CEFACT Buy-Ship-Pay Reference Data Model BSP-RDM Version 1.0.," 2019.

[4] [Online]. Available: https://developers.google.com/transit/gtfs.

[5] [Online]. Available: http://www.transmodel-cen.eu/.

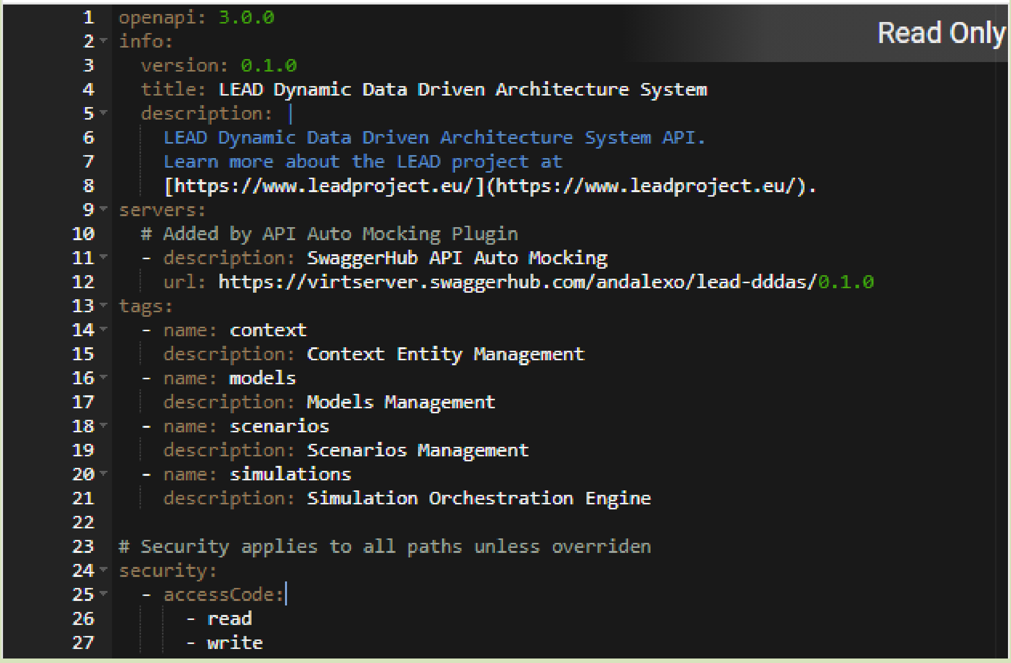
[6] A. 3. Specification, "Application Usage Viewpoint," [Online]. Available: https://pubs.opengroup.org/architecture/archimate3-doc/apdxc.html. [Accessed 10 02 2021].

[7] T. O. Group, "Implementation and Deployment Viewpoint".

8. Annex A

This annex depicts the current state (at the time of production of this deliverable) of each described API that is accessible in SwaggerHub as described in section 5.2.

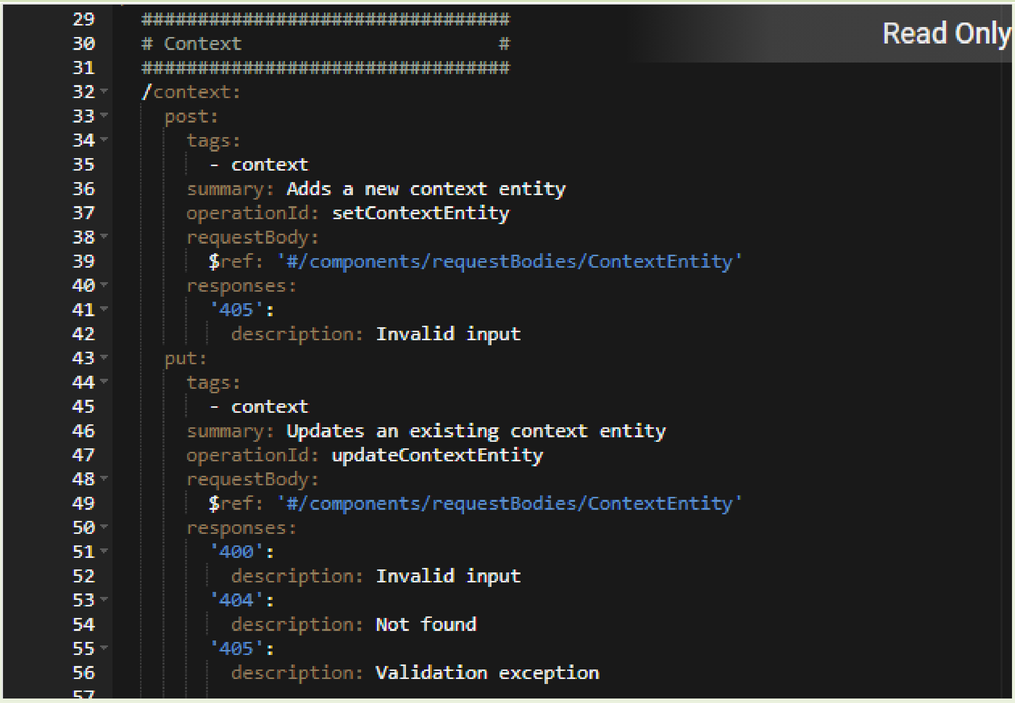
<table>
<tr><th>Component</th><th>Swagger Description</th></tr>
<tr><td>DDDAS</td><td>



Read Only

```yaml
openapi: 3.0.0
info:
  version: 0.1.0
  title: LEAD Dynamic Data Driven Architecture System
  description: |
    LEAD Dynamic Data Driven Architecture System API.
    Learn more about the LEAD project at
    [https://www.leadproject.eu/](https://www.leadproject.eu/).
servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/andalexo/lead-dddas/0.1.0
tags:
  - name: context
    description: Context Entity Management
  - name: models
    description: Models Management
  - name: scenarios
    description: Scenarios Management
  - name: simulations
    description: Simulation Orchestration Engine

# Security applies to all paths unless overriden
security:
  - accessCode:
      - read
      - write
```

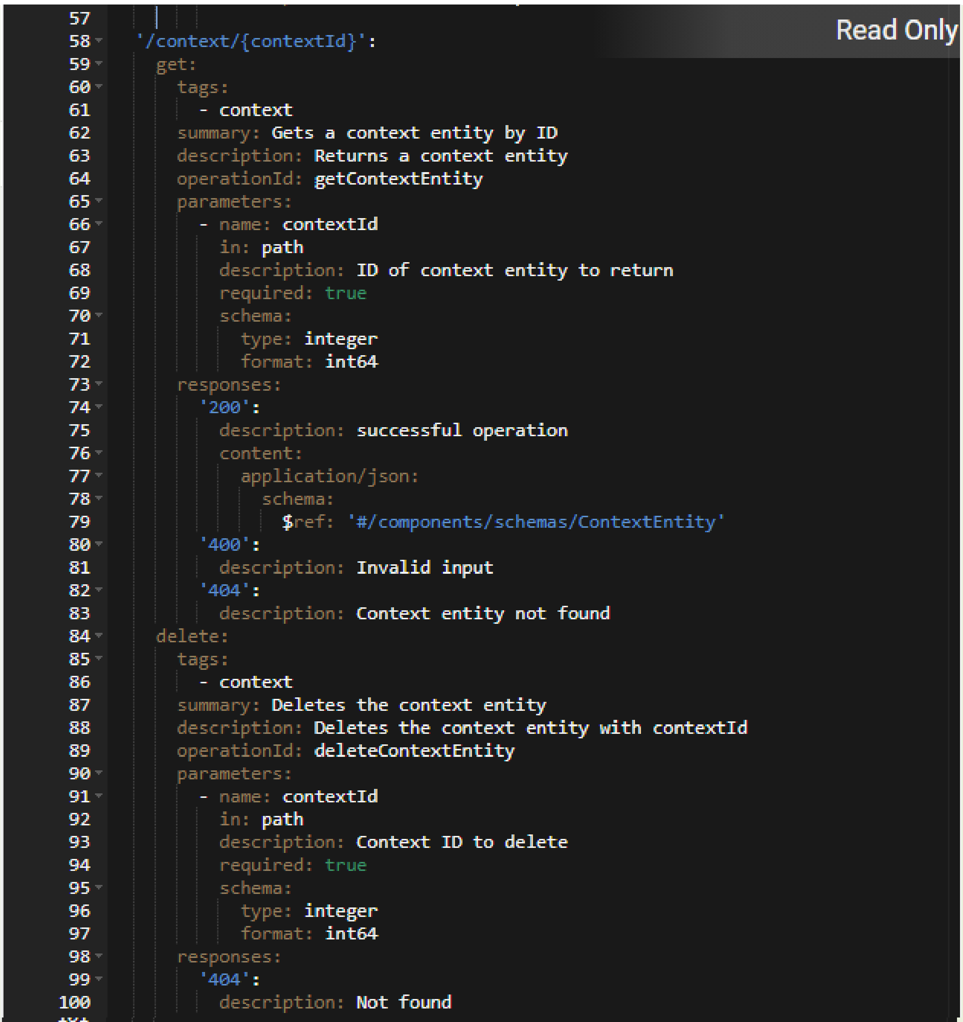
</td></tr>
<tr><td>Context Entity Manager</td><td>



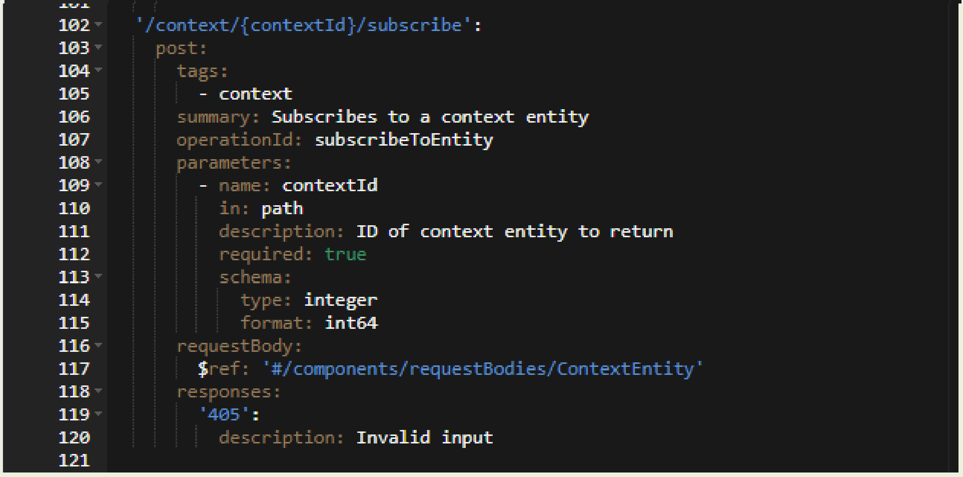
Read Only

```yaml
  #################################
  # Context                       #
  #################################
  /context:
    post:
      tags:
        - context
      summary: Adds a new context entity
      operationId: setContextEntity
      requestBody:
        $ref: '#/components/requestBodies/ContextEntity'
      responses:
        '405':
          description: Invalid input
    put:
      tags:
        - context
      summary: Updates an existing context entity
      operationId: updateContextEntity
      requestBody:
        $ref: '#/components/requestBodies/ContextEntity'
      responses:
        '400':
          description: Invalid input
        '404':
          description: Not found
        '405':
          description: Validation exception
```

</td></tr>
</table>

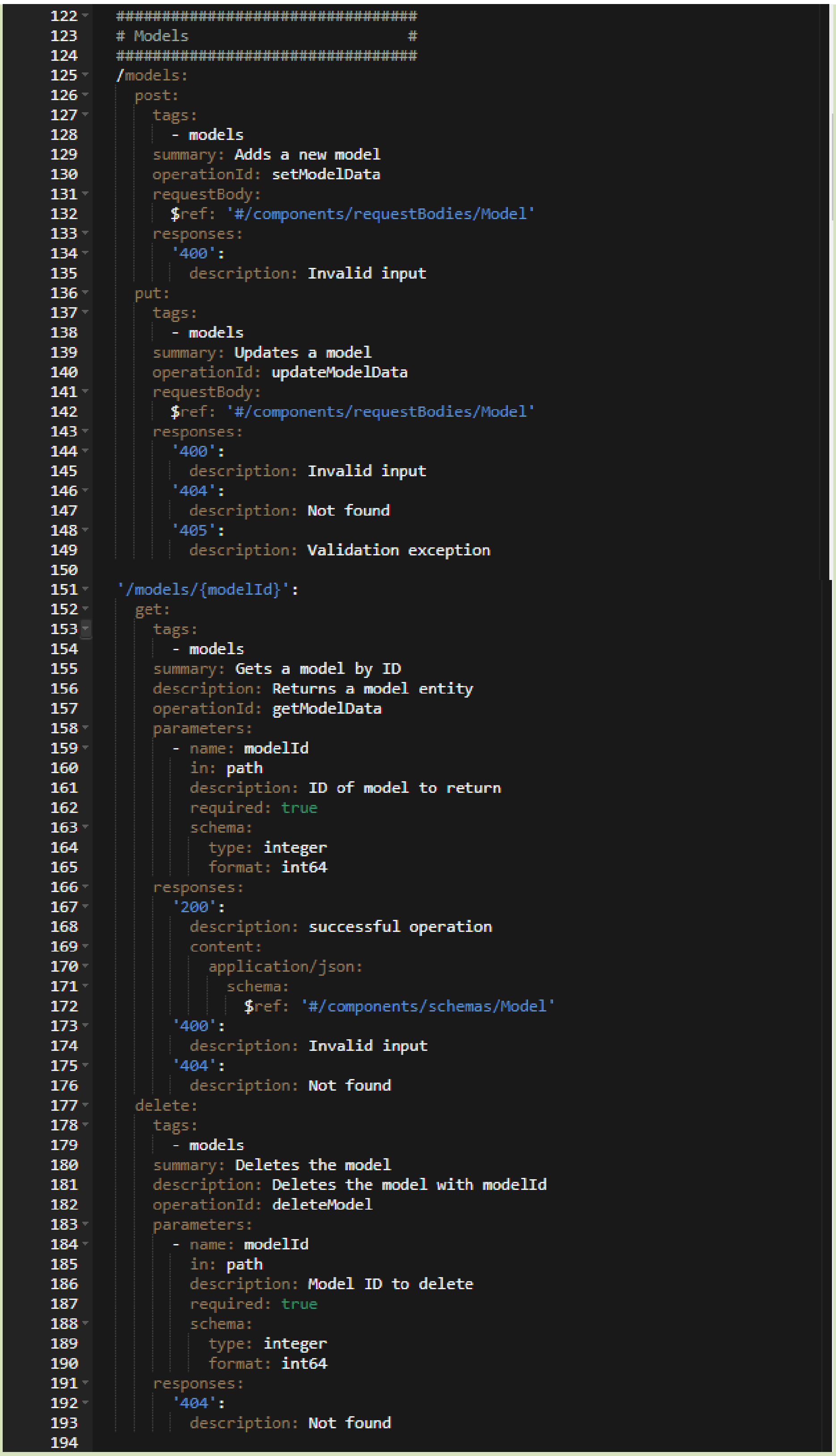


```yaml

  '/context/{contextId}':
    get:
      tags:
        - context
      summary: Gets a context entity by ID
      description: Returns a context entity
      operationId: getContextEntity
      parameters:
        - name: contextId
          in: path
          description: ID of context entity to return
          required: true
          schema:
            type: integer
            format: int64
      responses:
        '200':
          description: successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ContextEntity'
        '400':
          description: Invalid input
        '404':
          description: Context entity not found
    delete:
      tags:
        - context
      summary: Deletes the context entity
      description: Deletes the context entity with contextId
      operationId: deleteContextEntity
      parameters:
        - name: contextId
          in: path
          description: Context ID to delete
          required: true
          schema:
            type: integer
            format: int64
      responses:
        '404':
          description: Not found
```



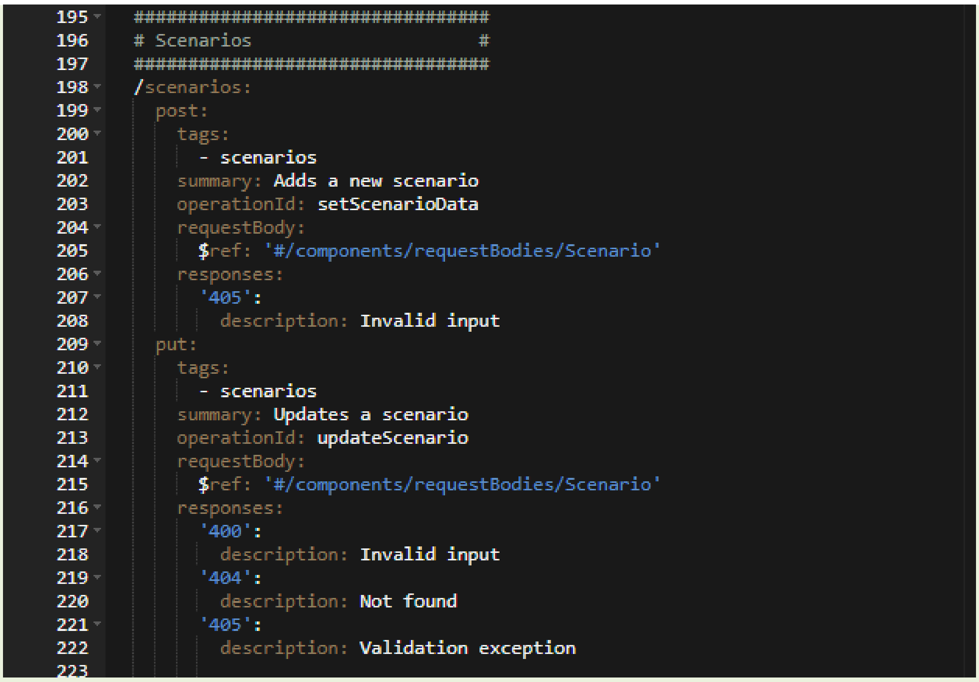
```yaml
  '/context/{contextId}/subscribe':
    post:
      tags:
        - context
      summary: Subscribes to a context entity
      operationId: subscribeToEntity
      parameters:
        - name: contextId
          in: path
          description: ID of context entity to return
          required: true
          schema:
            type: integer
            format: int64
      requestBody:
        $ref: '#/components/requestBodies/ContextEntity'
      responses:
        '405':
          description: Invalid input

```

Models Manager

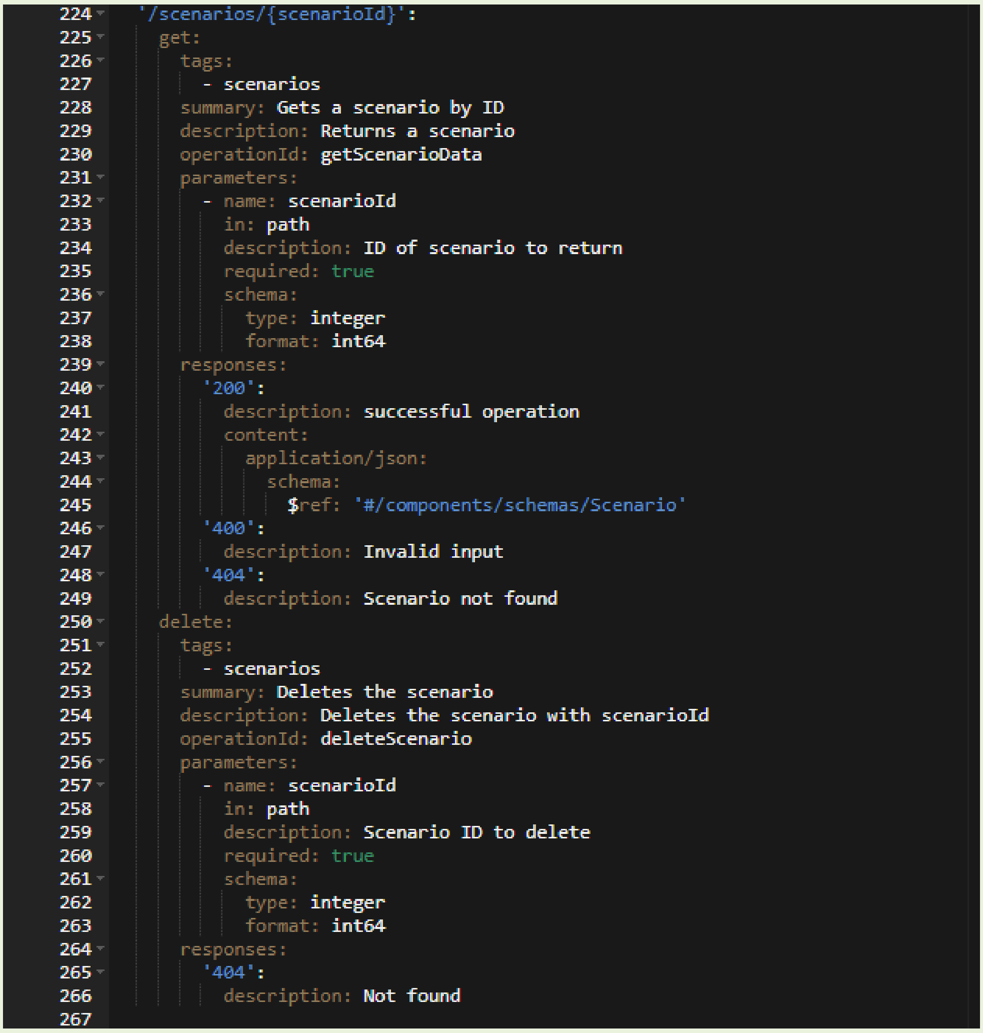


```yaml
#################################
# Models                        #
#################################
/models:
  post:
    tags:
      - models
    summary: Adds a new model
    operationId: setModelData
    requestBody:
      $ref: '#/components/requestBodies/Model'
    responses:
      '400':
        description: Invalid input
  put:
    tags:
      - models
    summary: Updates a model
    operationId: updateModelData
    requestBody:
      $ref: '#/components/requestBodies/Model'
    responses:
      '400':
        description: Invalid input
      '404':
        description: Not found
      '405':
        description: Validation exception

'/models/{modelId}':
  get:
    tags:
      - models
    summary: Gets a model by ID
    description: Returns a model entity
    operationId: getModelData
    parameters:
      - name: modelId
        in: path
        description: ID of model to return
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Model'
      '400':
        description: Invalid input
      '404':
        description: Not found
  delete:
    tags:
      - models
    summary: Deletes the model
    description: Deletes the model with modelId
    operationId: deleteModel
    parameters:
      - name: modelId
        in: path
        description: Model ID to delete
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '404':
        description: Not found

```

Scenario Manager

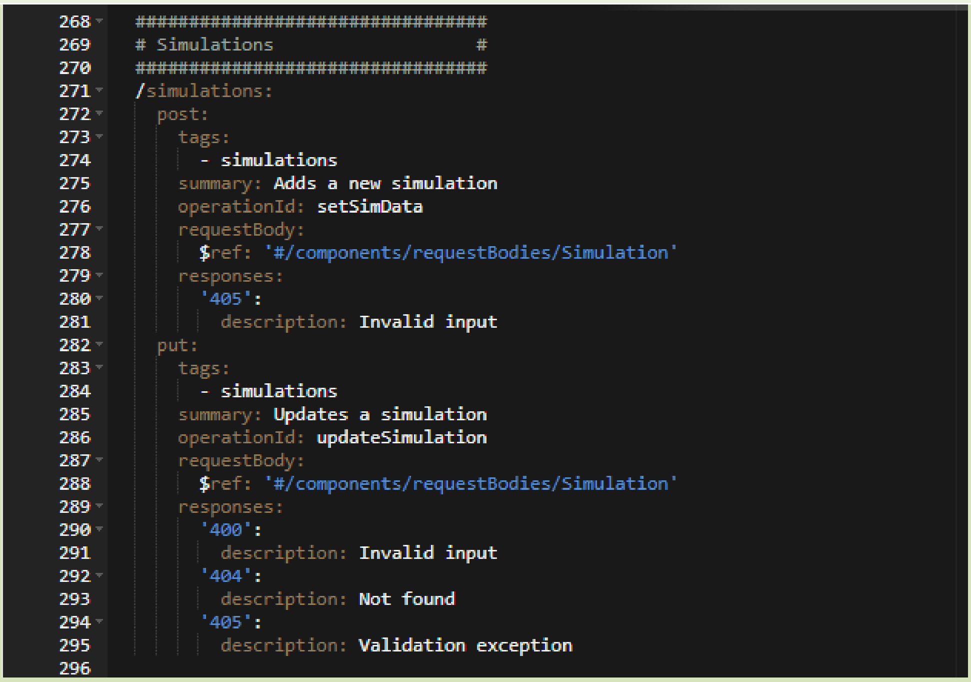


```
#################################
# Scenarios                     #
#################################
/scenarios:
  post:
    tags:
      - scenarios
    summary: Adds a new scenario
    operationId: setScenarioData
    requestBody:
      $ref: '#/components/requestBodies/Scenario'
    responses:
      '405':
        description: Invalid input
  put:
    tags:
      - scenarios
    summary: Updates a scenario
    operationId: updateScenario
    requestBody:
      $ref: '#/components/requestBodies/Scenario'
    responses:
      '400':
        description: Invalid input
      '404':
        description: Not found
      '405':
        description: Validation exception

```

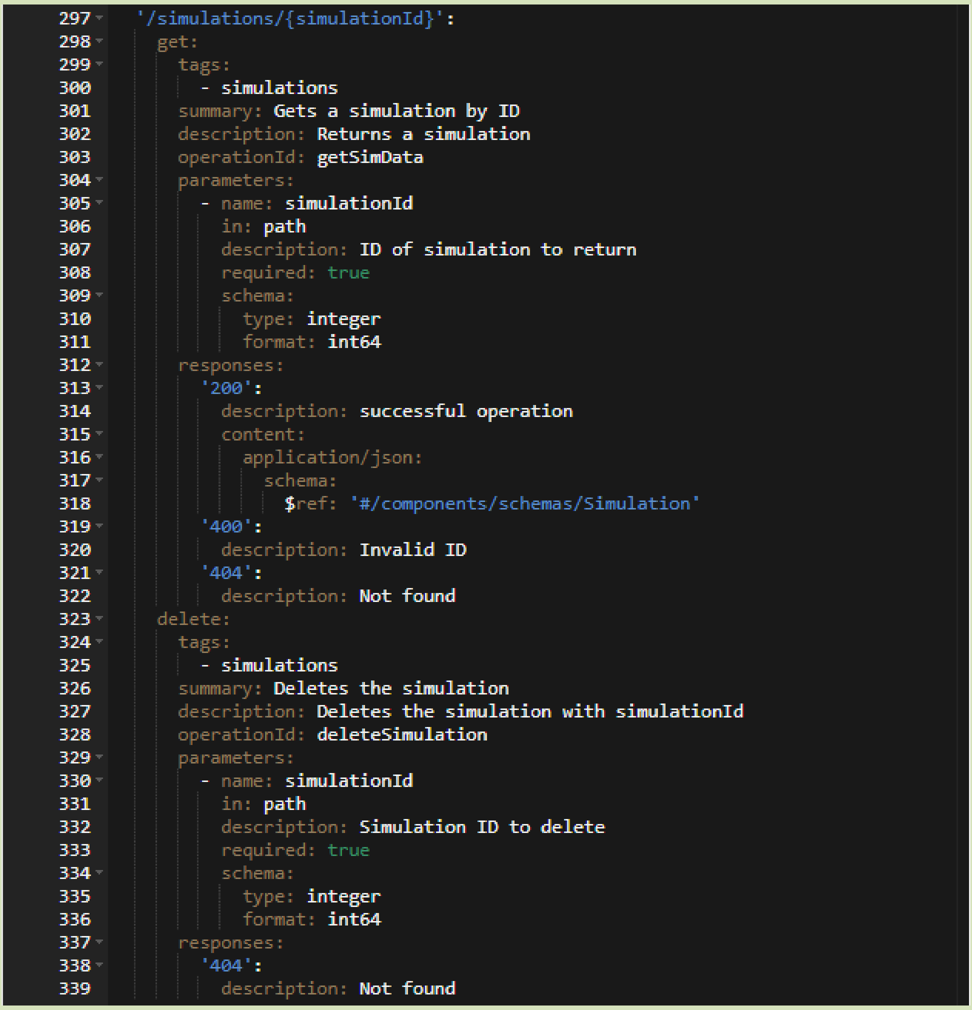


```
'/scenarios/{scenarioId}':
  get:
    tags:
      - scenarios
    summary: Gets a scenario by ID
    description: Returns a scenario
    operationId: getScenarioData
    parameters:
      - name: scenarioId
        in: path
        description: ID of scenario to return
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Scenario'
      '400':
        description: Invalid input
      '404':
        description: Scenario not found
  delete:
    tags:
      - scenarios
    summary: Deletes the scenario
    description: Deletes the scenario with scenarioId
    operationId: deleteScenario
    parameters:
      - name: scenarioId
        in: path
        description: Scenario ID to delete
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '404':
        description: Not found

```

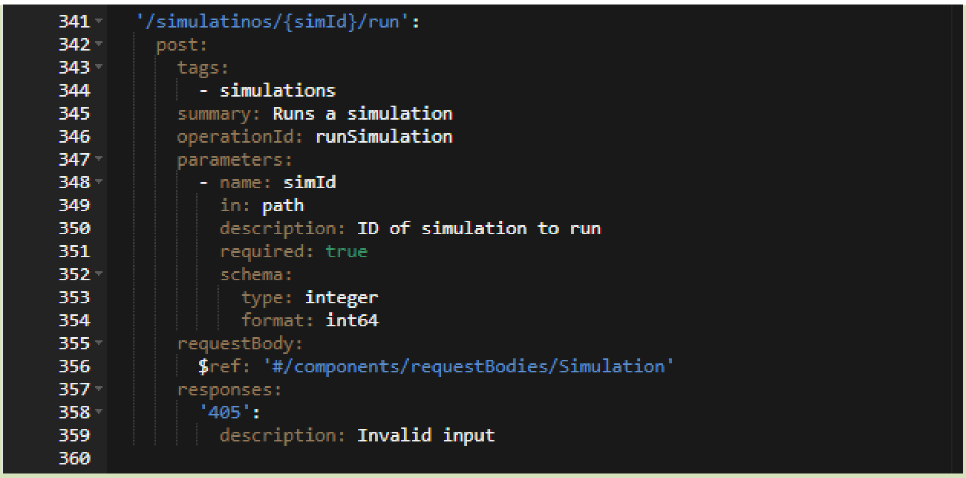
Simulation Orchestration Manager



```
#################################
# Simulations                   #
#################################
/simulations:
  post:
    tags:
      - simulations
    summary: Adds a new simulation
    operationId: setSimData
    requestBody:
      $ref: '#/components/requestBodies/Simulation'
    responses:
      '405':
        description: Invalid input
  put:
    tags:
      - simulations
    summary: Updates a simulation
    operationId: updateSimulation
    requestBody:
      $ref: '#/components/requestBodies/Simulation'
    responses:
      '400':
        description: Invalid input
      '404':
        description: Not found
      '405':
        description: Validation exception

```

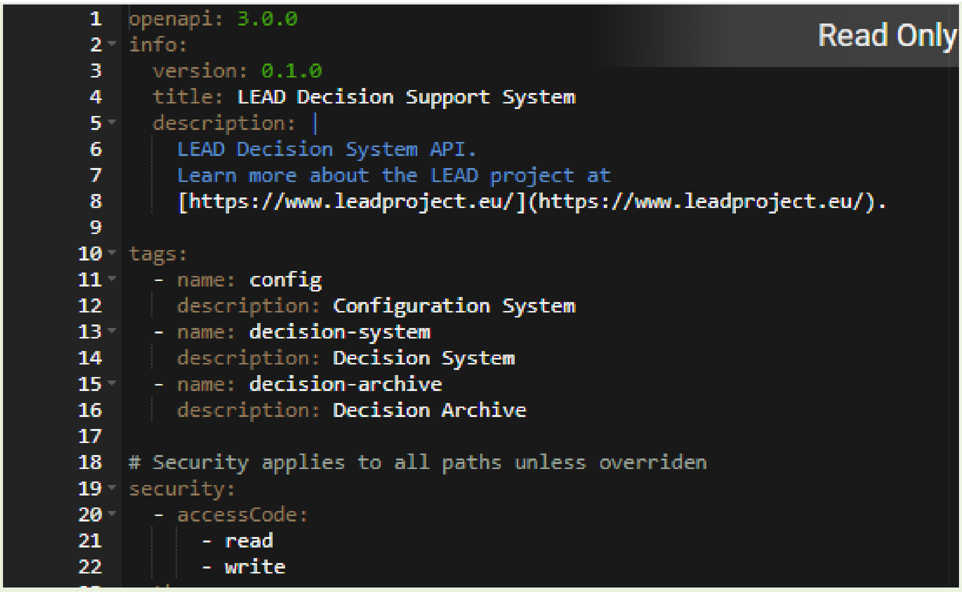


```
'/simulations/{simulationId}':
  get:
    tags:
      - simulations
    summary: Gets a simulation by ID
    description: Returns a simulation
    operationId: getSimData
    parameters:
      - name: simulationId
        in: path
        description: ID of simulation to return
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Simulation'
      '400':
        description: Invalid ID
      '404':
        description: Not found
  delete:
    tags:
      - simulations
    summary: Deletes the simulation
    description: Deletes the simulation with simulationId
    operationId: deleteSimulation
    parameters:
      - name: simulationId
        in: path
        description: Simulation ID to delete
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '404':
        description: Not found
```



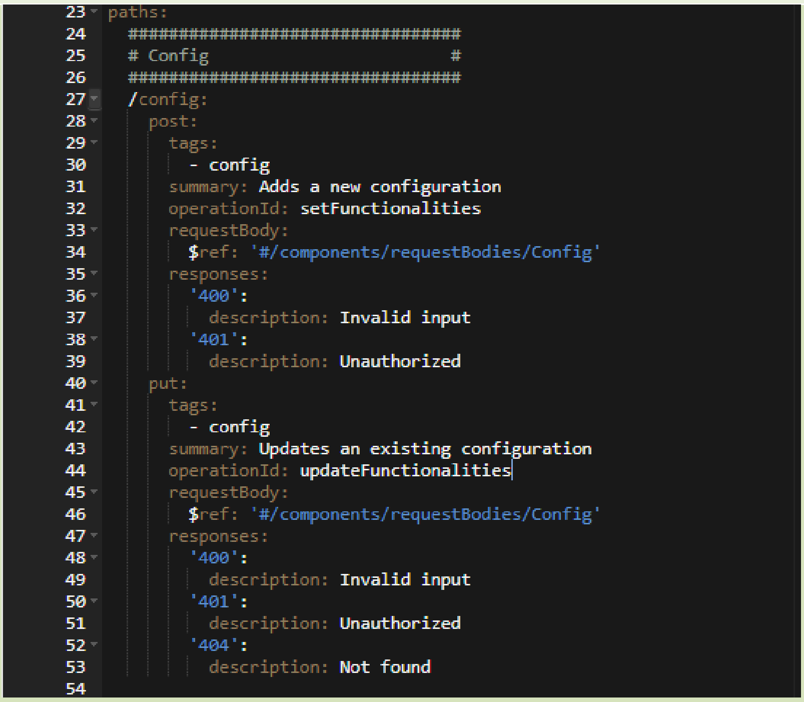
```
  '/simulatinos/{simId}/run':
    post:
      tags:
        - simulations
      summary: Runs a simulation
      operationId: runSimulation
      parameters:
        - name: simId
          in: path
          description: ID of simulation to run
          required: true
          schema:
            type: integer
            format: int64
      requestBody:
        $ref: '#/components/requestBodies/Simulation'
      responses:
        '405':
          description: Invalid input

```

DSS

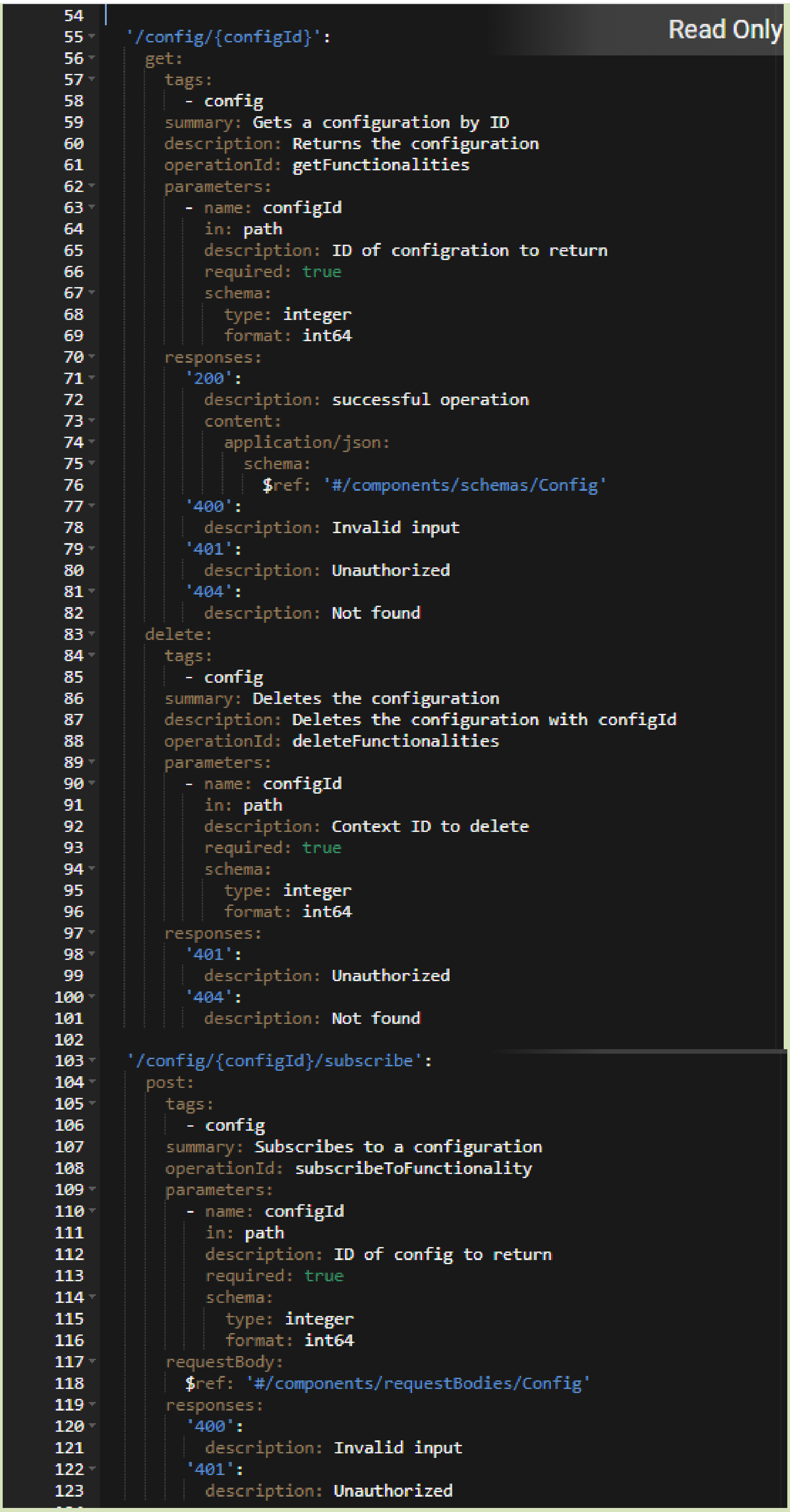


```
openapi: 3.0.0
info:
  version: 0.1.0
  title: LEAD Decision Support System
  description: |
    LEAD Decision System API.
    Learn more about the LEAD project at
    [https://www.leadproject.eu/](https://www.leadproject.eu/).

tags:
  - name: config
    description: Configuration System
  - name: decision-system
    description: Decision System
  - name: decision-archive
    description: Decision Archive

# Security applies to all paths unless overriden
security:
  - accessCode:
      - read
      - write
```

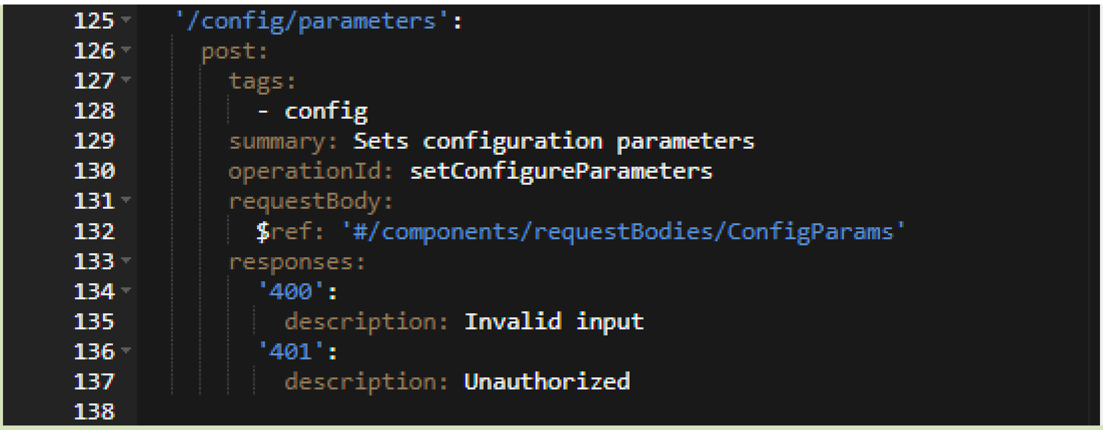
Configuration System



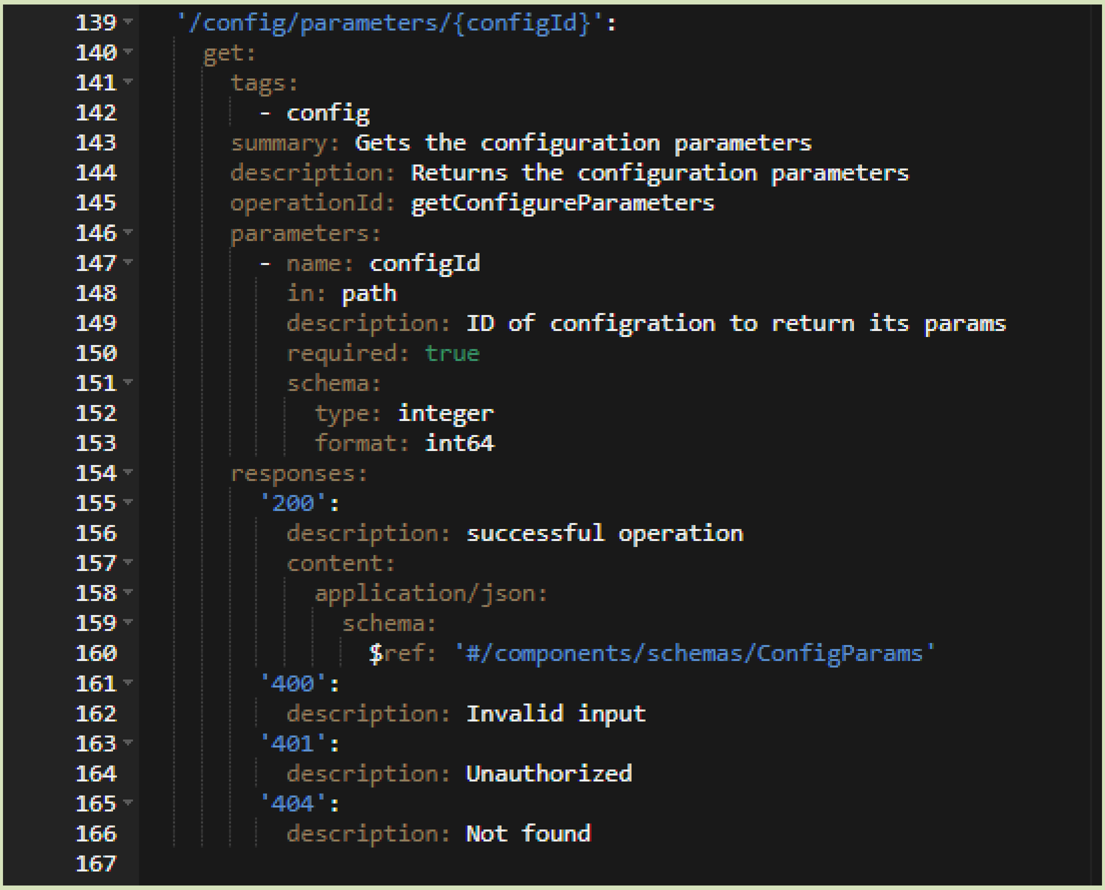
```
paths:
  ##################################
  # Config                         #
  ##################################
  /config:
    post:
      tags:
        - config
      summary: Adds a new configuration
      operationId: setFunctionalities
      requestBody:
        $ref: '#/components/requestBodies/Config'
      responses:
        '400':
          description: Invalid input
        '401':
          description: Unauthorized
    put:
      tags:
        - config
      summary: Updates an existing configuration
      operationId: updateFunctionalities
      requestBody:
        $ref: '#/components/requestBodies/Config'
      responses:
        '400':
          description: Invalid input
        '401':
          description: Unauthorized
        '404':
          description: Not found

```



```yaml

  '/config/{configId}':
    get:
      tags:
        - config
      summary: Gets a configuration by ID
      description: Returns the configuration
      operationId: getFunctionalities
      parameters:
        - name: configId
          in: path
          description: ID of configration to return
          required: true
          schema:
            type: integer
            format: int64
      responses:
        '200':
          description: successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Config'
        '400':
          description: Invalid input
        '401':
          description: Unauthorized
        '404':
          description: Not found
    delete:
      tags:
        - config
      summary: Deletes the configuration
      description: Deletes the configuration with configId
      operationId: deleteFunctionalities
      parameters:
        - name: configId
          in: path
          description: Context ID to delete
          required: true
          schema:
            type: integer
            format: int64
      responses:
        '401':
          description: Unauthorized
        '404':
          description: Not found

  '/config/{configId}/subscribe':
    post:
      tags:
        - config
      summary: Subscribes to a configuration
      operationId: subscribeToFunctionality
      parameters:
        - name: configId
          in: path
          description: ID of config to return
          required: true
          schema:
            type: integer
            format: int64
      requestBody:
        $ref: '#/components/requestBodies/Config'
      responses:
        '400':
          description: Invalid input
        '401':
          description: Unauthorized
```

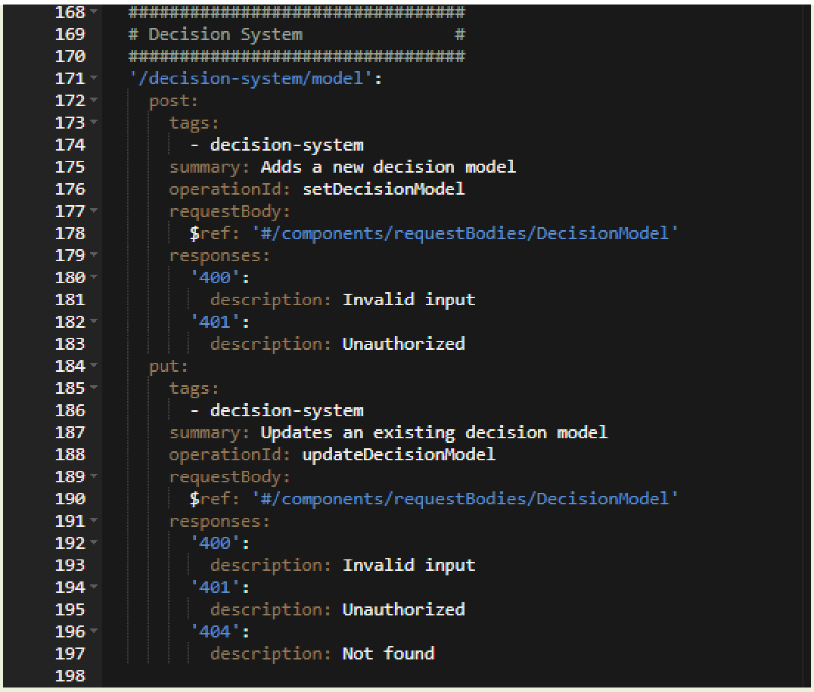


```yaml
'/config/parameters':
  post:
    tags:
      - config
    summary: Sets configuration parameters
    operationId: setConfigureParameters
    requestBody:
      $ref: '#/components/requestBodies/ConfigParams'
    responses:
      '400':
        description: Invalid input
      '401':
        description: Unauthorized

```

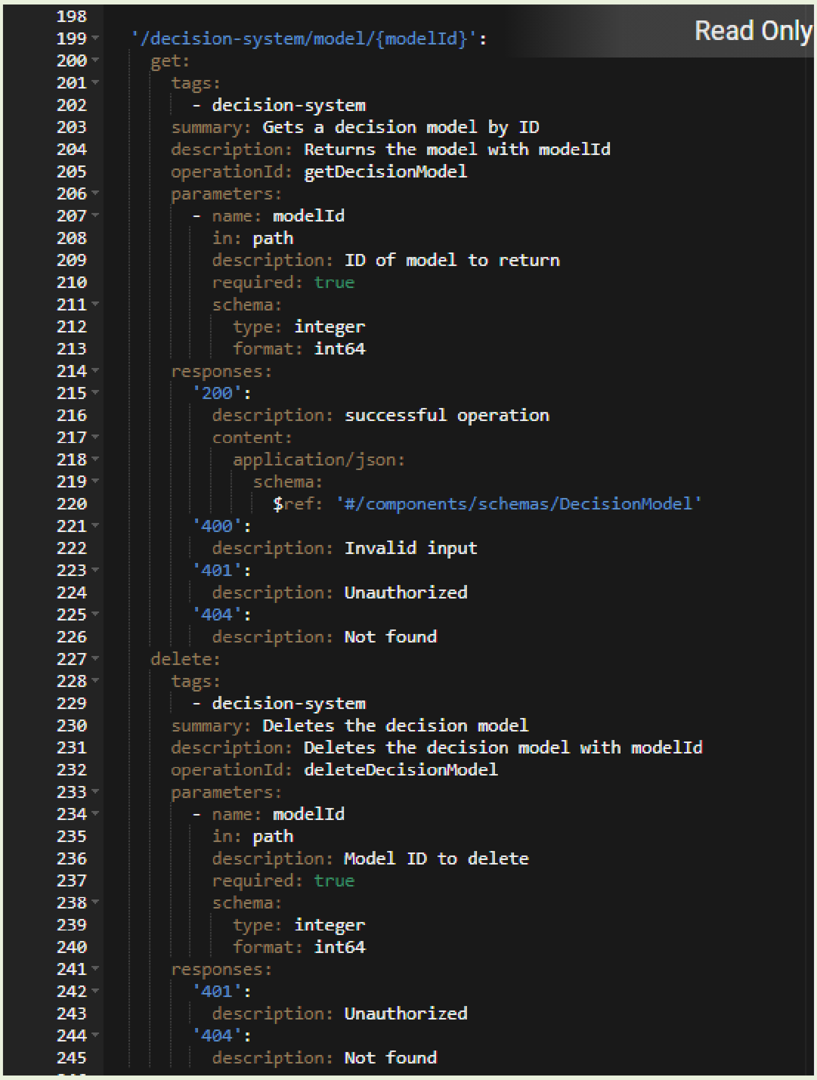


```yaml
'/config/parameters/{configId}':
  get:
    tags:
      - config
    summary: Gets the configuration parameters
    description: Returns the configuration parameters
    operationId: getConfigureParameters
    parameters:
      - name: configId
        in: path
        description: ID of configration to return its params
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ConfigParams'
      '400':
        description: Invalid input
      '401':
        description: Unauthorized
      '404':
        description: Not found

```

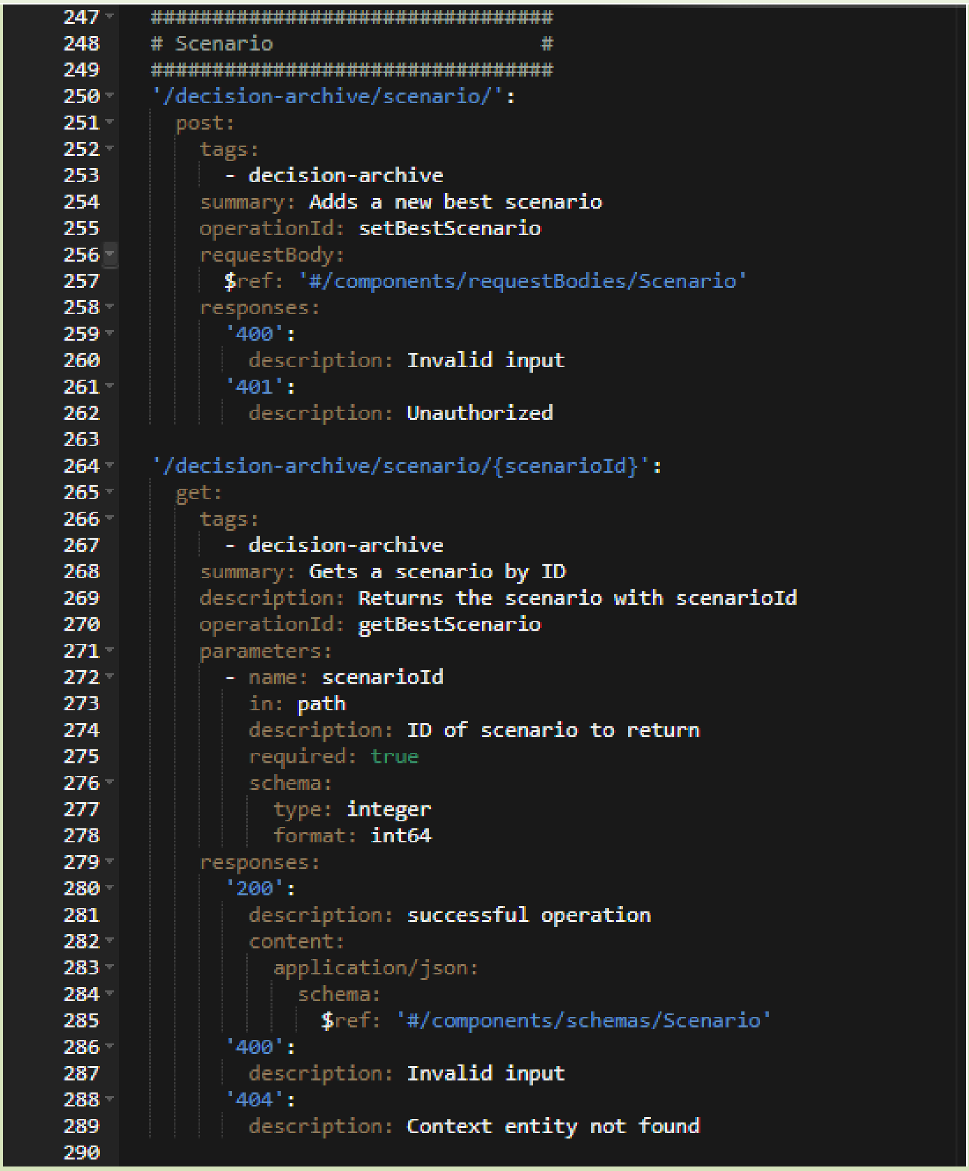
Decision System



```
##################################
# Decision System                #
##################################
'/decision-system/model':
  post:
    tags:
      - decision-system
    summary: Adds a new decision model
    operationId: setDecisionModel
    requestBody:
      $ref: '#/components/requestBodies/DecisionModel'
    responses:
      '400':
        description: Invalid input
      '401':
        description: Unauthorized
  put:
    tags:
      - decision-system
    summary: Updates an existing decision model
    operationId: updateDecisionModel
    requestBody:
      $ref: '#/components/requestBodies/DecisionModel'
    responses:
      '400':
        description: Invalid input
      '401':
        description: Unauthorized
      '404':
        description: Not found

```



```

'/decision-system/model/{modelId}':
  get:
    tags:
      - decision-system
    summary: Gets a decision model by ID
    description: Returns the model with modelId
    operationId: getDecisionModel
    parameters:
      - name: modelId
        in: path
        description: ID of model to return
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/DecisionModel'
      '400':
        description: Invalid input
      '401':
        description: Unauthorized
      '404':
        description: Not found
  delete:
    tags:
      - decision-system
    summary: Deletes the decision model
    description: Deletes the decision model with modelId
    operationId: deleteDecisionModel
    parameters:
      - name: modelId
        in: path
        description: Model ID to delete
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '401':
        description: Unauthorized
      '404':
        description: Not found
```

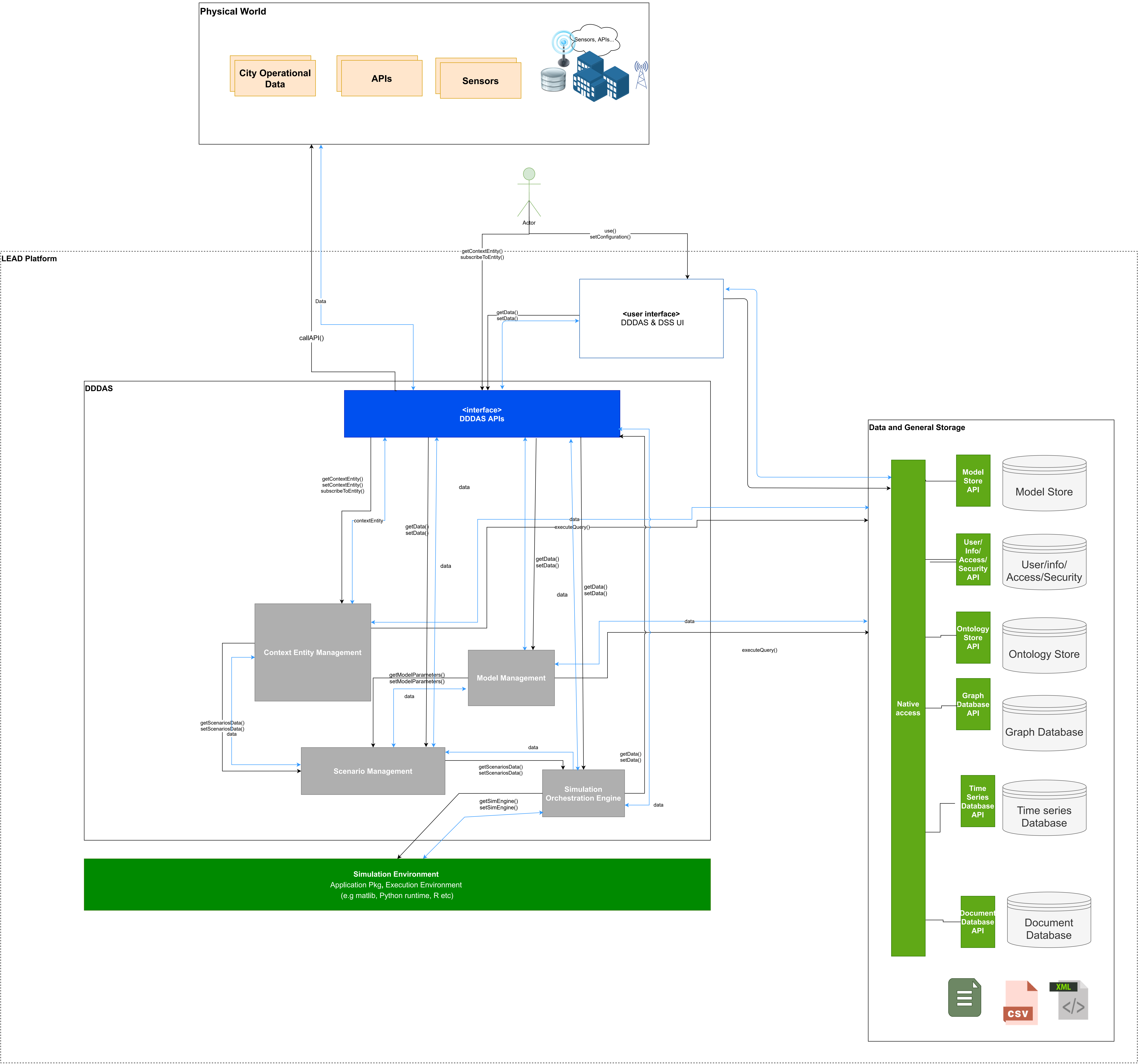
Decision Archive	

```
#################################
# Scenario                      #
#################################
'/decision-archive/scenario/':
  post:
    tags:
      - decision-archive
    summary: Adds a new best scenario
    operationId: setBestScenario
    requestBody:
      $ref: '#/components/requestBodies/Scenario'
    responses:
      '400':
        description: Invalid input
      '401':
        description: Unauthorized

'/decision-archive/scenario/{scenarioId}':
  get:
    tags:
      - decision-archive
    summary: Gets a scenario by ID
    description: Returns the scenario with scenarioId
    operationId: getBestScenario
    parameters:
      - name: scenarioId
        in: path
        description: ID of scenario to return
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Scenario'
      '400':
        description: Invalid input
      '404':
        description: Context entity not found

```

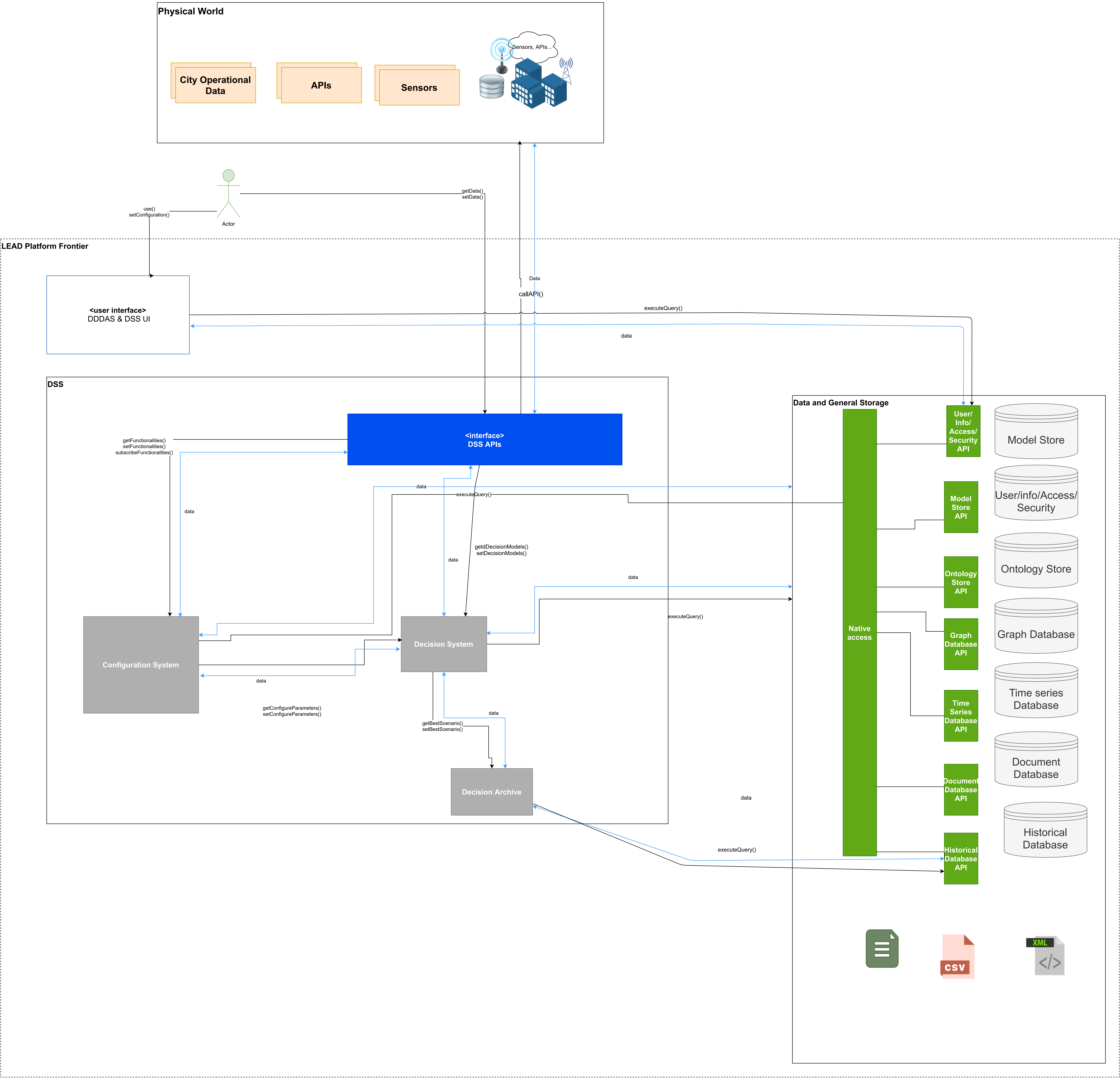
9. Annex B

The following diagrams present the main interactions of the DSS and the DDDAS components.

DDDAS - Component Interaction



DSS - Component Interaction


Physical World
City Operational Data
APIs
Sensors
Sensors, APIs...
Actor
use()
setConfiguration()
getData()
setData()
LEAD Platform Frontier
Data
callAPI()
<user interface>
DDDAS & DSS UI
executeQuery()
data
DSS
<interface>
DSS APIs
getFunctionalities()
setFunctionalities()
subscribeFunctionalities()
data
data
executeQuery()
data
getdDecisionModels()
setDecisionModels()
data
executeQuery()
Configuration System
Decision System
data
getConfigureParameters()
setConfigureParameters()
getBestScenario()
setBestScenario()
data
Decision Archive
data
executeQuery()
Data and General Storage
Native access
User/ Info/ Access/ Security API
Model Store API
Ontology Store API
Graph Database API
Time Series Database API
Document Database API
Historical Database API
Model Store
User/info/Access/ Security
Ontology Store
Graph Database
Time series Database
Document Database
Historical Database
CSV
XML